Tiny Neuron Network System based on RISC-V Processor: A Decentralized Approach for IoT Applications

Ngo-Doanh Nguyen, Duy-Hieu Bui, Xuan-Tu Tran
Information Technology Institute (ITI), Vietnam National University (VNU),
144 Xuan Thuy road, Cau Giay, Hanoi, Vietnam.
Corresponding author's email: hieubd@vnu.edu.vn

***Abstract*—The idea of Artificial Intelligence of Things (AIoT), a combination of Artificial Intelligence, especially Deep Learning, with edge devices in IoT networks, has recently emerged to reduce the communication cost, and server workloads and improve user experiences. This work presents our current research on a tiny neural network accelerator in a RISC-V System-on-Chip (SoC) to accelerate AI in IoT applications. This accelerator implements a variable-bit-precision MAC or a stochastic MAC to reduce hardware area and power consumption. The tiny AI accelerator has been successfully integrated into a low-power IoT SoC. The design has been implemented on FPGA technology using Arty A7 100T development kit with the operating frequency of 50MHz and the hardware resource of 12K slices. For the MNIST dataset, the accelerator with 8-bit precision can perform Convolutional Neural Network with an accuracy of 98.55%.**

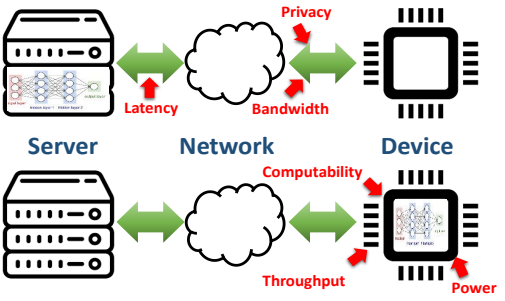

Fig. 1: Challenges for AIoT in centralization and decentralization.

I. INTRODUCTION

The Internet of Things (IoT) and the Artificial Intelligence of Things (AIoT) might be combined to make things smarter with better user experience. In fact, according to an annual report from Nokia in 2020 [1], the number of IoT devices will be sharply increased and be a critical part in various advanced applications such as AR, VR, and autonomous vehicles in the next five years. Most of these applications will be developed based on AI using Deep Learning (DL) to smoothly handle multiple operations such as collecting, processing and classifying data from the surrounding environment to act as a virtual assistant or transfer intelligent information to the cloud.

However, deep learning algorithms usually require a lot of computations which are often implemented on cloud computing servers with GPU accelerations. This server-oriented processing means that the AIoT devices need to send a large amount of data to the servers for processing and wait for the decision sent back from them. This approach increases the data processing latency and creates a long-delayed user experience. A new approach is edge-device-oriented processing which implements the deep learning algorithms in edge devices [2]. Edge devices are often lightweight devices with limited computation capacity and low memory footprint and require ultra-low-power consumption when running on battery. Therefore, implementing AI into IoT edge devices is challenging.

To overcome the challenges of DL at edge devices, many recent researches have balanced out two key components, the algorithm and device capability, to improve the quality of service and communication efficiency. On the one hand, edge devices need strong classification or decision-making algorithms such as Artificial Neural Network (ANN) and Convolutional Neural Network (CNN) [3], along with advanced learning algorithms to synchronize and unify data such as Federated Learning [4] and Split Learning [5], [6]. Because the difference in data size and distribution among heterogeneous devices is the main cause of reducing the classification/prediction accuracy or slowing down the convergence in learning for edge devices. On the other hand, edge devices also need to have hardware accelerations for AI computations while maintaining low cost, low memory footprint and low power to implement those advanced algorithms. Furthermore, at the edge, the devices tend to be small with limited hardware resources, which is one of the greatest challenges when applying decentralized DL to edge devices. Many works have tried to overcome this challenge by applying various techniques, such as Garofalo *et al.* [7], Lee *et al.* [8], Ando *et al.* [9] and so on. However, their works are partially optimized for implementing DL to edge devices.

