Doctoral Dissertation

Efficient Stochastic Computing Architectures for Deep Neural Networks

Van Tinh Nguyen

January 24, 2022

Graduate School of Science and Technology
Nara Institute of Science and Technology

A Doctoral Dissertation
submitted to Graduate School of Science and Technology,
Nara Institute of Science and Technology
in partial fulfillment of the requirements for the degree of
Doctor of ENGINEERING

Van Tinh Nguyen

Thesis Committee:

Professor Yasuhiko Nakashima	(Supervisor)
Professor Yuichi Hayashi	(Co-supervisor)
Associate Professor Renyuan Zhang	(Co-supervisor)

Efficient Stochastic Computing Architectures for Deep Neural Networks[*]

Van Tinh Nguyen

Abstract

This thesis comprises four parts, where the first one presents a novel architecture for radial basis function (RBF) computation employing stochastic computing. The RBF is optimized using proposed simple stochastic logic circuits. We validated this approach by comparison with both Bernstein polynomial and two-dimensional finite-state machine (2D-FSM)-based implementation. Optimally, the mean absolute error is reduced by 40% and 80% compared to two other well-known approaches, Bernstein polynomial and 2D-FSM-based implementation, respectively. In terms of hardware cost, our proposed solution required as much as the Bernstein method did. Moreover, the proposed approach outperforms the 2D-FSM-based implementation, roughly 54% less hardware cost. Regarding the critical path delay, the proposed system is less than 12% than others on average. Besides, the proposed architecture also required 70% less power than 2D-FSM-based implementation. The second part of the thesis proposes a novel technique implementation of hyperbolic $\tanh(ax)$ and $\text{sigmoid}(2ax)$ functions for high precision and compact computational hardware based on stochastic logic. This work demonstrates the stochastic computation of $\tanh(ax)$ and $\text{sigmoid}(2ax)$ functions-based Bernstein polynomial using a bipolar format. The format conversion from bipolar to unipolar format is involved in our implementation. One achievement is that our proposed method is more accurate than the state-of-the-art, including the finite-state machine (FSM)-based method, JK-FF. On average, 90% of improvement of this work in terms of mean square error (MAE) has been

[*]Doctoral Dissertation, Graduate School of Science and Technology, Nara Institute of Science and Technology, January 24, 2022.

achieved while the hardware cost and power consumption are comparable to the previous approaches. The third and fourth parts of the thesis propose an in-memory binary spiking neural network (BSNN) using stochastic computing (SC) for information encoding. The residual BSNN learning using a surrogate gradient that shortens the time steps in the BSNN while maintaining sufficient accuracy is proposed. At the circuit level, presynaptic spikes are fed to memory units through differential bit lines (BLs), while binarized weights are stored in a subarray of nonvolatile spin-transfer-torque magneto-resistive RAM (STT-MRAM). The hardware/software co-simulation results indicate that the proposed design can deliver a performance of 176.6 TOPS/W for an in-memory computing (IMC) subarray size of 1×288. The classification accuracy reaches 97.92% (83.85%) on the MNIST (CIFAR-10) dataset. The impacts of the device nonidealities and process variations are also thoroughly covered in the analysis.

Keywords:

Radial Basis Function (RBF), Hyperbolic tanh(ax) and sigmoid($2ax$) Functions, Stochastic Computing (SC), Binary Spiking Neural Network, Emerging Memory Technology, In-Memory Computing, Neuromorphic Computing, Process Variation, STT-MRAM

Contents

List of Figures

List of Tables

1 Introduction

This part represents the overview of SC in computational units for Deep Neural Networks (DNNs), SC-based activations, which this thesis has focused on improving their accuracy and performance. Since the SC approaches can not solve the problem of limited memory bandwidth in DNNs, we further study the stochastic binary spiking neural network, which partially shifts the processing load from the central processing unit to distributed processing elements in memory. It can greatly reduce memory access while increasing performance and energy efficiency.

1.1 Overview

DNNs are used in many artificial intelligence (AI) applications such as computer vision [1], natural language processing [2], and audio processing [3]. At the core of the algorithms and architectures of DNNs are some basic computations present in large numbers [4, 5]. In order to speed up the DNNs, many core architectures are employed [6]. However, the potential of parallelism/speedup is limited by the hardware efficiency [7]. Both algorithmic and architectural efforts have been made to reduce the computing cost for AI edge devices. The former relies on minimizing the demands on computations in the algorithm or architecture [8]. A typical strategy for optimization is by pruning complicated DNNs into the sparse schemes [9, 10, 11], which is also adapted in DNNs topology. On the architectural side, the hardware implementation with low cost is always favored [12]. As a promising candidate, the approximate computing technology has been widely applied to perform the necessary computations of DNNs efficiently [13]. It is noticed that reasonably inaccurate computational cores powered by resistive, quantum, analog, and SC can also offer satisfactory hardware cost in the DNN applications [14, 15, 16, 17, 18]. SC perfectly fits the DNNs operations in the sense of hardware efficiency [19, 20, 21, 22, 23, 24, 25, 26]. The SC, where data is represented and processed by a serial bit-stream with a probabilistic feature, conducts the multiplication or summation by a single logic gate or multiplexer (MUX) [27, 28, 29, 30, 31, 32, 33, 34, 35]. Some nonlinear functions can also be executed with supplemented mechanisms [36, 37, 38, 39] encountered in SC-based DNNs. Nonlinear functions based SC implementation for DNNs have performed much higher efficiency than other approximate computing technologies [38, 39, 40, 41]. In the first part of this thesis, my work aims to propose an implementation of nonlinear functions using stochastic logic to achieve high accuracy while requiring reasonable hardware cost and fit the stochastic end-to-end system. First, We propose novel approaches for nonlinear activations using SC including RBF, hyperbolic tangent and sigmoid functions. The proposed architectures are also synthesized using Synopsys ASIC flow, for 180 nm standard cell library. The synthesis results proved that our work enhanced the hardware performance significantly in terms of area, and energy.

On the other hand, DNNs face the challenge of high energy consumption due to the requirement of a large number of tensor operations, which incurs not only a high computational workload but also large memory accesses [42, 43, 44, 45, 46]. The conventional computer architecture with limited memory bandwidth and a sequential computing framework is not ideal for such operations, especially for DNNs in on-edge AI devices such as the Internet of Things (IoT) or mobile systems with strict resource and power budgets constraints. IMC has been recently introduced as a revolutionary approach to solving the memory bottleneck challenge [47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60]. This approach partially shifts the processing load from the central processing unit to distributed processing elements in memory, which greatly reduces memory access while increasing performance and energy efficiency. IMC finds it difficult to support complex processing operations such as multiply-accumulate (MAC); therefore, the binary neural network (BNN) has recently emerged [61, 62, 63, 64, 65, 66], with the aim of simplifying network operations. Interestingly, IMC approaches and BNNs apparently exhibit much synergy. Indeed, a BNN typically performs calculations based on bitwise XNOR operations (for multiplication), and bit counting (for accumulation) can be fully implemented in memory, as proposed in some prior works [67]. For instance, in [68, 69, 70, 71, 72], IMC macros employ 6T SRAM, 8T SRAM bitcell with dual wordline (WL) inputs to capture a MAC calculation. However, the area and leakage power disadvantages of SRAM degrade the expected effectiveness of in-memory calculation. Other recent IMC designs employ RRAM (Resistive RAM) instead of SRAM to yield a low standby power and improve array density [73, 74]. However, in these works, the current summing approach is used for the accumulation that can draw significant energy for a large network. As a further development, STT-MRAM was also adopted for IMC. STT-MRAM exhibits outstanding advantages over other memories (e.g., RRAM, PCM-RAM) in terms of endurance [75, 76, 77, 78, 79], thus, it is well-suited for on-chip memory and has been actively explored for the BNN-IMC combination [80, 81, 82, 83, 84, 85]. In-memory logic operations are implemented in [80, 84], but only bulk bit-wise functions are supported. Meanwhile, the STT-MRAM based BNN accelerator architecture in [84] is only reasonable for a small array size (8×4 array). The reason is that the magnetic tunnel junctions (MTJs) have a low tunneling magneto-resistance ratio (TMR) that makes the current sensing approach unsuitable for the larger memory array. Furthermore, in [85], the authors introduce a scalable and fully parallel in-memory BNN structure, which supports MAC operation in a single cycle by voltage sensing technique and achieves better scalability and energy-efficient. Nonetheless, the accuracy of BNNs strongly depends on the process variations, which could be quite severe in finer technologies.

