IMAGE LICENSED BY INGRAM PUBLISHING

Nanoscale Accelerators for Artificial Neural Networks

Arithmetic Design, Analysis and ASIC Implementations

ARTIFICIAL NEURAL NETWORKS (ANNs) are usually implemented in accelerators to achieve efficient processing of inference; the hardware implementation of an ANN accelerator requires careful consideration on overhead metrics (such as delay, energy and area) and performance (usually measured by the accuracy). This paper considers the ASIC-based accelerator from arithmetic design considerations. The feasibility of using different schemes (parallel, serial and hybrid arrangements) and different types of arithmetic computing (floating-point, fixed-point and stochastic computing) when implementing multilayer perceptrons (MLPs) are considered.

Digital Object Identifier 10.1109/MNANO.2022.3208757
Date of current version: 20 January 2023

FARZAD NIKNIA, ZIHENG WANG, SHANSHAN LIU, AHMED LOURI, AND FABRIZIO LOMBARDI

1932-4510/23©2023IEEE

The study of the human brain and its operation has helped scientists to find better and more effective ways to cure many diseases.

The evaluation results of MLPs for two popular datasets show that the floating-point/fixed-point-based parallel (hybrid) design achieves the smallest latency (area) and the SC-based design offers the lowest energy dissipation.

ARTIFICIAL NEURAL NETWORK ACCELERATORS

The study of the human brain and its operation has helped scientists to find better and more effective ways to cure many diseases; it has also inspired the subject of neuromorphic computing to deal with the ever increasing need for artificial intelligence (AI) based machine designs to mimic the function of the human brain for solving complex problems. These designs usually fall into the category of Artificial Neural Networks (ANNs) [1]. ANNs are more efficient in comparison with conventional regression and statistical models for solving complex non-linear problems. A significant advantage of ANNs is the higher processing speed, as allowed for parallel implementation [2], [3]. Specifically, an ANN consists of several layers, and each layer has multiple circuits (hereafter referred to as neurons) that are connected to the same/next layer through synaptic weights [2]. ANNs have been utilized in a wide range of applications, such as pattern recognition, business and finance, tracking renewable energy to achieve sustainability, and language processing [4], [5], [6], [7].

Despite the large variety of applications and superior performance for extracting high-level features from raw data (and thus, reaching a higher accuracy), the hardware complexity of an ANN must be addressed by designing a suitable platform to accelerate the entire computational process; this has been investigated over may years [8], [9]. The most efficient and popular platforms can be categorized in three different groups: GPUs, FPGAs, and ASICs. There are also other platforms (like CPUs), but due to the lack of accuracy, they are out of interest as accelerators.

GPUs perform complex computations by parallelization. Although a GPU increases accuracy, the higher power dissipation makes them unsuitable for low power systems. To address this issue, FPGA-based designs have been proposed [10]. FPGAs offer flexibility and parallelization albeit less than GPUs. Moreover, they have better power characteristics, and unlike GPUs, FPGAs are reconfigurable and programmed using hardware/software codesign. Their lower speed and the demand for platforms with even lower power dissipation make ASIC designs a competitor. ASICs have less flexibility in comparison with FPGAs, so for example, the design cannot be changed after being implemented [11]. Nevertheless, their higher speed and lower power dissipation make them a good choice to implement ANN accelerators in power-limited applications [9].

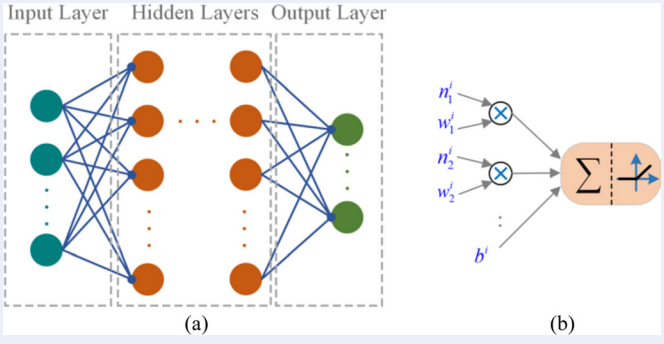

FIGURE 1 An MLP. (a) Network. (b) Neuron computation.

The main arithmetic operation in a fully connected ANN for each neuron is the sum of weighted products; this can be performed by multiply-and-accumulation (MAC) units [12], [13]. An example of these networks is the multilayer perceptron (MLP) that is a fully connected feedforward network; MLP has been extensively used in deep learning applications [14]. Since the neurons in each layer are connected to all neurons of the next layer through synapses, a significant increase in the number of neurons/layers requires an increase in MAC units; this makes the accelerator design challenging for systems requiring a higher performance in addition to power limitation, such as for Internet of Thing applications [2], [15].