As a premise, we determined to focus on completed system integration for AIoT devices with the ability of deep computing while maintaining low cost and low power consumption. In detail, this paper presents a configurable low-cost AI accelerator integrated into a System-on-Chip (SoC) with the evaluation of the performance of AI algorithms. The main contributions of this paper are listed in the following:

- Implementing a configurable single-neuron IoT-oriented AI accelerator adapted to different popular neural network algorithms such as ANN and CNN with two different computational Multiply-ACcumulate (MAC) engines, which are Variable-Precision-Bit-Width MAC and Stochastic MAC.
- The hardware has been implemented in PULPino RISC-

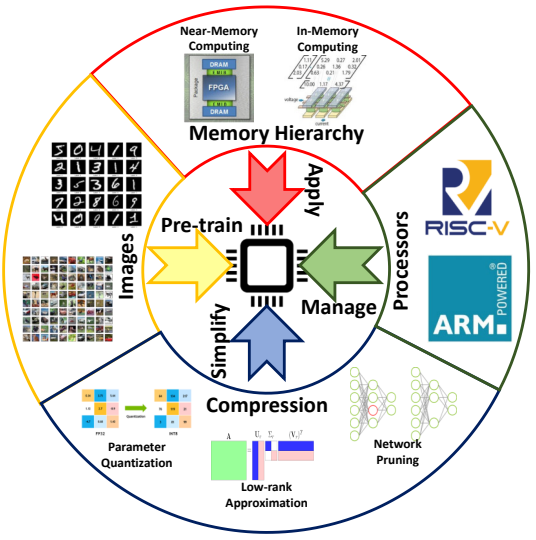

Fig. 2: Solutions for AIoT applications in decentralized processing.

V platform [10] with supporting Direct Memory Access (DMA) and a new data management to speed up the throughput of AI accelerator while saving memory footprint and hardware area.

- The performance of the tiny neural network system is evaluated on both FPGA and ASIC technology with multiple aspects from algorithms to computational engines. The results could be used to further optimize the AI accelerators for specific IoT applications as a case study.

The rest of this paper are organized as follows. Section II presents the related works of AI accelerators for IoT applications. Section III introduces our tiny neural network system with a RISC-V processor. In Section IV, the performance and energy consumption of our system with multiple configurations from algorithms to computational engines are evaluated. Finally, there are some conclusions and perspectives in Section V.

II. RELATED WORKS

The neural network architectures are considered the appealing solution for decentralized IoT applications. However, they require a huge number of computations which cost large hardware resources to design their accelerators and become the biggest problem for edge devices. Consequently, many solutions to embed or accelerate Deep Neural Networks (DNNs) for edge devices have been proposed in recent years. Figure 2 briefly summarizes the state-of-the-art solutions.

Optimizing microprocessor: This approach takes the advantages of a microprocessor in performing the complex computations for DL algorithms, which are not suitable for IoT edge devices because it uses software to implement the DL algorithms. The two favorite microprocessor architectures for IoT applications are the ARM Cortex-M and RISC-V. For example, Lai *et al.* in [11] and Garofalo *et al.* in [7] used the ARM core and RISC-V core, respectively. They not only increased the system's overall performance but also minimized the memory footprint. However, the processors are occupied with heavy computations. These computations can be further boosted by using a hardware accelerator.

Algorithm Compression: Another approach is to reduce a large number of processing data to increase throughput and save energy by using the compress algorithms. The bottleneck of applying DNN algorithms to edge devices is the continuous data transmission between the memory and the accelerator or processor. Haonan *et al.* in [12] and Chinthaka *et al.* in [13] used pruning algorithms to remove unnecessary neurons and weights while minimizing loss in accuracy and performance to solve this bottleneck. Along with pruning techniques, bitwidth quantization [14], [15] is also a favorite method to cut down memory storage and reduce computational complexity. However, the accuracy is usually dropped after applying these algorithms.