Table 1. The overall evaluation the performance of Stochastic SNN, SC-DNN, and DNN

Compare	Accuracy	Speed	Area	Power	Overall
DNNs	$\star$	$\circ$	$\times$	$\triangle$	$\circ$
SC-DNN	$\triangle$	$\odot$	$\star$	$\odot$	$\odot$
Stoch. SNN	$\triangle$	$\odot$	$\star$	$\star$	$\odot$

Spiking neural networks (SNNs), known as the third generation of DNNs, not only better mimic biological neural behaviors but also exhibit great fault tolerance and can potentially overcome the persistent drawback of binarized networks [86]. Stochastic SNN (Stoch. SNN) is the kind of SC-DNN in which the spikes are encoded by SNG [87, 88, 89]. Stoch. In comparison with conventional SC-DNNs, Stoch.SNN employs event-driven architecture, which can minimize its power consumption. The overall evaluation between Stoch.SNN, SC-DNN, and DNN have been illustrated in Table.1. Recently, IBM Neurosynapse (TrueNorth chip) system was produced as a well-known neuromorphic chip which is the most energy efficiency in comparison with GPU and FPGA [90]. The memristor crossbar (RRAM) based neuromorphic architectures have been studied that much improves the computational and energy efficiency compared to the TrueNorth design [91, 92]. Due to the advantages of lower access energies compared to RRAM and DRAM [46], MTJ was also adopted for stochastic SNNs [87, 88, 89, 93, 94, 95]. The overall performance evaluation of the neuromorphic architectures with various technologies is shown in Table.2.

In [87, 88, 89], the authors leveraged the binary switching of an MTJ to map the sigmoid function in an artificial neural network (ANN) to an SNN. However, this mapping may cause significant accuracy degradation. Additionally, the accuracy suffers from unstable MTJ switching and bias current variation. Inhibitory and excitatory spike-timing-dependent plasticity was processed for on-chip learning with MTJ-synaptic [93] and MTJ-neuron [94] implementations. In [93], a stochastic synapse was introduced, in which synaptic propagation was modulated stochastically by a full-precision weight. Then, each neuron accumulated incoming synaptic events sequentially using a spike counter, which significantly affected the network latency. In [94], the leaky-integrate-fire spiking model was used to emulate biological neuron dynamics, but this work focused only on neuron circuitry and did not cover the impact of variation. On the other hand, in [95], the author presented a compact probabilistic Poisson method based on a back-hopping oscillation in an MTJ, where the number of spikes was exponentially proportional to the synaptic current in the utilized sampling time (within a time step). However, the classification accuracy of this approach is highly sensitive to the sampling period.

Table 2. The overall performance evaluation of neuromorphic architectures

Compare	Accuracy	Speed	Area	Power	Overall
GPU	$\star$	$\circ$	$\times$	$\times$	$\circ$
Memristor	$\triangle$	$\odot$	$\star$	$\triangle$	$\circ$
True North	$\odot$	$\odot$	$\triangle$	$\triangle$	$\circ$
MRAM	$\triangle$	$\odot$	$\star$	$\star$	$\odot$

1.2 Research Contribution

Our research contributions can be summrized as follows:

• We propose a novel design of the RBF using SC. The experimental results show that our work achieves a compact and high-level accuracy in the hardware design compared with previous works. We have also presented the unipolar and bipolar stochastic format analyses applying for our proposal.

• We propose an implementation of hyperbolic tangent and sigmoid functions using SC to achieve high accuracy while requiring reasonable hardware costs and fitting the stochastic end-to-end system. Bernstein polynomial has been used in this work as a kernel to approximate those two functions in our proposed implementation. Format conversion from bipolar to unipolar format has been used in our work.

• We propose a BSNN with residual connections and train the network with a surrogate gradient, which enables higher classification accuracy with fewer time steps.

• The proposed dynamic threshold mechanism allows neural synapses to be mapped to XNOR cells based on the STT-MRAM subarray and reduces the complexity of the neuron circuit by incorporating BN.

• The proposed approaches are built for circuit-level simulations. The network's accuracy and other performance metrics are then evaluated based on parameters realistically extracted from circuit simulations, including the nonlinearity and process variations.

The contributions have been published in [96, 97, 98, 99].

1.3 Dissertation Layout

The rest of this dissertation is organized as follows. Section 2 presents related works in high-performance SC-based activations and BSNN. Section 3 describes proposed SC-based architectures for some activation functions. Section 4 expresses the models of the STT-BSNN subarray, and the energy efficiency of our design is compared with that of previous studies. Then, Section 5 concludes this work.

2 Related Literature

2.1 Overview of Stochastic Computing

The SC constituent elements are circuits that convert binary numbers to stochastic numbers and vice versa, as shown in Fig. 1a. Fig. 1a shows a circuit, which describes the process of binary-to-stochastic numbers conversion, which is called a stochastic number generator (SNG). The SNG often consists of a comparator and a random number generator RNG (in this work, an LFSR is employed for RNG) [14, 27, 36]. An m-bit random binary number is generated in each clock cycle by the RNG and compared to the m-bit binary number (that is to be converted). If the random number is less than the binary number, then the output of the comparator will be 1, otherwise a 0. A stochastic bit-stream is converted back to a binary number using a counter that counts the number of 1's in this stream.

In SC, we use unipolar and bipolar formats for positive and negative fractional numbers, respectively. For the unipolar format, a value of stochastic bit stream X is indicated by x, where $x \subseteq [0, 1]$, and $P_X = x$ defines the probability of ones in X.

Suppose that Y is a stochastic bitstream in a bipolar format. Then, we determined the value of this bitstream, which is indicated by y, where $y \subseteq [-1, 1]$. Additionally, the probability of the ones in the bit-stream Y is indicated by P_Y, where $P_Y \subseteq [0, 1]$. Finally, the relation between y and P_Y is expressed as $y = 2P_Y - 1$.

The main advantage of SC is that the fundamental processing elements can be implemented based on simple logic gates. For example, the NOT gate is used to implement 1-x in unipolar format and -x in bipolar format. Multiplication which requires a high hardware cost in the binary format representation is now implemented just by an AND gate in unipolar format or XNOR gate in the bipolar format as shows in Fig. 1b and Fig. 1c, respectively. A multiplexer (MUX) implements a scaled addition for both unipolar and bipolar formats.

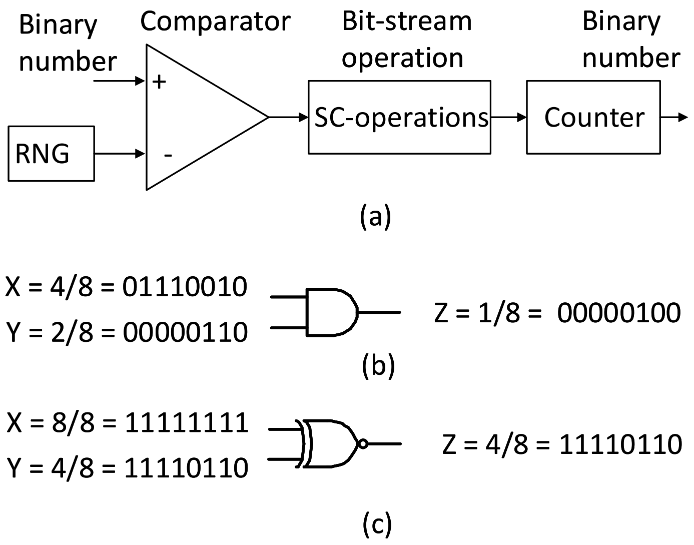


Figure 1. Fundamental stochastic computational elements.

2.2 Stochastic Computing based Activations

Brown and Card in [34] have first proposed SC implementations of hyperbolic tangent and exponential functions using FSM. An improvement of this approach used 2-D FSM was presented in [37]. One problem with this means of implementation is accuracy degradation when it is small small. An example of 2D-FSM topology in Fig. 2 accepts two input bit streams X and K and provides the transition condition numbers corresponding to the input X and the input K. For example, for 2D-FSM topology with eight states (s_i), the output of 2D-FSM is encoded by using three binary bits. The complete circuit for implementation of the stochastic function is illustrated in Fig. 3. The corresponding probabilities of K and w are P_k and P_{w_i} (i- the number of states) which are calculated by using the optimized algorithm as described in [37].

An alternative approach was proposed in [41], in which hyperbolic tangent and sigmoid function were approximated by series expansion and JK-Flip Flops. However, this approach leads to even lower accuracy than the FSM approach while requiring a higher hardware cost [41]. In [36, 39], the authors proposed an approximated approach for those two functions based on a piecewise linear

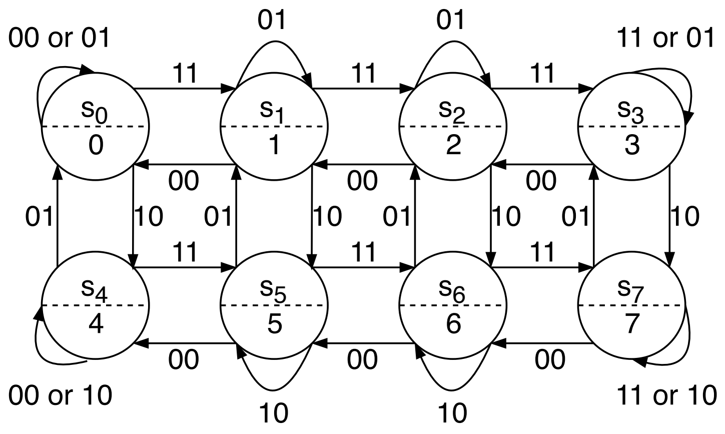


Figure 2. The state transition diagram of the 2-D FSM with 8 states.