To pursue a tradeoff between power/energy dissipation and clock rate, different arithmetic methods have been utilized [9]. Floating-point (FLP) arithmetic units based on the IEEE-754 standard [16] achieve a higher accuracy, but they also incur in a higher overhead due to the complex hardware [9], [17]. Fixed-point (FIP) numbers are simpler than FLP numbers when computing; so they incur in a lower overhead but lose accuracy due to a loss in precision [2], [12], [13]. Additionally, another method is to use stochastic computing (SC) that can be implemented with simple logics; it leads to a significant drop in overhead and allows the higher parallelization in computation in comparison with the other two formats [18], [19].

In this paper, we focus on implementing an ASIC-based MLP design using three arithmetic formats (i.e., FLP, FIP and SC) and introduce a tradeoff between complexity and accuracy to present the advantages and disadvantages of each method with simulation results for their implementations. The rest of this paper is organized as follows. The Multilayer Perceptron section reviews the MLP and discusses the challenges of its implementation using ASICs. The

Arithmetic Computing Schemes section presents different arithmetic computing methods. The MLP Implementation and Comparison section initially implements the proposed MLP designs for two popular datasets using different arithmetic computing methods; then, it compares the performance of the MLPs and the hardware overhead required for the different schemes. Finally, the last section concludes this article.

MULTILAYER PERCEPTRON

MULTILAYER PERCEPTRON

The multilayer perceptron (MLP) is a simple feedforward neural network (NN) with no recurrent connection. It consists of multiple layers including an input layer, at least a single hidden layer, and an output layer. Generally, the features dimension determines the number of neurons at the input layer, and the value of each neuron is given by the sum of weighted products. Therefore, the value of neuron n_k^{i+1} in layer $(i+1)$ is calculated by the sum of the product of each neuron n^i at layer i with its mapping weight w^i to neuron n_k^{i+1}. Finally, a bias value b^i is added to this sum and the so-called activation function (e.g., ReLU, which is considered in the remaining of this paper) is applied to it [2], [14]. Figure 1 shows the structure of an MLP and its neuron computation. As amenable to an MLP, different levels of parallelization in an ASIC-based MLP implementation are illustrated in subsequent subsections.

FULLY PARALLEL IMPLEMENTATION

In this scheme, the NN (with all layers and neurons) is implemented on the platform (i.e., ASIC); as the implemented design is specific to the MLP, each neuron receives all possible pairs of inputs (the weight and neuron value) from the previous layer at the same time and calculates the products and sums simultaneously. To perform all multiplications and sums, many multiplier and adder trees are required. Assume that there are m neurons in layer i; then m multipliers and a related adder tree to sum them up is required for each neuron at layer $(i+1)$ [9]. The fully parallel scheme permits a low computation latency; however, the implementation depends on the restrictions of the platform, or application. For example, when implementing larger networks, the hardware incurs in a large power dissipation and significant area overhead. In many cases, it is not feasible for applications with a limited power budget [20]. SC arithmetic units may be used in a fully parallel design for inference, because they use simpler hardware for multiplication and addition [21].

To perform all multiplications and sums, many multiplier and adder trees are required.

SERIAL IMPLEMENTATION

A serial implementation utilizes only a single neuron as process engine (PE); it can be realized in two ways using multiplier and adder trees.

1) Fully Parallel PE : This type of PE receives all possible input pairs of each neuron per clock cycle and then multiplies and accumulates them during the next cycles in a fully pipelined mode. In this case, the output of a neuron at the end of each cycle is obtained. This procedure continues for all neurons in the network sequentially. For example, assume a MLP as *A-B-C-D* (i.e., it has four layers and the number of neurons in each layer is A, B, C, D), with $C > A > B > D$. The neuron implementation must have at least $C + 1$ entries to support the calculation of all possible neurons in each layer; this occurs because C is the maximum number of neurons in a layer and then, in the output layer, a neuron with at least $C + 1$ entries is required to receive all neurons values from the second last hidden layer in addition to the bias value [9], [22]. This method of PE implementation incurs in a low delay; however, the efficiency significantly decreases with an increase of the size of the network. When there are hundreds of neurons in each layer; then, hundreds of multipliers and adders are also required for a PE design; this requires a massive area and incurs in a significant power dissipation. Moreover, it increases the static power dissipation when the number of input pairs in a layer is less than the number of entries in a PE, because the unnecessary entries must be connected to zero logic [9].