Data Pre-Training: In order to reduce the complexity of hardware implementations for DL IoT applications, pre-training methods are often used. These methods allow the edge devices to skip all training processes and store only the trained weights in memory. However, it requires a good dataset for training, and they can not adapt to the changes at run-time. For example, some image datasets widely used to evaluate the implementation are MNIST [16], CIPHAR-10 [17], ImageNet [18] and so on.

Optimal Memory Hierarchy: Finally, this approach tries to reduce the data transmission by moving the memory close to the accelerator or computational logic element. For example, Lee *et al.* in [8] proposed a tightly coupled data memory next to the logic element to reduce data movement. On the other hand, a different computational method utilizing computation-in-memory is introduced by Ando *et al.* in [9]. Since all the computations of DNN algorithm are executed directly inside the memory, the energy consumption for data transmission is greatly saved.

To cope with some of the above-mentioned problems, we have integrated an AI accelerator into PULPino platform, which uses a RISC-V processor with ultra-low-power features for IoT applications. The AI accelerator is used to improve the performance while keeping the power consumption as low as possible. As a result, the RISC-V processor can be used for other tasks. To optimize the AI operations, we use two low-cost MAC designs capable of varying bit precisions with a small drop of accuracy. In addition, a strategy for data movement management combined with a DMA module is proposed to increase the system throughput while reducing the memory footprint.

III. PROPOSED HARDWARE ARCHITECTURE

The overview hardware architecture of our System-on-Chip (SoC) based on RISC-V processor is illustrated in Fig. 3. In addition, Fig. 4 illustrates the hardware architecture of our AI accelerator with a core AI module, which uses only one physical neuron to minimize the hardware cost. This accelerator is then integrated into the SoC via the 32-bit AXI master and slave bus interfaces.

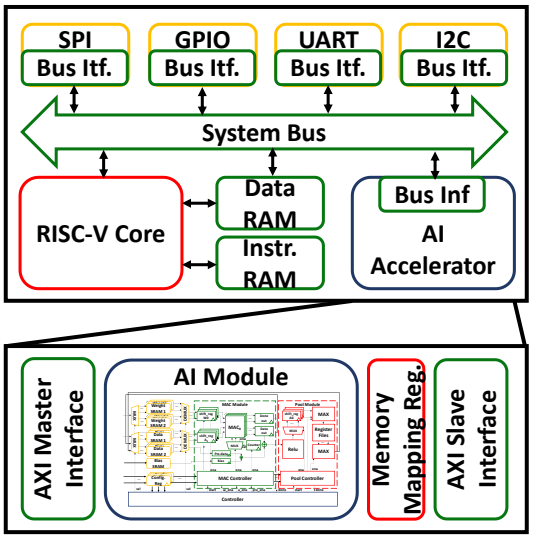

Fig. 3: The overview RISC-V system on chip with an AI accelerator

A. The MAC Module

The dot product is the main operation in neural network algorithms. First, each neuron receives the input data from the previous neurons. Then it calculates the dot product of these inputs along with their weights. In hardware design, this dot product is normally done by a Multiply-ACcumulate (MAC) module. Therefore, to accelerate the computation for neural networks and save energy for hardware systems in general, the optimization for the MAC is very important.

In this work, we have implemented two different MAC models, including the VBP MAC [8] and the SC MAC [19]. The selection of these designs is determined at the design phase. Each design has its own advantages and disadvantages. The details of the SC MAC and VBP MAC will be explained in the following subsections.

1) Stochastic MAC: The stochastic neuron's architecture is implemented following the work of Nguyen *et al.* in [19]. The idea of Stochastic MAC is to transform numbers back and forth between the stochastic domain and the binary domain. Stochastic numbers reduce hardware cost for arithmetic operations because they are performed using logic gates. The multiplication is converted to the AND operation in the stochastic domain. Therefore, complex and costly hardware multiplier can be removed in the stochastic domain. On the other hand, Stochastic numbers is not suitable for storage because of its lengthy bit stream. Therefore, we converted it into binary form after some computations to reduce the memory footprint. However, the throughput is lowered due to the transformation between two numeric domains.