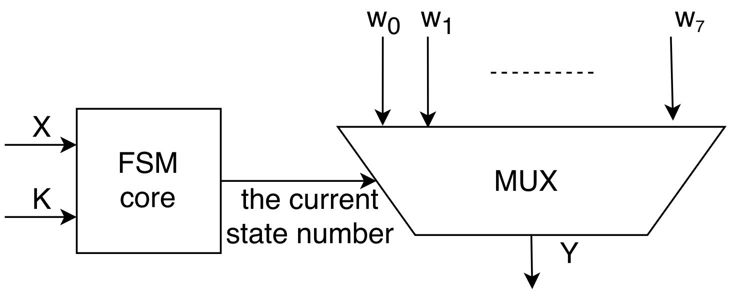


Figure 3. SC based implementation using 2-D FSM.

approximation which achieves good accuracy. However, this approach requires a stochastic to binary converter used in the pipelined system as a look-up table to store coefficients.

2.3 Stochastic BSNN with a Surrogate Gradient

This section presents the training process for BSNNs using surrogate gradient backpropagation with residual connections. In our training model, the binarized weights are represented in bipolar format (i.e., ± 1), as introduced in [67]. The input information is encoded by using SNG [100].

In the training model, we use the conventional integrate-and-fire (IF) model, where the membrane potential $u_i^{t,l}$ can be expressed as follows

$$u_i^{t,l} = u_i^{t-1,l} + \frac{\gamma}{\sigma}\left(\sum_{j=1}^{M} w_{ij}^{l} \cdot s_j^{t,l-1} - \mu\right) + \beta \tag{1}$$

Here, γ and β are scaling and shifting parameters, respectively; σ and μ correspond to the standard deviation and mean of batch normalization (BN), respectively. M denotes the number of presynaptic spikes, and $s_j^{t,l-1}$ is the presynaptic spike in the j-th neuron. $w_{ij}^{l} = \alpha \cdot w_{ij}^{b,l}$ represents the latent weight of the BSNN, where $w_{ij}^{b,l} = sign(w_{ij}^{l})$ is the corresponding binary weight and $\alpha = |w_{ij}^{l}|$ is the latent weight scaling factor.

The real input data are converted into spike format using a Poisson random number generator as an SNG for the spike representation. . The generated value is proportional to the total number of spikes within T time steps. According to the IF model in (1), if the membrane potential $u_i^{t,l}$ surpasses the firing threshold θ_i^{l}, a postsynaptic spike $o_i^{t,l}$ is generated. Then, the membrane potential is reset to zero before being accumulated again. Furthermore, the cross-entropy loss function is calculated through the last output membrane potential $u_i^{T,L}$, which is expressed as follows

$$\mathcal{L}_p = \sum_{i=1}^{C} y_i \cdot \log\left(\frac{\exp\left(u_i^{T,L}\right)}{\sum_{k=1}^{C}\exp\left(u_k^{T,L}\right)}\right) \tag{2}$$

where $Y = (y_1, y_2, \dots, y_C)$ is a label vector and C is the total number of network outputs. During the training process, the loss function $\mathcal{L}_p$ is minimized by gradient descent, and the latent weight is updated as follows [100]

$$w_{ij}^{l} = w_{ij}^{l} - \eta \cdot \sum_{t=1} \frac{\partial \mathcal{L}_p}{\partial w_{ij}^{t,l}} \tag{3}$$

where η is a learning rate. $\sum_{t=1} \frac{\partial \mathcal{L}_p}{\partial w_{ij}^{t,l}}$ is the accumulated gradient over all time steps, which is calculated as in [100]:

$$\sum_{t=1} \frac{\partial \mathcal{L}_p}{\partial w_{ij}^{t,l}} = \begin{cases} \sum_{t=1} \frac{\partial \mathcal{L}_p}{\partial o_i^{t,l}} \frac{\partial o_i^{t,l}}{\partial u_i^{t,l}} \frac{\partial u_i^{t,l}}{\partial w_{ij}^{t,l}} & \text{if } 1 \le L < l \\ \sum_{t=1} \frac{\partial \mathcal{L}_p}{\partial u_i^{T,L}} \frac{\partial u_i^{T,L}}{\partial w_{ij}^{t,L}} & \text{if } l = L \end{cases} \tag{4}$$

In (4), the gradient calculation suffers from nondifferentiable spiking activities. To address this issue, an approximate gradient (i.e., a surrogate gradient) was introduced in [100, 101], which is formally expressed as

$$\frac{\partial \mathcal{L}_p}{\partial w_{ij}^{t,l}} = \delta \cdot max\left(0, 1 - \left|\frac{u_i^{t,l} - \theta_i^l}{\theta_i^l}\right|\right) \tag{5}$$

where δ typically is set to 0.3. The surrogate gradient is effective for solving nondifferentiable spiking activity. However, when the network gets deeper, the training process based on gradient descent in (3)-(4) suffers from the degradation problem [102, 103]. In the following, we present how a residual connection [102, 103] can be adopted for our BSNN to tackle the degradation issue.

3 Proposed Stochastic Computing based Activations

This part of the thesis presents the proposed hardware efficiency of several SC-based activations, including radial basis, hyperbolic tangent, and sigmoid functions. The proposed approaches are built on a 180-nm CMOS for hardware implementation and comparison. The proposed design approaches with a practical circuit solution could potentially enhance the performance of DNNs to be applied in on-edge AI applications.

3.1 Efficient RBF Architectures

In this subsection, we focus on reducing the hardware complexity and power consumption as well as improving the accuracy of RBF. The first contribution in this subsection is a proposed novel design of the RBF, which includes a compact and a high level of accuracy in the hardware design compared with other works. The second contribution is the analyses of unipolar and bipolar stochastic format applying for RBF.
The RBF kernel is a kernel in the form of radial basis functions. The RBF kernel is defined as follows

$$\begin{aligned} K_{rbf}(\mathbf{x}, \mathbf{c}) =& exp\left(-k\|\mathbf{x}-\mathbf{c}\|^2\right) \\ k =& \frac{s^2}{2\sigma^2} \end{aligned} \tag{6}$$

where σ is the width of the Gaussian, s is a scaled factor of $\mathbf{x}$ and $\mathbf{c}$. All elements in vectors $\mathbf{x}$ and $\mathbf{c}$ must be scaled in [0,1] for unipolar stochastic representation and [-1,1] for bipolar stochastic representation. Notice that the output of $K_{rbf}(\mathbf{x}, \mathbf{c})$ is unipolar stochastic representation. If there are M features in given input data, the dimension of vectors $\mathbf{x}$, and $\mathbf{c}$ are M, where $\mathbf{x} = x_1, x_2, \ldots, x_M$, $\mathbf{c} = c_1, c_2, \ldots, c_M$. The RBF in (6) can be rewritten as follows

$$\begin{aligned} K_{rbf}(\mathbf{x}, \mathbf{c}) &= exp\left(-k\sum_{i=1}^{M}(x_i - c_i)^2\right) \\ &= \prod_{i=1}^{M} exp\left(-k(x_i - c_i)^2\right) \end{aligned} \tag{7}$$

The RBF is defined as follows

$$K_{rbf}(x, c) = exp\left(-k(x-c)^2\right) \tag{8}$$

Notice that the output of function $K_{rbf}(x, c)$ is arranged in $[0, 1]$. Consider the stochastic computing of $K_{rbf}(\mathbf{x}, \mathbf{c})$ shown in Fig. 4. The whole stochastic RBF kernel is implemented by multiplying M outputs from function $K_{rbf}(x, c)$ using an AND gate. Since the accuracy of $K_{rbf}(\mathbf{x}, \mathbf{c})$ depends on the accuracy of $K_{rbf}(x, c)$ calculation, in this paper we only synthesize and evaluate the RBF performance by employing an univariate RBF $K_{rbf}(x, c)$. Additionally, the calculation of multivariate from depends on the accuracy of univariate form, but also on the correlation between bit streams.

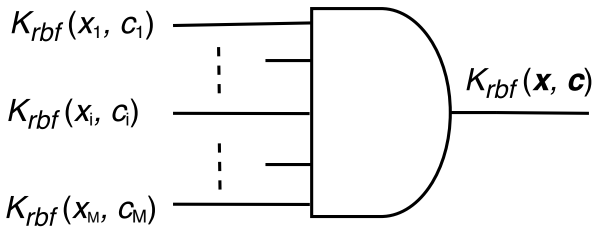


Figure 4. The RBF kernel implementation using SC.