2) Semi Parallel PE : This PE design is simpler by eliminating some of the arithmetic units. Instead of choosing the number of entries for the PE based on the maximum number of neurons in a layer, a smaller number is selected as a tradeoff between area, power, and latency. For example, if a PE is designed with A entries and there are B neurons in layer i $(B > A)$, then for implementing each neuron in layer $i + 1$, the PE must receive inputs during $\lceil B / A \rceil$ consequent iterations (i.e., more cycles than fully pipelined serial designs) [9], [22]. Although

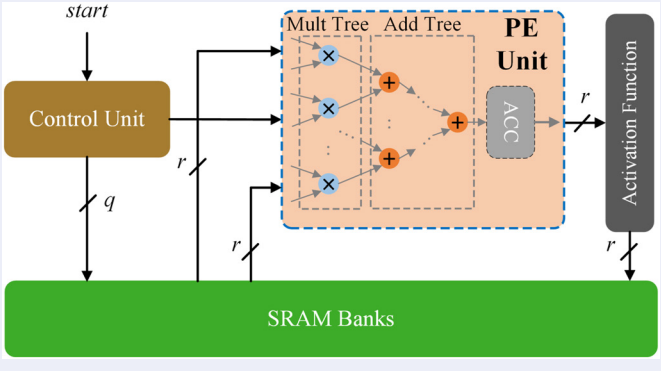

FIGURE 2 Serial implementation of an MLP.

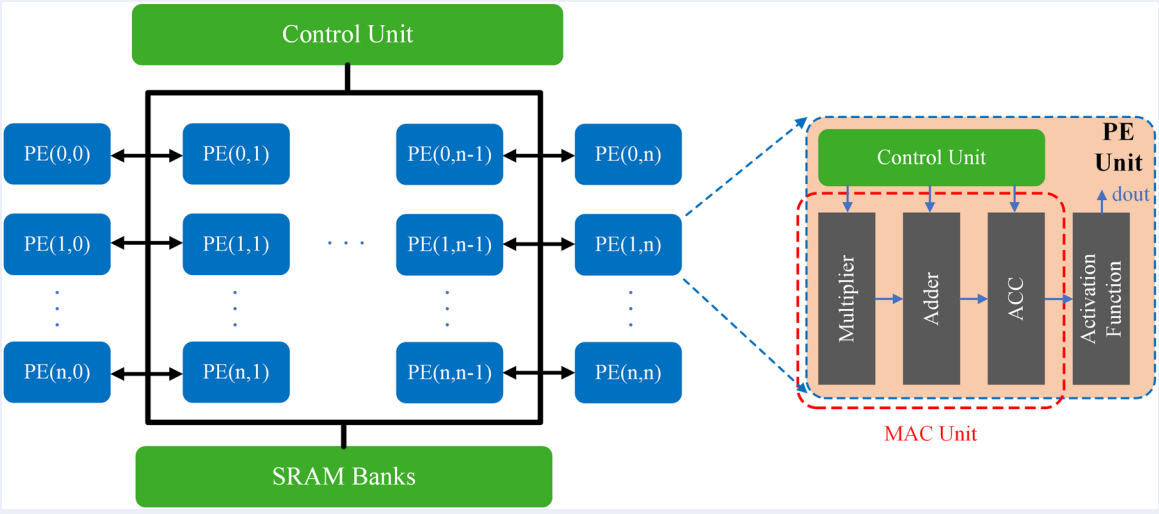

FIGURE 3 Hybrid implementation of an MLP.

it addresses the area/power issue of a fully parallel PE discussed previously, the semi parallel PE still significantly increases the latency for NNs of a large size.

Figure 2 shows a serial network; an SRAM is utilized for saving the weight values and the neuron values after a PE completes its operations for each neuron. The bus size r denotes the number of inputs the PE can receive simultaneously multiplied by the data width. The control unit generates two signals; one is for activating a PE, and the other is for determining the read/write, address, and the bank of the memory. The PE unit may have an accumulator (ACC) register based on the use of either a fully or semi parallel PE. For fully parallel PEs, the result of a neuron is ready at the last stage of the adder tree; thus, there is no need for the ACC. However, for the semi parallel PE, the process is completed in several iterations, and therefore, an ACC is required to accumulate the results of each iteration.

HYBRID IMPLEMENTATION

A hybrid MLP implementation employs a combination of parallelization and serialization in implementing the NN for a tradeoff between area, power, and latency. In this implementation, different strategies are illustrated next.

1) Full layer with fully parallel PEs: This scheme uses fully parallel PEs (consisting of multiplier and adder trees) as discussed in paragraph 3) of this section for implementing a layer with the maximum number of neurons in the NN [22]. This is a fast scheme, but it is not feasible for NNs of a large size; if the network has hundreds of neurons in each layer, then even a single PE occupies a large area and dissipates significant power. This further deteriorates the performance for the entire network, because the PE must be replicated hundreds of times for a full layer (as the layer with the largest number of neurons). Additionally, for layers with a smaller number of neurons, some neurons may be idle, so imposing unnecessary static power.