2) Variable-Bit-Precision (VBP) MAC: For the VBP MAC, the design is developed based on the work of Lee *et al.* in [8]. The benefits of VBP MAC could be gained due to the serial bit process mechanism. In each clock cycle, a partial sum is accumulated consecutively following the appearance of bit '0' or '1' in weights' binary representation. The number of clock cycles required to get the final result is equal to the bit width used for representing the weight. However, the important point is that the algorithm behavior does not depend on the number of bits used for data representation, which is vital to implement variable bit widths. One weak point of this approach is that the number of clock cycles needed to complete one calculation increases according to the number of bits in binary representation. However, due to serial implementation, it is possible for three or more pairs of weights and activation values can be processed simultaneously, then all the computations will be done after the same number of clock cycles. In other words, this method is potential for parallel computations, which are essential for neural networks having a large number of neurons and weights.

B. Data Management

In this paper, the MAC module can support switching between parallel and serial processing for energy-saving purposes. It is determined by the characteristics of each stage in the neural network algorithms. For example, there are a few weights and shared among layers in the convolution step. The data can be read fast because they are stored in the local SRAMs. Therefore, in order to avoid the timing bubbles, the requirement for the throughput of MAC module is very high. It leads to the MAC module needing to operate in parallel to satisfy this requirement. In contrast, the fully connected operations have a large number of weights and each neuron uses its unique set. In order to store all the weights in one layer operation, the design needs a huge amount of hardware resources, which is not preferable for IoT devices. As a result, the optimal option is to transfer a subset of unique weights of neurons sequentially from DRAM to local SRAMs in the AI accelerator, which the readiness of data is slow. At this stage, the MAC module is preferred to compute in serial because of fewer switching activities and lower power consumption. In detail, the data flow of these operations in our SoC is illustrated in Fig. 5.

As shown in Fig. 5, we use two SRAMs for each pre-trained weight and the input data for data buffering, which can pre-load and store data to the next layer. So that the MAC module does not need to wait data fetched to the local SRAMs and can continuously move onto the next layer right away. In addition, multiple MAC engines are used in convolution operations because of the weight-sharing characteristic. The number of MAC engines is equal to the number of computational cycles for processing continuously without delay. For the fully connected operations, we only use one MAC engine because the bottleneck of this operation is loading weights from system memory to local SRAMs with limited bandwidth of data bus. If only one value is accessed per cycle, the time of pre-loading data of the next layer is equal to the processing time of the current layer. As a result, performing the MAC calculation in parallel takes the same time as performing in serial despite the increase in hardware area and memory footprint. Moreover, the goal is applied to edge devices, which prefer small memory footprint and low hardware cost.

Another key point is to avoid the timing of requesting data from system memory to the AI accelerator. Therefore,