3.1.1 Unipolar Stochastic Format for RBF

Using the definition of a limit of an exponent as the exponential function, we have

$$\begin{aligned} K_{rbf}(x, c) &= exp\left(-k(x-c)^2\right) \\ &= \lim_{n\to\infty}\left(1-\frac{k(x-c)^2}{n}\right)^n \\ &\approx \left(1-\frac{k(x-c)^2}{N}\right)^N \end{aligned} \tag{9}$$

Where N is a finite value of n. The higher the N we select the closer approximate we get for $K_{rbf}(x, c)$. Our experiment shows that the optimized value of N can be selected from 8 to 16, in this case k is not expected to exceed N. If we put $k_1 = k/N$, the (9) becomes

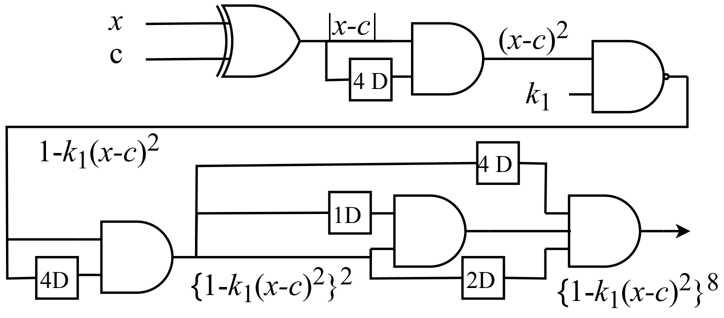


Figure 5. The proposed RBF architecture with $N = 8$.

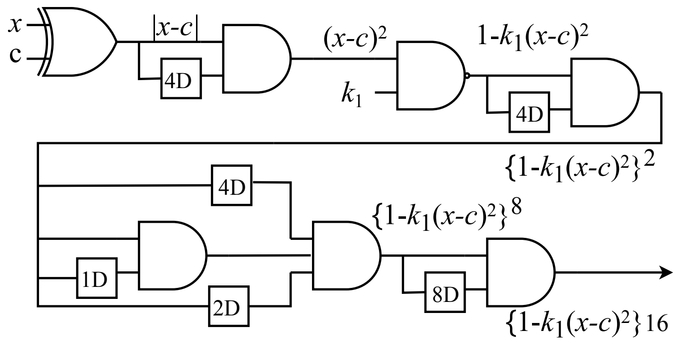


Figure 6. The proposed RBF architecture with $N = 16$.

$$K_{rbf}(x, c) = \left(1 - k_1(x - c)^2\right)^N \tag{10}$$

Fig. 5 shows the stochastic implementation of $K_{rbf}(x, c)$ using (10) with the inputs x and c in the range $[0, 1]$ for $N = 8$. Notice that N must be chosen to satisfy the range of k_1 is $[0, 1]$. Fig. 6 shows the stochastic implementation of $K_{rbf}(x, c)$ with $N = 16$. To describe in more detail for Fig. 5 and Fig. 6, the fundamental building blocks with input and output in unipolar format are illustrated in Fig. 7. The delay elements are employed for the decorrelation of all paths.

3.1.2 Bipolar Stochastic Format for RBF

Consider the case where the input is represented in bipolar format and the output is represented in unipolar format. The absolute value of a subtraction $|x - c|$ can

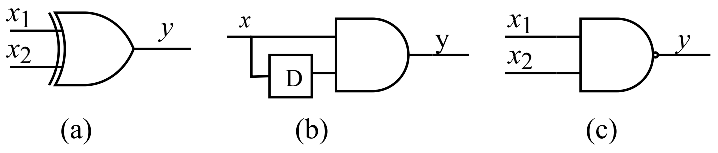


Figure 7. Fundamental unipolar stochastic computational elements.

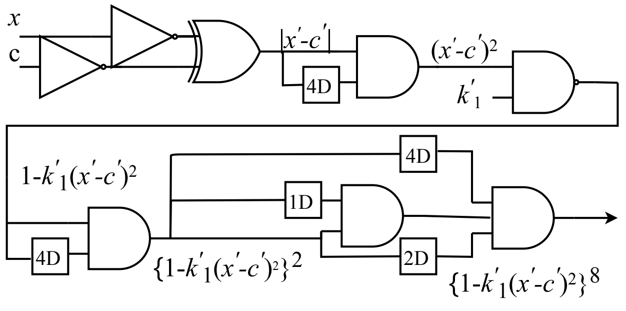


Figure 8. The proposed RBF architecture with $N = 8$ for with input in bipolar format.

be rewritten as follows

$$y = |x - c| = 2\left|\frac{1-x}{2} - \frac{1-c}{2}\right| = 2y' \tag{11}$$

if we put $\begin{cases} x' = \frac{1-x}{2} \\ c' = \frac{1-c}{2} \end{cases}$ we have $y' = |x' - c'|$. x' and c' can be simply implemented using a NOT gate with input in bipolar format and output in unipolar format [41]. Notice that x' and c' are arranged in $[0, 1]$. The XOR gate is used to perform the absolute value of a subtraction $|x' - c'|$. Then the final output is given by

$$K_{rbf}(x', c') = \left(1 - k'_1(x' - c')^2\right)^N \tag{12}$$

where $k'_1 = 4k/N$. Since the range of inputs x' and c' is [0,1] is unipolar format, it is possible to implement of $K_{rbf}(x', c')$ as in Fig. 8 for $N = 8$. The whole architecture of the univariate function $K_{rbf}(x, c)$ for $N = 8$ with input in bipolar format and output in unipolar format is illustrated in Fig. 8. Notice that the coefficient k'_1 must be arranged in $[0, 1]$. Fig. 9 shows the the software simulation result of proposed method to compared with original one-dimention RBF with k= 7 and c=0.5. The MAE is equal to 0.53 %. Fig. 10 shows the

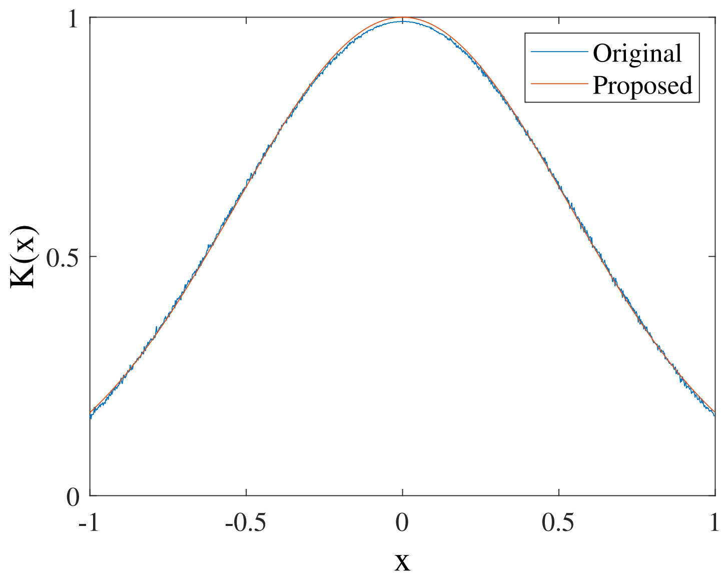


Figure 9. Simulation result of proposed method for one-dimention RBF.

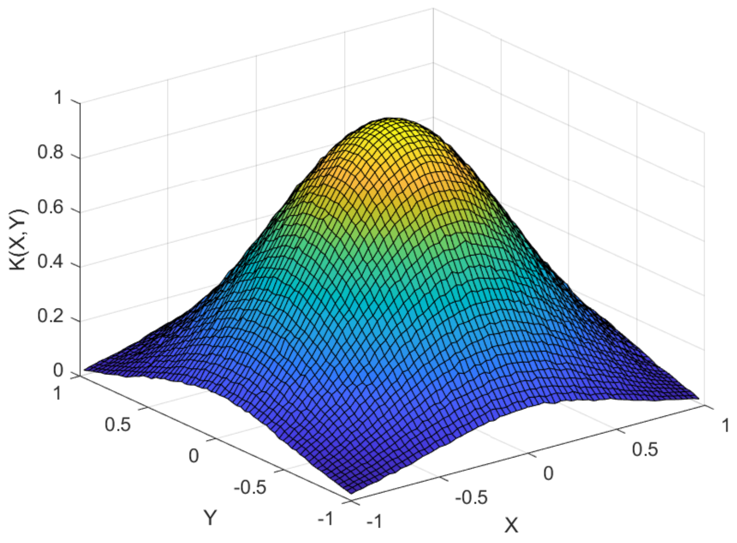


Figure 10. Simulation result of proposed method for two-dimention RBF.

the simulation result of two-dimensions RBF with k= 7 and c=0.5. The MAE is equal to 0.45 %.