2) Full layer with MAC-based PEs: In this case, a fully parallel layer is also required based on the largest number of neurons among all layers in the NN. However, instead of using multiplier and adder trees in a PE as in the previous schemes, a simple multiply-and-accumulation (MAC) unit [23] is introduced in a PE to receive a pair of inputs per cycle. Each MAC-based PE can be considered as a neuron and receives all input pairs serially (one pair per cycle), rather than in parallel. This type of PE combines the serialization with parallelization to make an efficient accelerator for ANNs. Although it is more efficient for power and area, the latency deteriorates. By using a fully pipelined implementation of MAC units, the latency can be significantly improved, albeit larger than the scheme employing fully parallel PEs. A significant reduction in power and area is also achieved; however, for larger NNs and with a limited power budget, this implementation is still not efficient for a full layer with hundreds of MAC units; an improvement in design is discussed next to address this issue.

3) Array of MAC-based PEs: Different from the full layer with MAC-based PEs, this scheme does not implement a layer with the largest number of neurons; instead, it utilizes an arbitrary number of PEs based on the arithmetic computing units to achieve a good tradeoff. In this case, all neurons in a layer are not necessarily computed in a single iteration; as per the number of MAC units and number of neurons in a layer, the computation process can continue for several iterations [9]. Hence, this is a hybrid scheme. Figure 3 denotes the overview of a MAC-based hybrid design. A control unit is responsible for distributing the data from the SRAM banks to the different MAC units through the design. Each PE consists of a multiplier, an adder and an ACC to calculate the product of each pair of inputs and accumulate the results; when all input pairs are received and accumulated, the result is sent to the activation function. An additional control unit may be needed to deal with flushing [24]; this is described in more detail in The MLP Implementation and Comparison section.

ARITHMETIC COMPUTING SCHEMES

In this section different types of arithmetic computing units including FLP, FI, and SC are described.

FLOATING-POINT (FLP) ARITHMETIC

When utilizing FLP to implement a NN, the IEEE 754 single precision format (Figure 4) [16] is typically used for calculation. EQN. (1) gives its mathematical form, where *S*, *E*, *M* and *H* denote the sign, exponent, mantissa, and hidden bit respectively. The hidden bit is obtained by performing the OR on all exponent bits; a result of (1) refers to a normal FLP number, otherwise it is a sub-normal number.

$$\text{FLP Number} = (-1)^{S} \times 2^{(E-127)} \times (H.M). \quad (1)$$

The FLP arithmetic units often have a high complexity in hardware and thus, they incur in a significant area overhead in addition encountering large power dissipation in computation.

FIXED-POINT (FIP) ARITHMETIC

Compared to FLP numbers with a constant bit width, FIP numbers requires a longer bit width to provide a large range of values. FIP numbers have a simpler representation; a binary point separates the integer and fractional parts within a data representation. Therefore, its data resolution is not as good as for FLP numbers. However, unlike FLP arithmetic computing, only simple integer arithmetic units are utilized for implementing FIP computation. Moreover, the pre and post normalization stages in FLP arithmetic (specially for addition due to its complicated computation) require combinational encoders and shifters that may deteriorate latency, area, and power; thus, it makes FLP more complex than FIP arithmetic. Overall, the use of FIP arithmetic decreases area and power dissipation, while improving latency, because they do not have the exponent part (so related complicated computations are not required). Therefore, the tradeoff between hardware overhead and accuracy must be considered when using these arithmetic units.

Different configurations (including the fully parallel design, serial design and hybrid design) are utilized for implementing a FL P-based MLP.

STOCHASTIC COMPUTING (SC)

In conventional FLP/FIP computing, processing a large amount of data in a fully parallel network implementation makes the design often complicated and infeasible due to the complex computational units. Therefore, emerging computing paradigms have been investigated for reducing the hardware complexity [18]; for example, a promising solution is to use stochastic computing (SC). SC was introduced in 1960s for the first time, but it has attracted more attention in recent years for nano-scaled designs [26].

An important characteristic of SC is that real numbers are converted into bit-stream, and thus, very simple designs can be used for performing arithmetic operations [26]. Specifically, an SC bit-stream is generated using a stochastic number generator (SNG); the occurrence probability of 1 in the sequence is used to represent the real number. Therefore, its possible range of values is [0, 1] (for unipolar computation) and [-1, 1] (for bipolar computation). A simple AND (XNOR) gate can perform the multiplication on unipolar (bipolar) SC bit-streams; moreover, since SC operates with probabilities, the computation is capable of tolerating transient or soft errors caused by process variations or cosmic interference [26].