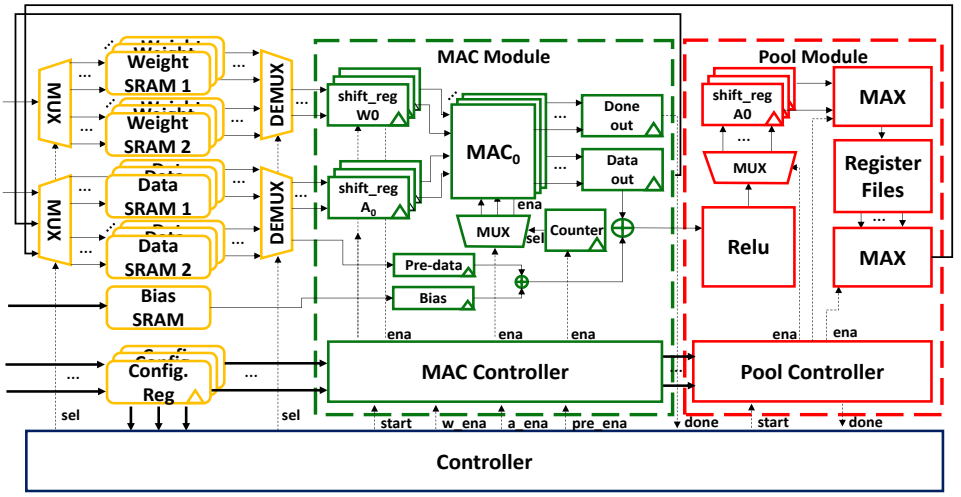

Fig. 4: The overall architecture of configurable low-cost AI Accelerator for AIoT application.

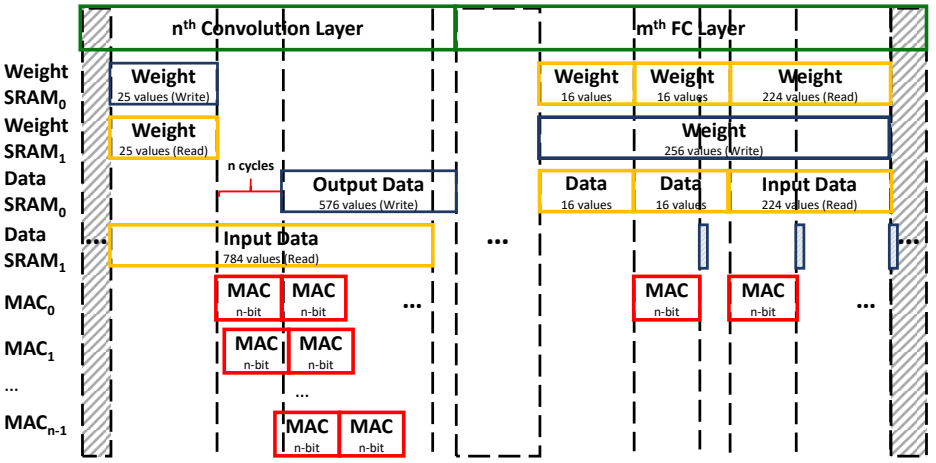

Fig. 5: Data diagram of configurable low-cost AI Accelerator with DMA support.

data management is implemented in the controller of the AI accelerator co-operated with the configurations received from the software program via the RISC-V processor. In detail, the software program firstly provides all the configurations of the algorithms via memory-mapped registers. After all the configurations are transferred, the accelerator automatically requests data from system memory with using Direct Memory Access (DMA) module. The RISC-V processor can be used for other tasks.

IV. EVALUATION

A. Experimental Setup

In this work, we implemented a full SoC in both FPGA and ASIC technology to evaluate the performance of our AI accelerator. The Arty A7 100T development kit is used for FPGA implementation, and the 65nm TSMC library is used for ASIC one. Moreover, we used the reference results from the software model, which was trained by the Pytorch framework. In this case, the MNIST dataset has been used for the training process. The MNIST is a standard dataset for neuron network development, which consists of 70,000 handwriting digits. Precisely, there are 60,000 images for training and 10,000 images for testing.

B. Experimental Result

In this work, we chose two bit widths for the weights and bias to verify the functionality of our hardware accelerator and our SoC, which are 16 bits and 8 bits. For each version of hardware implementation, 10,000 testing images in the MNIST dataset are used for evaluation. The inference results are compared to the images' actual labels to estimate the accelerator's accuracy. In reality, this load-and-check process is implemented in our SoC as an embedded program. Consequently, our system achieves nearly the same accuracy as the software version, which uses floating-point operations. It drops only 0.26% for the 16-bit accelerator and 0.24% for the 8-bit one with CNN, as shown in Figure 6.

Table I shows the ASIC implementation of our SoC operated at the frequency of 25MHz with multiple different configurations from algorithm to MAC model to bit operation. For ANN, the configuration of the neural network is a 3-layer network with one input layer of 784 neurons, one hidden

TABLE I: Hardware Implementation Result using The 65nm TSMC Technology.