3.1.3 Experiment Results and Comparison

Table 3. Hardware evaluation of proposed RBF, the 2D-FSM and Bernstein methods without sharing LFSR

METHOD	None Sharing LFSR					
	Proposed		2D-FSM		Bernstain	
	N=8	N=16	8 States	16 States	Degree 3	Degree 7
Area (μm^2)	2948	4157	6525	10644	4590	9828
Latency(ns)	3.37	3.65	3.86	4.86	3.83	5.442
Power (mW)	0.141	0.144	0.46	0.83	0.26	0.59

Table 4. Hardware evaluation of proposed RBF, the 2D-FSM and Bernstein methods with sharing LFSR

METHOD	Sharing LFSR					
	Proposed		2D-FSM		Bernstain	
	N=8	N=16	8 States	16 States	Degree 3	Degree 7
Area (μm^2)	2496	3712	5838	9412	1632	2106
Latency(ns)	2.57	2.78	3.98	4.6	2.18	3.3
Power (mW)	0.097	0.116	0.42	0.72	0.09	0.11

The Table.3 and Table.4 illustrates the hardware evaluation of RBF using the proposed method, the 2D-FSM-based implementation [40] and the Bernstein method [31] with different orders. The architectures are implemented using CMOS 180 nm libraries and synthesized using Synopsys Design Complier. In our experiments, we use 1024 bits to represent a numerical value stochastically, so the bit width of LFSRs is 10 to calculate the functions with both sharing LFSR and non sharing LFSR. In terms of hardware cost, our proposed solution required as much as the Bernstein method did. Moreover, the proposed approach outperforms the two-dimension finite state machine, roughly 54% less hardware cost. Regarding the critical path delay, the proposed approach is less 12% than others on average. Besides, the proposed architecture also required 70% less power than the two-dimension finite state machine.

Fig. 11 shows the effect on MAE of different methods by employing the non

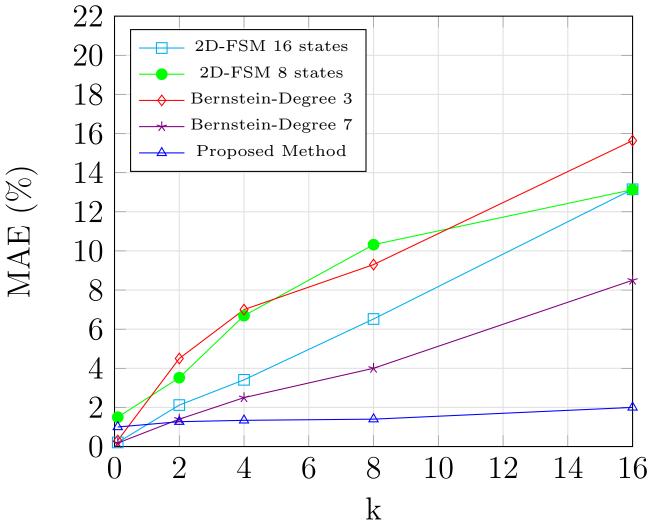


Figure 11. MAE dependence of k of the RBF employing none sharing LFSR technique.

sharing LFSR technique. The Monte Carlo simulation method was used to evaluate the MAE of different algorithms. The output results are obtained using Monte Carlo experiments for different inputs, and the experiment is repeated for different values of coefficient k. In the most optimal case, the MAE is reduced 40% and 80% compared to two other well-known approaches, Bernstein polynomial and two-dimension finite-state machine-based implementation, respectively. However, the results indicate that the different error by varying the coefficient k when employing the proposed method is slightly changed.

In this subsection, we have presented a novel architecture of stochastic computing for the univariate RBF kernel. Fig. 12 shows the MAE slightly increased by employing the sharing LFSR technique. The correlation causes this problem among SNG as they use the same LFSR. The generalized methods for unipolar and bipolar stochastic computation using the 2D-FSM-based and Bernstein methods have also been presented. However, the proposed method reached higher

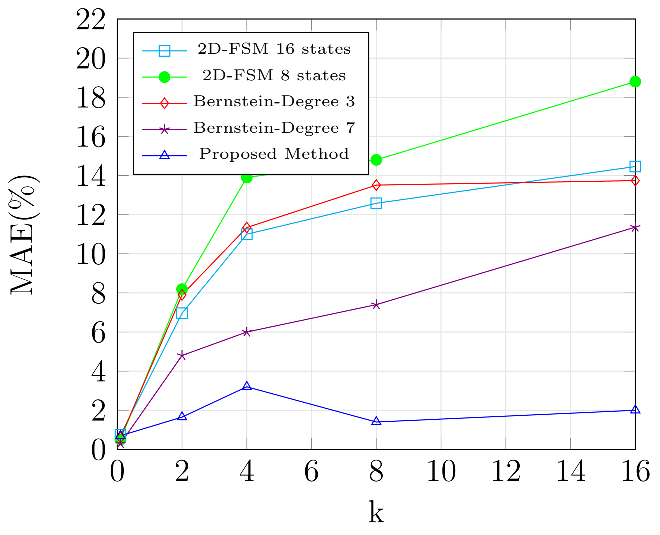


Figure 12. MAE dependence of k of the RBF employing sharing LFSR technique.

accuracy in the different cases of k, and the ASIC implementation results have clarified the improvement of the proposed method.

3.2 SC based Hyperbolic Tangent and Sigmoid Computation

3.2.1 Background SC based Berstein Polynomial

The form of SC-based Bernstein polynomial [31] is expressed as follows:

$$B(x) = \sum_{i=0}^{n} b_i B_{i,n}(x) \tag{13}$$

where b_i is a Bernstein coefficient, and n is the degree of the Bernstein polynomial. $B_{i,n}(x)$ is a Bernstein basic polynomial, which is described as follows:

$$B_{i,n}(x) = \binom{n}{i} x^i \left(1-x\right)^{n-i} \tag{14}$$

Suppose that $X_1, ..., X_n$ are the associated stochastic bit streams of the inputs $x_1, ..., x_n$ which are given by the probabilities $P(X_i = 1) = x_i = x \subseteq [0, 1]$, for $1 \leq i \leq n$. Similarly, the stochastic bit streams of the inputs $b_0, ..., b_n$ are given by the probabilities $P(B_i = 1) = b_i \subseteq [0, 1]$, for $0 \leq i \leq n$. We must find a set of coefficients $b_0, ..., b_n$ in the interval $[0, 1]$, that minimize an objective function:

$$\int_0^1 (f(x) - \sum_{i=0}^{n} b_i B_{i,n}(x))^2 d_x \tag{15}$$

Following [31], an objective function which is obtained by expanding (14) is given as follows:

$$f(\mathbf{b}) = \frac{1}{2}\mathbf{b}^T\mathbf{H}\mathbf{b} + \mathbf{c}^T\mathbf{b}, \tag{16}$$

where

$$\mathbf{b} = \begin{bmatrix} b_0, & \cdots & , b_n \end{bmatrix}^T,$$

$$\mathbf{c} = \begin{bmatrix} -\int_0^1 f(x)B_{0,n}(x)d_x \ , & \cdots & , \ -\int_0^1 f(x)B_{n,n}(x)d_x \end{bmatrix}^T,$$

$$\mathbf{H} = \begin{bmatrix} \int_0^1 B_{0,n}(x)B_{0,n}(x)d_x & \cdots & \int_0^1 B_{0,n}(x)B_{n,n}(x)d_x \\ \int_0^1 B_{1,n}(x)B_{0,n}(x)d_x & \cdots & \int_0^1 B_{1,n}(x)B_{n,n}(x)d_x \\ \vdots & \ddots & \vdots \\ \int_0^1 B_{n,n}(x)B_{0,n}(x)d_x & \cdots & \int_0^1 B_{n,n}(x)B_{n,n}(x)d_x \end{bmatrix}$$

Hence, optimizing (16) leads to a (linearly constrained) quadratic optimization. Therefore, the optimization vector $[b_0, ..., b_n]$, is obtained using quadratic programming. Generally speaking, the main circuit consists of a multiplexer, delay elements and an n bit adder. To analyze the behaviour of this circuit bahavior, firstly, we define a set $P_{X_1}, ..., P_{X_n}$ as the input data of the adder, and the set $P_{B_0}, ..., P_{B_n}$ as the input data of multiplexer, where $P_{X_i} = P(X_i = 1) = x(1 \leq i \leq n)$ and $P_{B_i} = P(B_i = 1) = b_i(0 \leq i \leq n)$. Secondly, the output of the adder $\sum_{i=1}^{n} X_i$ is used to select which input signal gets relayed to the output of multiplexer, whose value is defined by a set of the probabilities P_{B_i}. Finally, a set of delay elements $1D, 2D, ..., (n-1)D$ is used by employing $(n-1)$ D-type flip-flops respectively to all input paths for the binary adder.

3.2.2 SC based Hyperbolic Tangen and Sigmoid Functions

This section presents an approach to implementing hyperbolic tangent and sigmoid function in the bipolar format. The format conversion is embedded in our approach. The input lies in the range $[-1, 1]$. The mathematical equation of

$\tanh(ax)(a > 0)$ is described as follows:

$$\tanh(ax) = \frac{1 - \exp(-2ax)}{1 + \exp(-2ax)} \tag{17}$$

Additionally, the sigmoid function is given by:

$$\text{sigmoid}(2ax) = \frac{1}{1 + \exp(-2ax)} \tag{18}$$

From the two equations above, a relation between $\tanh(ax)$ and $\text{sigmoid}(2ax)$ is illustrated as equation below:

$$\tanh(ax) = 2\frac{1}{1 + \exp(-2ax)} - 1 = 2 \cdot \text{sigmoid}(2ax) - 1 \tag{19}$$

The bipolar format defines $x = 2P_x - 1$ in which x represents a bipolar value while P_x represent the number of ones in the corresponding bitstream. Clearly, x is in the range $[-1, 1]$ and P_x is in the range $[0, 1]$. The definition of bipolar format also suggests that a format conversion between unipolar and bipolar format is possible.