Even though SC has unique advantages in terms of low hardware complexity and error tolerance against bit flips in the sequence, its use may cause an accuracy loss. This issue is generated due to two causes: i) the low data resolution determined by the probabilities; ii) the limited range of values. To address these issues, a longer sequence can be used to improve the data resolution; moreover, designs based on extended stochastic logic (ESL) have been investigated to extend the computation range by utilizing the quotient of two stochastic sequences [18]. However, these solutions may incur in hardware penalties, so that a careful design is required for a tradeoff between computation accuracy and

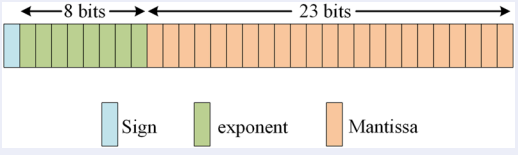

FIGURE 4 Single precision format with IEEE 754 standard.

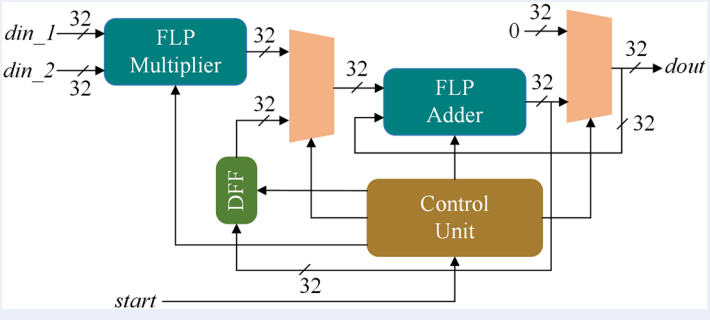

FIGURE 5 The pipelined FLP MAC Unit.

Due to less complex arithmetic designs in SC, the fully parallel implementation of the NN is usually selected.

hardware. For more details, the reader should refer to [18], [21], [26].

MLP IMPLEMENTATION AND COMPARISON

In this section, MLPs for two popular datasets are implemented by using different arithmetic units; the network structure is given by 784-100-200-10 for MNIST and 1024-100-200-10 for the SVHN. The classification accuracy and hardware overhead for the different schemes are evaluated and compared.

IMPLEMENTATION USING FLP ARITHMETIC

Different configurations (including the fully parallel design, serial design and hybrid design) are utilized for implementing a FLP-based MLP. The fully parallel and serial (using fully parallel PEs as an example) implementations are performed as discussed in 2) and 3) of the second section. In the hybrid scheme, an array of MAC-based PEs is selected because it incurs in a low hardware overhead as discussed previously.

For designing the MAC unit, an FLP multiplier and an FLP adder (more details of their designs and algorithms can be found in [25]) are needed (Figure 5). A pipelined strategy has been proposed in [9] to improve the clock rate; so, the adder and multiplier complete the computation in four and five cycles, respectively. As these units require different number of cycles, a control unit is used to manage the data and flush the MAC unit at the end. The pipelined MAC unit is shown in Figure 5. To perform computation as a PE, the multiplier initially receives an input pair per cycle; once the multiplication result is generated after four cycles, the multiplexer transfers them to the input of the adder. Since the adder requires five cycles to generate the result, the other input of the adder must be 0, and it is handled by the second multiplexer. At the end of cycle 9, both the adder and multiplier outputs are valid, then the second multiplexer switches from 0 to the adder outcome to accumulate the results. When all input pairs are received, the pipeline stages are flushed. Therefore, DFF saves the output of the adder at cycle i and adds it with its output at cycle i+1; [9] presents this process in more details.

Therefore, the hybrid implementation of an MLP using FLP MAC units is like in Figure 3. For the datasets considered in this paper, we use 64 PEs (MAC units) and a control unit for distributing data on the PEs; for example, for a layer with 100 neurons, the MAC array completes the calculation of 64 neurons in first iteration and then the remaining 36 neurons in the second iteration.

IMPLEMENTATION USING FIP ARITHMETIC

As the FIP arithmetic units are simpler than FLP (they are like integer arithmetic units) and due to the lower overhead, a full layer using MAC-based PEs is selected for the hybrid implementation. The lower complexity of the units allows us to use fewer pipelining stages; so, the only pipelining stages are the ones between the multiplier and adder. Then the design consists of a simple integer multiplier and an adder on top of an accumulator such that each design works in one cycle. A 16-bit format [2] (including one bit for the sign, eight bits for the integer and seven bits for the fraction) is used for calculations as widely utilized. By increasing the number of bits, the accuracy can be improved but at a higher hardware overhead. Hence, a tradeoff between latency, area, and power dissipation must be considered. In this paper, the goal is to decrease the latency, and accordingly, shorter sequences are selected to decrease the overhead which allows to increase the parallelization.