Configuration	Full SoC			AI IP		MAC Module		Bias SRAM	Weight SRAM	Data SRAM	Others
	Perf. *(fps)*	Power *(mW)*	Area *(MGEs)*	Power *(mW)*	Area *(KGEs)*	Power *(mW)*	Area *(KGEs)*	Area *(KGEs)*	Area *(KGEs)*	Area *(KGEs)*	Area *(KGEs)*
ANN&16-bit SC @25MHz	0.12 (28×28*pix.*)	47.305 (100%)	3.867	31.292 (66.2%)	1,424	8.51e-02 (0.2%)	9.89	41.55 (1024×16*b*)[1]	664.91 (1024×256*b*)[1]	664.91 (1024×256*b*)[1]	43.08
ANN&8-bit SC @25MHz	30 (28×28*pix.*)	31.835 (100%)	3.163	15.819 (49.7%)	720.18	5.93e-02 (0.2%)	5.40	20.77 (1024×8*b*)[1]	332.46 (1024×128*b*)[1]	332.46 (1024×128*b*)[1]	29.10
ANN&16-bit VBP @25MHz	488 (28×28*pix.*)	47.314 (100%)	3.87	31.303 (66.2%)	1,426	9.43e-02 (0.2%)	12.33	41.55 (1024×16*b*)[1]	664.91 (1024×256*b*)[1]	664.91 (1024×256*b*)[1]	43.06
ANN&8-bit VBP @25MHz	968 (28×28*pix.*)	31.822 (100%)	3.164	15.805 (49.7%)	720.46	4.31e-02 (0.1%)	5.69	20.77 (1024×8*b*)[1]	332.46 (1024×256*b*)[1]	332.46 (1024×256*b*)[1]	29.08
CNN&16-bit VBP @25MHz	250 (28×28*pix.*)	21.104 (100%)	3.037	5.088 (24.1%)	594.61	1.893 (9.0%)	141.41	18.78 (128×16*b*)[2]	74.83 (512×16*b*)[2]	297.64 (2048×16*b*)[2]	61.91
CNN&8-bit VBP @25MHz	250 (28×28*pix.*)	19.750 (100%)	2.905	3.737 (18.9%)	461.74	0.762 (3.9%)	57.09	9.99 (128×8*b*)[2]	39.66 (512×8*b*)[2]	297.64 (2048×16*b*)[2]	57.37

1: This memory is generated by using ARM Memory Compiler.
2: This memory is generated by using Synopsys Design Compiler.

TABLE II: The Experimental Comparison between FPGA Implementation.

	Our Work				Zhang'17	Chen'15	Sanka.'09
	16-bit SoC	16-bit AI IP	8-bit SoC	8-bit AI IP	8-bit AI IP	32-bit AI IP	16-bit Coprocessor
FPGA Board	Arty A7 100T				Virtex-7 VC709	Virtex -7 VC707	Virtex-5
Frequency	50*MHz*	50*MHz*	50*MHz*	50*MHz*	100*MHz*	100*MHz*	230*MHz*
Slice LUTs	42792 (67.50%)	18826 (29.69%)	31818 (50.19%)	7842 (12.37%)	316250 (73%)	186251 (61.3%)	35263 (17%)
Slice registers	24218 (19.10%)	8701 (6.86%)	19765 (15.59%)	4248 (3.35%)	231165 (37%)	205704 (33.87%)	39501 (19%)
Slice	12749 (80.44%)	5418 (34.18%)	9931 (62.66%)	2413 (15.22%)	-	-	-
Block RAM	72.5 (53.70%)	2.5 (1.85%)	72.5 (53.70%)	2.5 (1.85%)	1508 (51%)	1024 (50%)	3 (1%)
DSP	0 (0%)	0 (0%)	0 (0%)	0 (0%)	3130 (87%)	2240 (80%)	107 (55%)