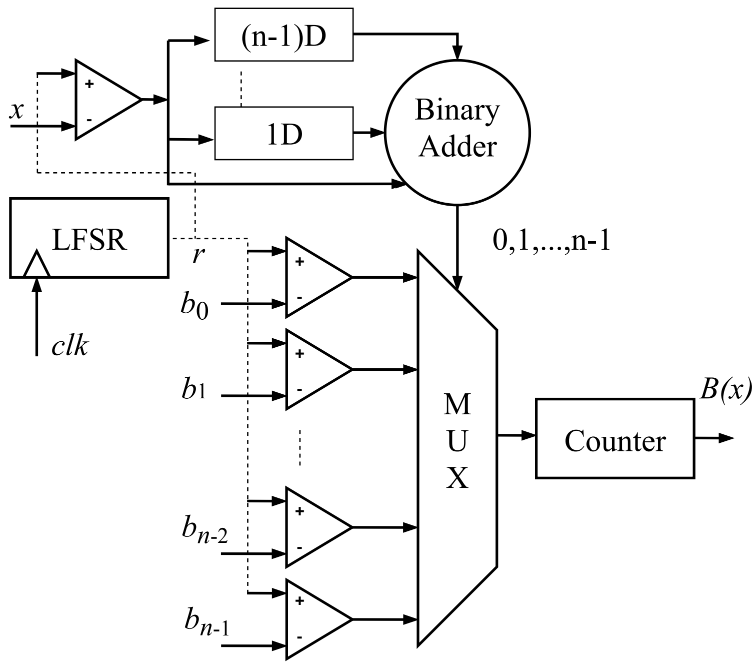


Figure 13. Stochastic implementation of $\text{sigmoid}(2ax)$ via Bernstein computation.

Given the input in the range $[-1,1]$, the output of sigmoid$(2ax) \in [0,1]$. Hence, the output of sigmoid$(2ax)$ can be represented in unipolar format. By using the same bitstream of sigmoid$(2ax)$, and applying to equation (19) which can be considered as format conversion, then the bipolar output is $\tanh(ax)$. This analysis means that the same circuit can be used to implement bot functions, by considering the output bitstreams in unipolar format for sigmoid$(2ax)$ while the same output in bipolar format for $\tanh(ax)$. The implementation of sigmoid$(2ax)$ can be done by using bipolar stochastic logic elements. However, with some simple mathematical transformations below, sigmoid$(2ax)$ can be implemented by using unipolar stochastic logic elements.

$$\begin{aligned}\text{sigmoid}(2ax) &= \frac{1}{1+\exp(-2ax)} = \frac{1}{1+\exp(-2a(2P_x-1))} \\ &= \frac{1}{1+\exp(-4aP_x)\exp(-2a)} = \frac{\exp(-2a)}{\exp(-2a)+\exp(-4aP_x)}\end{aligned} \tag{20}$$

From (20) x is substituted by $2P_x - 1$, where P_x is the unipolar value of the input bitstream X. Therefore, sigmoid$(2ax)$ can be implemented by unipolar stochastic logic while the input is still original bitstream. The approximation of equation (20) can be made by using Bernstein computation. The circuit diagram approximating sigmoid$(2ax)$ is shown in Fig. 13. In the circuit, the set of Bernstein coefficients and input x are represented in unipolar format. Binary addition is used whose output is fed to the input of the multiplexer to select which input is connected to the output. To reduce the complexity of the circuit, only one LFSR is used to generate the pseudo-random number. The uncorrelated requirement of bitstreams is solved by inserting a set of delays shown in Fig. 13.

3.2.3 Experiment Results and Comparison

This section gives the experimental results of the performance of our proposed implementation compared to the previous studies.

Table 5. Bernstein coefficients

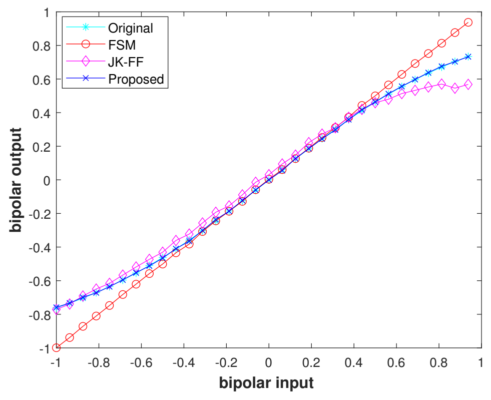
Coefficients	b_0	b_1	b_2	b_3	b_4	b_5
sigmoid($2x$)	0.12	0.2	0.34	0.66	0.8	0.87
sigmoid($4x$)	0.03	0.02	0	1	0.98	0.96

Sigmoid($2x$) and sigmoid($4x$), tanh(x) and tanh($2x$) were respectively simulated to evaluate the accuracy. Solving the quadratic programming problem in equation (16) for sigmoid($2ax$) applied equation (20), a set of Bernstein coefficients of 5^{th} order Bernstein polynomial is obtained as being shown in Table.5. The length stochastic bitstream is 1024, which means that 10-bit LFSR is used for SNG. In our simulation, the inputs of target functions are given by 0:0.03:1. The output results are collected through Monte Carlo experiments. The accuracy is evaluated via MAE. Fig. 14 shows the simulation results of approximated functions in different approaches and target functions.

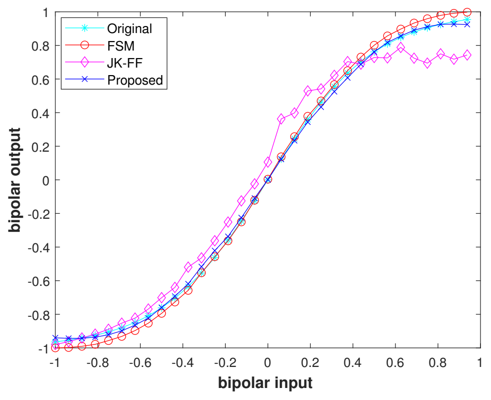
Table 6. Hardware evaluation and MAE of SC for hyperbolic tangent and sigmoid computation

Function	tanh(x) and sigmoid($2x$)				tanh($2x$) and sigmoid($4x$)			
METHOD	Proposed		FSM [34]	JK-FF [41]	Proposed		FSM	JK-FF
	D=3	D=5	2 States	–	D=5	D=7	4 States	–
Area (μm^2)	1554	1777	1345	10121	1777	2106	1551	17660
Latency(ns)	2.25	2.33	2.38	3.42	2.33	3.3	3.07	4.3
Power (mW)	0.07	0.08	0.06	0.4	0.08	0.11	0.08	0.8
MAE	0.003	0.001	0.06	0.02	0.007	0.003	0.03	0.05

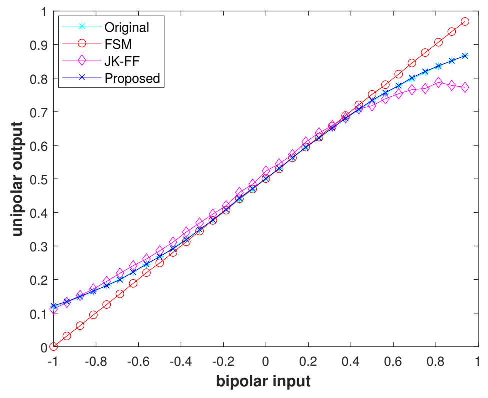
Synthesize results of the proposed function and state-of-the-arts for sigmoid($2x$) and sigmoid($4x$), tanh(x) and tanh($2x$) are considered. All architectures are implemented using $180nm$ CMOS technology node and synthesized Synopsys Design Compiler. A summarized table of power consumption, area, delay and MAE is shown in Table.6. It is noted that the same circuit can be used to implement both sigmoid($2ax$) and tanh(ax), then the same hardware cost, power consumption and MAE are achieved. In terms of accuracy, our proposed implementation is roughly 60 times and 20 times more accurate than FSM-based method and JK-FF based method, respectively, for sigmoid($2ax$) and tanh(x). Additionally, 10 and 16 times of improvement of accuracy, on average, are achieved by our proposed method in comparison to FSM and JK-FF based method, respectively, sigmoid($4x$) and tanh($2x$). Our proposed implementations are 80% and 85% less area and power consumption than the JK-FF approach. The proposed circuit was employing 3^{th} and 5^{th} order Bernstein polynomial reduced roughly 20% of critical path delay in comparison with FSM and JK-FF-based implementation.