IMPLEMENTATION USING SC ARITHMETIC

Due to less complex arithmetic designs in SC, the fully parallel implementation of the NN is usually selected (unlike FLP and FIP in which more complex hardware is needed). Sequences with a length of 256 bits are utilized to assess the trade-off between computational accuracy and hardware overhead. Figure 6 shows the SC-based implementation for a neuron (i.e., a PE) of the

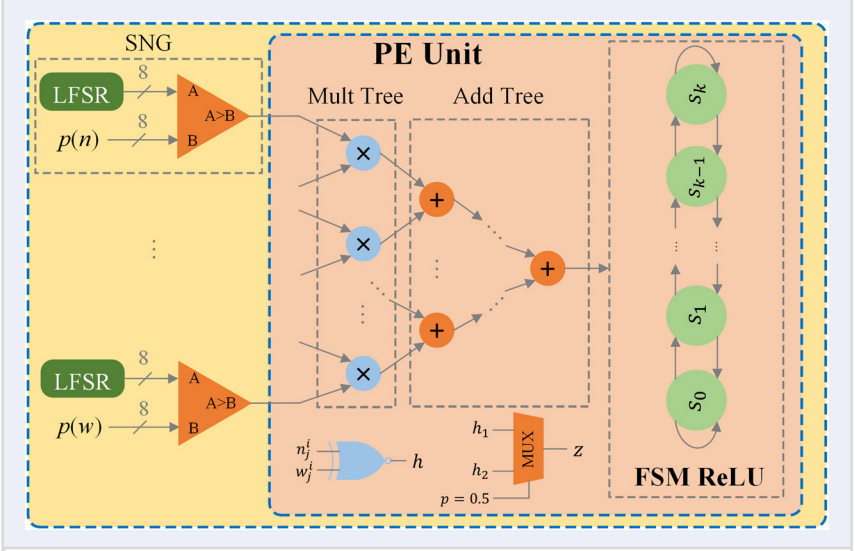

FIGURE 6 SC-based PE implementation for a neuron (the detailed circuit of a multiplier/adder is shown below the tree block).

MLPs. As shown in this figure, an SNG consists of a linear feedback shift register (LFSR) and a comparator to convert the probability of a real number (i.e., the input data or weight) to its stochastic sequence. For neurons in the first hidden layer, the SNG is required for each input of the multiplier; for the other layers, it is only required for converting the weight. The SC multiplier and adder are also given in Figure 6; the activation function ReLU is implemented using an FSM [21]. As this scheme is fully parallel, then the latency is decreased significantly; however, the power dissipation increases as well. Also, there is no need for memory to save the neuron values for the next uses. Consider the energy as a further metric, this design may be better suited than the FLP/FIP-based hybrid design, because the lower latency compensates the increase in power. A hybrid design of SC design is also possible, but it needs to store the long stochastic sequences in a large SRAM for next use; hence, it may be counterproductive if its purpose is to reduce area and power dissipation of a parallel design.

EVALUATION AND COMPARISON

Table 1 shows the synthesis results of different schemes for the two datasets considered in this paper. The same clock frequency (800MHz) is utilized to ensure a fair comparison.

> Overall, for the two datasets considered in this paper, the FLP/FIP-based parallel design achieves the smallest latency, the hybrid design achieves the smallest area, and the SC design requires the smallest energy.

Consider the conventional FLP and FIP implementations first. As shown in Table 1, the parallel scheme requires the largest area and energy, but the smallest latency to complete the entire MLP inference process; this is expected due to its fully parallelization. Compared to the parallel design, the serial scheme achieves a reduction in area and energy, but these results are still considerable. The hybrid design offers the smallest area and energy, but a significant latency is incurred; this occurs because a single MAC unit is required for calculating each neuron, so requiring many cycles. As the designs are fully pipelined, then a neuron per cycle (all neurons per cycle) for each layer is processed for serial (fully parallel) implementations. This causes a significant decrease in the number of cycles by imposing a significant increase in overhead. Moreover, an SRAM for storing the temporary computation results for next uses is also required in the serial and hybrid schemes; the overhead for the SRAM is the same in these implementations for the same MLP structure.

The SC scheme is shown to be more efficient in area and energy compared to the FLP/FIP-based parallel and serial schemes due to the low complexity of the arithmetic units. However, its area is larger than the FLP/FIP-based hybrid design; this occurs because a fully parallel implementation is employed in the SC scheme. The latency for an SC implementation is larger than the FLP/FIP-based parallel design, but it is smaller than the other schemes. Also, no SRAM is required in the SC scheme.

Overall, for the two datasets considered in this paper, the FLP/FIP-based parallel design achieves the smallest latency, the hybrid design achieves the smallest area, and the SC design requires the smallest energy. The classification accuracy of the MLP in different schemes is also evaluated and compared in Table 1.

TABLE 1 Synthesis results for MLPs using different implementation methods.