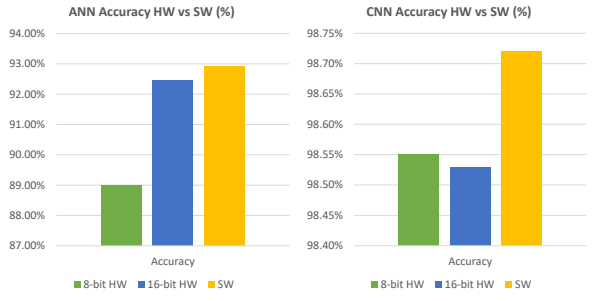

Fig. 6: Overall accuracy of our SoC in case of MNIST dataset with CNN and ANN algorithm.

layer of 64 neurons, and one output layer of 10 outputs. In this case, we only use 1 MAC module (both SC and VBP model) to perform the dot product calculation. As shown in Table I, by using the VBP model, the hardware is capable of processing at 968 fps (28×28 pixels), which dominates the SC one. However, the SC model has an advantage in the hardware area, which is more suitable for small AIoT systems. On the other hand, the configurations of CNN change into 2 convolution layers and 3 fully-connected layers of 128, 84 and 10 neurons, respectively. Furthermore, because of the characteristic of CNN, which has a large number of MAC operations compared to ANN, we add more MAC modules to perform parallel computations to maintain the throughput. In detail, the 16-bit version uses 8 MACs doubled to the 8-bit one, because it needs to cover the processing time of the MAC itself when the bit width of activation values is increased from 8 bits to 16 bits, as the explanation in Section III. As a result, the hardware area of the MAC module used in CNN is larger than the one used in ANN.

For FPGA implementation, the results are shown in Table

II, along with the other results of previous works. For IoT applications, our accelerator with a single physical neuron achieves the smallest hardware area compared to previous works such as Sankaradas *et al.* in [20], Chen *et al.* in [21] and Zhang *et al.* in [22]. On the other hand, their works are only implemented as a single AI IP or co-processor. In this work, we have further integrated the accelerator into a complete SoC with peripherals, system memory and interconnect. As a result, our hardware implementation of full SoC costs about 12k slices, which takes 80.44% hardware resource in FPGA Arty A7 100T devkit.

V. CONCLUSIONS

In this work, we have introduced a full System-on-Chip based on RISC-V processors integrated a configurable AI accelerator for AIoT applications in both FPGA and ASIC. This accelerator utilizes only one physical neuron for processing neural network algorithm, which aims to reduce the hardware area and energy consumption while maintaining its accuracy. With the full-system evaluation of multiple aspects from the neural network algorithms to MAC engines and to bit precisions of computation, it could become a case study to optimize AI accelerators for IoT applications further.

REFERENCES

[1] "Ericsson mobility report," https://www.ericsson.com/en/mobility-report, accessed: 2021-09-13.

[2] Y. Sun, H. Ochiai, and H. Esaki, "Decentralized deep learning for multi-access edge computing: A survey on communication efficiency and trustworthiness," *IEEE Transactions on Artificial Intelligence*, pp. 1–1, 2021.

[3] K. O'Shea and R. Nash, "An introduction to convolutional neural networks," *CoRR*, vol. abs/1511.08458, 2015.

[4] J. Konečný, H. B. McMahan, F. X. Yu, P. Richtárik, A. T. Suresh, and D. Bacon, "Federated learning: Strategies for improving communication efficiency," *CoRR*, vol. abs/1610.05492, 2016.

[5] P. Vepakomma, O. Gupta, T. Swedish, and R. Raskar, "Split learning for health: Distributed deep learning without sharing raw patient data," *CoRR*, vol. abs/1812.00564, 2018.

[6] C. Thapa, M. A. P. Chamikara, and S. Camtepe, "Splitfed: When federated learning meets split learning," *CoRR*, vol. abs/2004.12088, 2020.