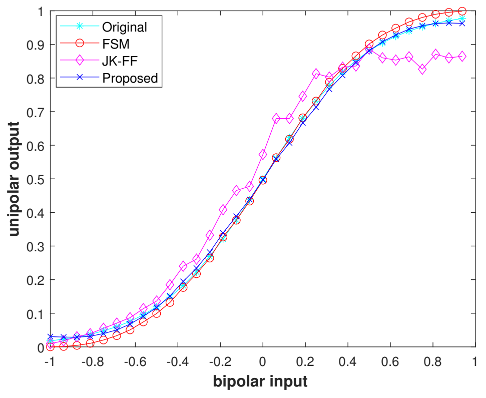
(a) tanh(x)



(b) tanh($2x$)



(c) sigmoid($2x$)



(d) sigmoid($4x$)

Figure 14. Simulation results compared different approaches with target functions.

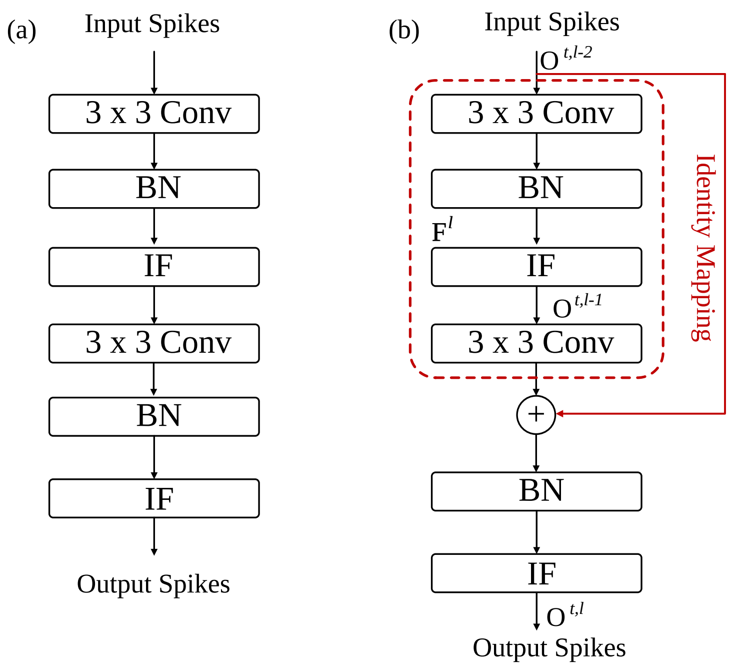


Figure 15. (a) A conventional BSNN topology, (b) a BSNN topology with residual connections using an inverter-AND spike-element-wise function.

4 In-Memory Stochastic Binary Spiking Neural Network

4.1 BSNN with Residual Connections

Residual connection is an effective technique that helps to stabilize the training processes and improve the classification accuracies of deep networks [102, 103]. We hence propose a BSNN training model using a surrogate gradient in conjunction with residual connections. As illustrated in Fig. 15, relative to the conventional BSNN network in Fig. 15(a), each convolutional (Conv) layer in the residual structure in Fig. 15(b) has an additional connection from layer (l-1) to layer l via the inverter-AND spike-element-wise (SEW) function g. The spike $s_i^{t,l}$ of layer l is now dependent on the IF output $o_i^{t,l}$ and the spike $s_i^{t,l-1}$ as follows

$$s_i^{t,l} = g(o_i^{t,l}, s_i^{t,l-1}) = (1 - o^{t,l}) \wedge s^{t,l-1} \tag{21}$$

By supporting the SEW function in (21), if $o^{t,l} = 0$, the output of the element-wise function becomes $s^{t,l} = s^{t,l-1}$, which satisfies the identity mapping condition.

4.2 In-Memory Stochastic BSNN on STT-MRAM

This section presents a BSNN inference model that utilizes the MAC operation based on an XNOR array; the latter is suited for implementation in memory using emerging technologies [85, 104]. To simplify the inference model in (1) without accuracy degradation, we set $\gamma = 1$ and $\beta = 0$ in the BN layers [105]. The original IF model for a BSNN is expressed as

$$\begin{aligned} \text{Integration } &: u_i^{t,l} = u_i^{t,l-1} + \frac{\alpha}{\sigma}\left(\sum_{j=1}^{M} w_{ij}^{b,l} \cdot s_j^{t,l-1} - \frac{\mu}{\alpha}\right), \\ \text{Firing } &: o_i^{t,l} = \begin{cases} 1 & \text{if } u_i^{t,l} > \theta_i^l \\ 2 & \text{otherwise} \end{cases}, \\ \text{Resetting } &: u_i^{t,l} = 0. \end{aligned} \tag{22}$$

In this model, during every time step, the membrane potential $u_i^{t,l}$ accumulates with $\frac{\alpha}{\sigma}\left(\sum_{j=1}^{M} w_{ij}^{b,l} \cdot s_j^{t,l-1} - \frac{\mu}{\alpha}\right)$ and then is compared with a threshold θ_i^l for the firing decision. To avoid multiplication in (22), which could be costly to implement at the circuit level, we scale both the membrane potential and the threshold by a factor of $\frac{\alpha}{\sigma}$. Given that, the scaled membrane potential $\hat{u}_i^{t,l}$ and the threshold $\hat{\theta}_i^l$ can be rewritten as

$$\begin{aligned} \hat{u}_i^{t,l} &= \hat{u}_i^{t,l-1} + \sum_{j=1}^{M} w_{ij}^{b,l} \cdot s_j^{t,l-1} - \frac{\mu}{\alpha}, \\ \hat{\theta}_i^l &= \frac{\sigma}{\alpha} \cdot \theta_i^l. \end{aligned} \tag{23}$$

The MAC component $\sum_{j=1}^{M} w_{ij}^{b,l} \cdot s_j^{t,l-1}$ in (23) is essentially the most computationally extensive operation. On the other hand, in the binary spiking model, a spike signal is represented in unipolar format (0, 1) [106, 107], while the weights $w^{b,l}$ are trained with a bipolar format (-1, 1). Therefore, unlike in the case of a BNN [85, 104], it is not possible to directly utilize an XNOR array for a BSNN MAC function. To overcome this issue, assuming that the M weights in (23) have M_1 negative weights (w_{ij}^{-b}) and $M - M_1$ positive weights (w_{ij}^{+b}), the MAC function in (23) can be reformulated as

$$\begin{aligned}\sum_{j=1}^{M} w_{ij}^{b,l} \cdot s_j^{t,l-1} &= \sum_{j=1}^{M} \overline{(1 + w_{ij}^{-b}) \oplus s_j^{t,l-1}} - M_1 + \sum_{j=1}^{M-M_1} \overline{(1 + w_{ij}^{+b}) \oplus s_j^{t,l-1}} \\ &= \sum_{j=1}^{M} \overline{w_{ij}^{u,l} \oplus s_j^{t,l-1}} - M_1.\end{aligned} \tag{24}$$

In (24), $w_{ij}^{u,l}$ is represented in unipolar format ($w_{ij}^{u,l} = \frac{(w_{ij}^{b,l}+1)}{2}$). Substituting the expression of (24) into (23) and denoting $\rho = M_1 + \frac{\mu}{\alpha}$, the scaled membrane potential in (23) is now expressed as

$$\hat{u}_i^{t,l} = \hat{u}_i^{t,l-1} + \sum_{j=1}^{M} \overline{w_{ij}^{u,l} \oplus s_j^{t,l-1}} - \rho \tag{25}$$

In (25), the component $\sum_{j=1}^{M} \overline{w_{ij}^{u,l} \oplus s_j^{t,l-1}}$ can be realized entirely in memory by using an XNOR array. However, this transformation introduces a constant ρ in (25) implies that the hardware implementation must support both accumulation and subtraction. The latter leads to undesirable complexity in the circuit implementation. To solve this problem, we propose an equivalent IF model with a dynamic threshold mechanism. Specifically, instead of a fixed threshold, we can transform the negative component ρ of the scaled membrane in (25) into a positive component of the threshold $\hat{\theta}_i^l$, which is now considered the time-dependent quantity $\hat{\theta}_{dyn,i}^{t,l}$. Therefore, the proposed IF model can be formulated as follows

$$\begin{aligned}\text{Integration } &: \begin{cases}\hat{u}_{xnor,i}^{t,l} = \hat{u}_{xnor,i}^{t,l-1} + \sum_{j=1}^{M} \overline{w_{ij}^{u,l} \oplus s_j^{t,l-1}} \\ \hat{\theta}_{xdyn,i}^{t,l} = \hat{\theta}_{dyn,i}^{t,l-1} + \rho, \hat{\theta}_{dyn,i}^{t=0,l} = \hat{\theta}_i^l\end{cases} \\ \text{Firing } &: o_i^{t,l} = \begin{cases}1 & \text{if } \hat{u}_i^{t,l} > \hat{\theta}_i^l \\ 0 & \text{otherwise}\end{cases} \\ \text{Resetting } &: \hat{u}_i^{t,l} = 0\end{aligned} \tag{26}$$

Compared to the IF model in (22), the IF model in (26) requires only accumulation operations. This means that the latter model can help to simplify the

Table 7. MTJ Parameters

MTJ size (W $\times$ L)	**60 nm $\times$ 60 nm**
MTJ thickness (Tm)	**1.5 nm**
Oxide thickness (TMgO)	**1.15 nm**
Relative MTJ resistance variability	**5 %**
Nominal R_MTJ at P (AP)	**2 k Ω (4 k Ω)**
TMJ	**100 %**

subsequent hardware implementation. However, the model in (26) is correct only when ρ is positive. Although it rarely occurs, the value of ρ can theoretically be negative, i.e., the subtraction in (25) is an addition. In that case, we retain the original expression in (25) and keep the threshold constant. Accordingly, a small modification is required in the IF neuron circuit implementation (detailed in Section IV) for covering both positive and negative values of ρ.