DATASET	IMPLEMENTATION		ARITHMETIC UNITS			SRAM			# OF CYCLES	ACCURACY
			Area (mm^2)	Latency[a] (μs)	Energy (μJ)	Size (Bytes)	Power (mW)	Area (mm^2)		
MNIST [27]	FLP	Parallel	1465.39	**0.14**	104.48	-	-	-	115	**98.36%**
		Serial	11.44	0.53	3.01	310*32	54.61	0.15	425	**98.36%**
		Hybrid	0.93	2.94	1.36	310*32	54.61	0.15	2352	**98.36%**
	FIP	Parallel	334.55	**0.03**	7.91	-	-	-	30	93.31%
		Serial	2.61	0.42	0.7	310*16	31.63	0.08	340	93.31%
		Hybrid	**0.67**	1.37	0.57	310*16	31.63	0.08	1096	93.31%
		SC	1.72	0.32	**0.39**	-	-	-	260	97.49%
SVHN [28]	FLP	Parallel	1815.69	**0.14**	129.48	-	-	-	**115**	**89.91%**
		Serial	14.94	0.53	3.93	310*32	54.61	0.15	425	**89.91%**
		Hybrid	0.93	3.54	1.64	310*32	54.61	0.15	2832	**89.91%**
	FIP	Parallel	414.52	**0.03**	9.80	-	-	-	30	82.81%
		Serial	3.41	0.42	0.91	310*16	31.63	0.08	340	82.81%
		Hybrid	**0.67**	1.65	0.69	310*16	31.63	0.08	1324	82.81%
		SC	2.11	0.32	**0.48**	-	-	-	260	88.61%

[a]The latency result is for the entire computation of an MLP inference, i.e., including the number of cycles.

The FLP scheme offers the highest accuracy, followed by the SC scheme and then the FIP scheme. Therefore, different implementation schemes can be selected as per the requirement of different applications.

CONCLUSION

This paper has presented the ASIC-based design of various MLP accelerators. The hardware for performing network computation using conventional floating-point and fixed-point based computing and stochastic computing, as well as different schemes (parallel, serial and hybrid arrangements), has been analyzed and implemented. The evaluation shows that each type of implementation has a unique advantage in hardware metric. All findings and designs presented in this paper provide a comprehensive assessment for designing nanoscale accelerators when the ANN application has different requirements of learning performance and hardware overhead.

ABOUT THE AUTHORS

Farzad Niknia (niknia.f@northeastern.edu) is with the Department of Electrical and Computer Engineering, Northeastern University, Boston, MA, 02215, USA. He is a Student Member of IEEE.

Ziheng Wang (wang.zihe@northeastern.edu) is with the Department of Electrical and Computer Engineering, Northeastern University, Boston, MA, 02215, USA. He is a Student Member of IEEE.

Shanshan Liu (ssliu@coe.neu.edu) is with the Klipsch School of Electrical and Computer Engineering, New Mexico State University, Las Cruces, NM, 88001, USA. He is a Member of IEEE.

Ahmed Louri (louri@gwu.edu) is with the Department of Electrical and Computer Engineering, George Washington University, DC, 20052, USA. He is a Fellow of IEEE.

Fabrizio Lombardi (lombardi@ece.neu.edu) is with the Department of Electrical and Computer Engineering, Northeastern University, Boston, MA, 02215, USA. He is a Life Fellow of IEEE.

REFERENCES

[1] Q. Zhang, H. Yu, M. Barbiero, B. Wang, and M. Gu, "Artificial neural networks enabled by nanophotonics," *Light: Sci. Appl.,* vol. 8, no. 42, pp. 1–14, May 2019.

[2] D. Kim, J. Kung, and S. Mukhopadhyay, "A power-aware digital multilayer perceptron accelerator with on-chip training based on approximate computing," *IEEE Trans. Emerg. Topics Comput.*, vol. 5, no. 2, pp. 164–178, Apr.–Jun. 2017.

[3] O. I. Abiodun, A. Jantan, A. E. Omolara, K. V. Dada, N. A. Mohamed, and H. Arshad, "State-of-the-art in artificial neural network applications: A survey," *Heliyon*, vol. 4, no. 11, pp. 1–41, Nov. 2018.

[4] O. I. Abiodun et al., "Comprehensive review of artificial neural network applications to pattern recognition," *IEEE Access*, vol. 7, pp. 158820–158846, 2019.

[5] M. Tkáč and R. Verner, "Artificial neural networks in business: Two decades of research," *Appl. Soft Comput.*, vol. 38, pp. 788–804, Jan. 2016.

[6] A. P. Marugán, F. P. G. Márquez, J. M. P. Perez, and D. R. Hernández, "A survey of artificial neural network in wind energy systems," *Appl. Energy*, vol. 228, pp. 1822–1836, Oct. 2018.

[7] L. M. Elobaid, A. K. Abdelsalam, and E. E. Zakzouk, "Artificial neural network-based photovoltaic maximum power point tracking techniques: A survey," *IET Renewable Power Gener.*, vol. 9, no. 8, pp. 1043–1063, Nov. 2015.