[7] A. Garofalo, M. Rusci, F. Conti, D. Rossi, and L. Benini, "Pulp-nn: accelerating quantized neural networks on parallel ultra-low-power risc-v processors," *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, vol. 378, no. 2164, p. 20190155, Dec 2019.

[8] J. Lee, C. Kim, S. Kang, D. Shin, S. Kim, and H.-J. Yoo, "Unpu: An energy-efficient deep neural network accelerator with fully variable weight bit precision," *IEEE Journal of Solid-State Circuits*, vol. 54, no. 1, pp. 173–185, 2019.

[9] K. Ando, K. Ueyoshi, K. Orimo, H. Yonekawa, S. Sato, H. Nakahara, S. Takamaeda-Yamazaki, M. Ikebe, T. Asai, T. Kuroda, and M. Motomura, "Brein memory: A single-chip binary/ternary reconfigurable in-memory deep neural network accelerator achieving 1.4 tops at 0.6 w," *IEEE Journal of Solid-State Circuits*, vol. 53, no. 4, pp. 983–994, 2018.

[10] "Pulpino," https://github.com/pulp-platform/pulpino, accessed: 2015-12.

[11] L. Lai, N. Suda, and V. Chandra, "CMSIS-NN: efficient neural network kernels for arm cortex-m cpus," *CoRR*, vol. abs/1801.06601, 2018.

[12] H. Guo, S. Li, B. Li, Y. Ma, and X. Ren, "A new learning automata-based pruning method to train deep neural networks," *IEEE Internet of Things Journal*, vol. 5, no. 5, pp. 3263–3269, 2018.

[13] C. Gamanayake, L. Jayasinghe, B. K. K. Ng, and C. Yuen, "Cluster pruning: An efficient filter pruning method for edge ai vision applications," *IEEE Journal of Selected Topics in Signal Processing*, vol. 14, no. 4, pp. 802–816, 2020.

[14] B. Reagen, P. Whatmough, R. Adolf, S. Rama, H. Lee, S. K. Lee, J. M. Hernández-Lobato, G.-Y. Wei, and D. Brooks, "Minerva: Enabling low-power, highly-accurate deep neural network accelerators," in *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, 2016, pp. 267–278.

[15] J. Choi, Z. Wang, S. Venkataramani, P. I. Chuang, V. Srinivasan, and K. Gopalakrishnan, "PACT: parameterized clipping activation for quantized neural networks," *CoRR*, vol. abs/1805.06085, 2018.

[16] L. Deng, "The mnist database of handwritten digit images for machine learning research," *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp. 141–142, 2012.

[17] A. Krizhevsky, V. Nair, and G. Hinton, "Cifar-10 (canadian institute for advanced research)."

[18] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE conference on computer vision and pattern recognition*. Ieee, 2009, pp. 248–255.

[19] D.-A. Nguyen, H.-H. Ho, D.-H. Bui, and X.-T. Tran, "An efficient hardware implementation of artificial neural network based on stochastic computing," in *2018 5th NAFOSTED Conference on Information and Computer Science (NICS)*, 2018, pp. 237–242.

[20] M. Sankaradas, V. Jakkula, S. Cadambi, S. Chakradhar, I. Durdanovic, E. Cosatto, and H. P. Graf, "A massively parallel coprocessor for convolutional neural networks," in *2009 20th IEEE International Conference on Application-specific Systems, Architectures and Processors*, 2009, pp. 53–60.

[21] C. Zhang, P. Li, G. Sun, Y. Guan, B. Xiao, and J. Cong, "Optimizing fpga-based accelerator design for deep convolutional neural networks." New York, NY, USA: Association for Computing Machinery, 2015.

[22] X. Zhang, A. Ramachandran, C. Zhuge, D. He, W. Zuo, Z. Cheng, K. Rupnow, and D. Chen, "Machine learning on fpgas to face the iot revolution," in *2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2017, pp. 819–826.