[8] V. Sze, Y. H. Chen, T. J. Yang, and J. S. Emer, "Efficient processing of deep neural networks: A tutorial and survey," *Proc. IEEE*, vol. 105, no. 12, pp. 2295–2329, Dec. 2017.

[9] F. Niknia, Z. Wang, S. Liu, and F. Lombardi, "Nanoscale design of multi-layer perceptrons using floating-point arithmetic units," in *Proc. IEEE 22nd Int. Conf. Nanotechnol.*, 2022, pp. 1–6.

[10] M. A. Talib, S. Majzoub, Q. Nasir, and D. Jamal, "A systematic literature review on hardware implementation of artificial intelligence algorithms," *J. Supercomput.*, vol. 77, no. 2, pp. 1897–1938, May 2020.

[11] D. Moolchandani, A. Kumar, and S. R. Sarangi, "Accelerating CNN inference on ASICs: A survey," *J. Syst. Architecture*, vol. 113, Feb. 2021, Art. no. 101887.

[12] A. Shiri, A. N. Mazumder, B. Prakash, H. Homayoun, N. R. Waytowich, and T. Mohsenin, "A hardware accelerator for language-guided reinforcement learning," *IEEE Des. Test*, vol. 39, no. 3, pp. 37–44, Jun. 2022.

[13] A. Shiri et al., "Energy-efficient hardware for language guided reinforcement learning," in *Proc. Great Lakes Symp. VLSI*, 2020, pp. 131–136.

[14] S. Haykin, *Neural Networks and Learning Machines*, 3rd ed. Noida, India: Pearson Education India, 2009.

[15] L. Ye et al., "The challenges and emerging technologies for low-power artificial intelligence IoT systems," *IEEE Trans. Circuits Syst. I: Regular Papers*, vol. 68, no. 12, pp. 4821–4834, Dec. 2021.

[16] *IEEE Standard for Floating-Point Arithmetic*, IEEE Standard 754-2019, Jul. 2019, pp. 1–84.

[17] W. Jiang et al., "Achieving super-linear speedup across multi-fpga for real-time dnn inference," *ACM Trans. Embedded Comput. Syst.*, vol. 18, no. 67, pp. 1–23, Oct. 2019.

[18] S. Liu et al., "Stochastic dividers for low latency neural networks," *IEEE Trans. Circuits Syst. I: Regular Papers*, vol. 68, no. 10, pp. 4102–4115, Oct. 2021.

[19] Y. Liu, S. Liu, Y. Wang, F. Lombardi, and J. Han, "A survey of stochastic computing neural networks for machine learning applications," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 32, no. 7, pp. 2809–2824, Jul. 2021.

[20] L. D. Medus, T. Iakymchuk, J. V. Frances-Villora, M. Bataller-Mompeán, and A. Rosado-Muñoz, "A novel systolic parallel hardware architecture for the FPGA acceleration of feedforward neural networks," *IEEE Access*, vol. 7, pp. 76084–76103, 2019.

[21] Y. Liu, S. Liu, Y. Wang, F. Lombardi, and J. Han, "A stochastic computational multi-layer perceptron with backward propagation," *IEEE Trans. Comput.*, vol. 67, no. 9, pp. 1273–1286, Sep. 2018.

[22] I. Westby, X. Yang, T. Liu, and H. Xu, "FPGA acceleration on a multi-layer perceptron neural network for digit recognition," *J. Supercomput.*, vol. 77, no. 12, pp. 14356–14373, May 2021.

[23] N. Nedjah, R. M. da Silva, L. M. Mourelle, and M. V. C. da Silva, "Dynamic MAC-based architecture of artificial neural networks suitable for hardware implementation on FPGAs," *Neurocomputing*, vol. 72, no. 10–12, pp. 2171–2179, Jun. 2009.

[24] Z. Wang, F. Niknia, S. Liu, P. Reviriego, P. Montuschi, and F. Lombardi, "Tolerance of siamese networks (SNs) to memory errors: Analysis and design," *IEEE Trans. Comput.*, early access, Jun. 28, 2022, doi: 10.1109/TC.2022.3186628.

[25] J. M. Muller et al., *Handbook of Floating-Point Arithmetic*. Basel, Switzerland: Birkhäuser, 2018.

[26] A. Alaghi and J. P. Hayes, "Survey of stochastic computing," *ACM Trans. Embedded Comput. Syst.*, vol. 12, no. 2s, pp. 1–19, May 2013.

[27] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proc. IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov. 1998.

[28] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Y. Ng, "Reading digits in natural images with unsupervised feature learning," in *Proc. NIPS Workshop Deep Learn. Unsupervised Feature Learn.*, 2011, pp. 1–9.

N