4.2.1 XNOR-based Complementary 2T-2R STT-MRAM

Table 7 shown the fundamental MTJ parameters used in this work. The complimentary XNOR circuit as shown in Fig. 16 (presented in [85]) is used in this work. It includes two programable magnetic junctions $MTJ, 0$ and $MTJ, 1$, are represented by high resistance (anti-parallel magnetization resistance R_{AP}) and low resistance (parallel magnetization resistance R_P) to store the binarized weights (+1, 0). The compliment bitlines (BLs) are applied to the first terminal of $MTJ, 0$ and $MTJ, 1$ to represent a single input spike.

For simplified analysis, the pair bitlines $V_{BL}, 0 > 0$ and $V_{BL}, 1 = 0$ corespond the high state (bit 1) input spike access to junctions $MTJ, 0$ and $MTJ, 1$ in a single bitcell are applied. This means that only $R_{MTJ,0}$ contributes to the $I_{bitcell}$. $R_{MTJ,0}$ detemines $I_{bitcell}$, hence, the stored weight represent $I_{bitcell}$ the XNOR between bitcell weight and input spike. The ovelrall resistance formed by the bicell SL terminal is a data-independent and defined by $(R_{MTJ,0} + R_{access})||(R_{MTJ,1} + R_{access})$.

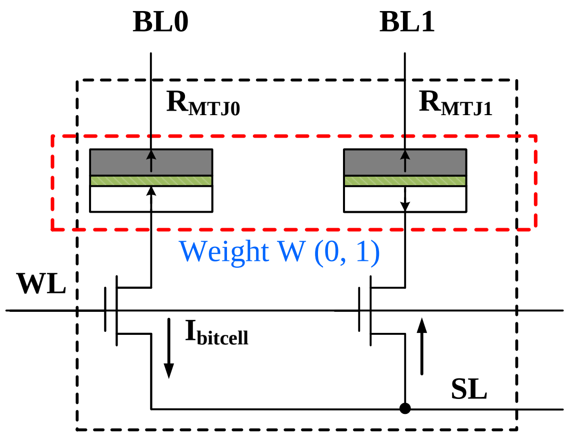


Figure 16. XNOR-based complementary 2T-2R STT-MRAM structure.

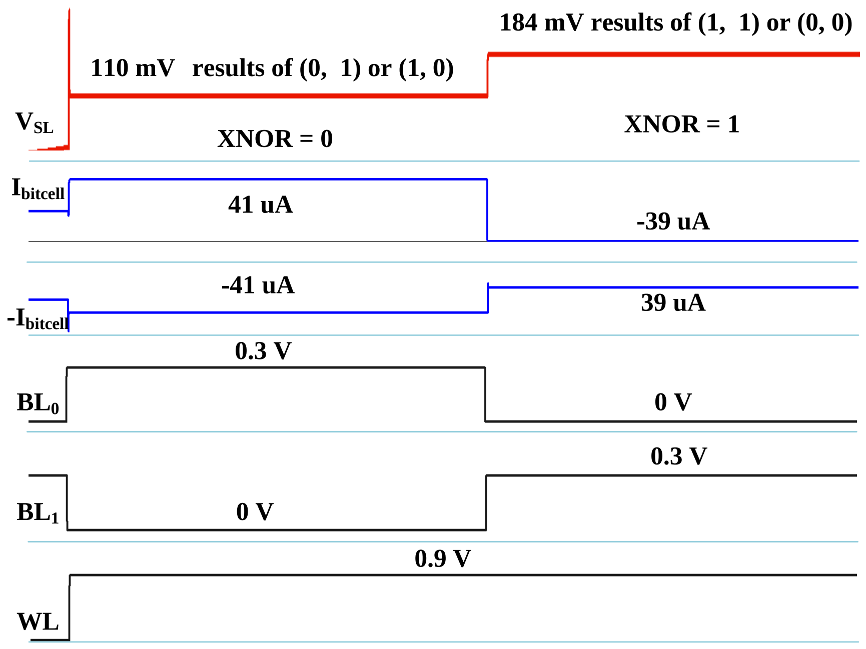


Figure 17. SL voltages and XNOR bitcell currents for (+1) and (0) output.

In a different context, the V_{SL} formed by $R_{bitcell}$ passes $I_{bitcell}$ to the single bitcell at the position (i, j) in-memory array can be expressed as follows [85]

$$V_{SL,bitcell} = R_{bitcell} I_{bitcell} = \frac{V_{BL}}{X_{i,j}} \tag{27}$$

where $X_{i,j}$ is detemined as follows

$$X_{i,j} = \begin{cases} \dfrac{R_P + R_{access}}{R_{bitcell}} & \text{if } \overline{W_{i,j} \oplus IN_j} = +1 \\ \dfrac{R_{AP} + R_{access}}{R_{bitcell}} & \text{if } \overline{W_{i,j} \oplus IN_j} = 0 \end{cases} \tag{28}$$

where $W_{i,j}$ denotes the bitcell stored weight and IN_j is a binary spike corespond collum j in-memory array. In a deeper analysis, Fig. 17 shows the circuit simulation of the BSNN bitcell. As intended, the maximum read current is bellow 50 μA. The simulation conditions are $V_{WL} = 0.9V$ and $V_{BL} = 0.3$, and the maximum MTJ current has $I_{bitcell} = 41\mu A$. At this maximum $I_{bitcell}$, the sourse line volatge $V_{SL} = 110mV$ corespoding the combination (+1, 0) or (0, +1), while $V_{SL} = 184mV$ resulting from (+1, 0) or (0, +1).

4.2.2 Spiking MAC operations Using XNOR-based STT-MRAM

This part presents the circuit implementation for the BSNN model in (26). The general architecture for intra-layer processing using the proposed model is shown in Fig. 18. First, the binarized weight $w_{ij}^{u,l} \equiv w_{ij}$ is mapped into the memory of the $N \times M$ STT subarray. Then, the digital presynaptic spikes $s_j^{t,l-1}$ are encoded by the column decoder and fed to the array through bit line BL_j $(j = 1 - M)$ to fit the XNOR-MAC computation. Finally, the source line (SL) SL_i $(i = 1 - N)$ voltage, which represents the output of the MAC operation, is passed into the IF model in (26) to generate postsynaptic spikes $o_i^{t,l}$. To map an MAC computational unit into the IMC memory, we employ a 2T-2R STT-MRAM-based XNOR cell in subsection 4.2.1. Updating the binary weights is performed at the beginning. Since SL is shared among the cells in the same row, each MTJ has to be written individually. Specifically, apart from the BL corresponding to the active MTJ, other BLs are left in the high-impedance state while an appropriate voltage is applied across (BL, SL) for flipping the MTJ magnetization. The write peripheral circuit is omitted for clarity, details can be

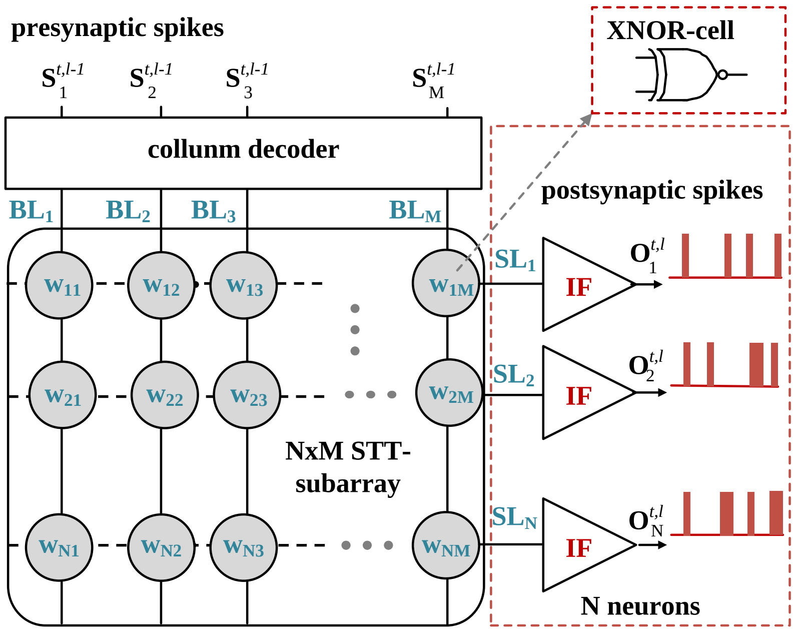


Figure 18. BSNN architecture for intra-layer processing using an XNOR cell array.

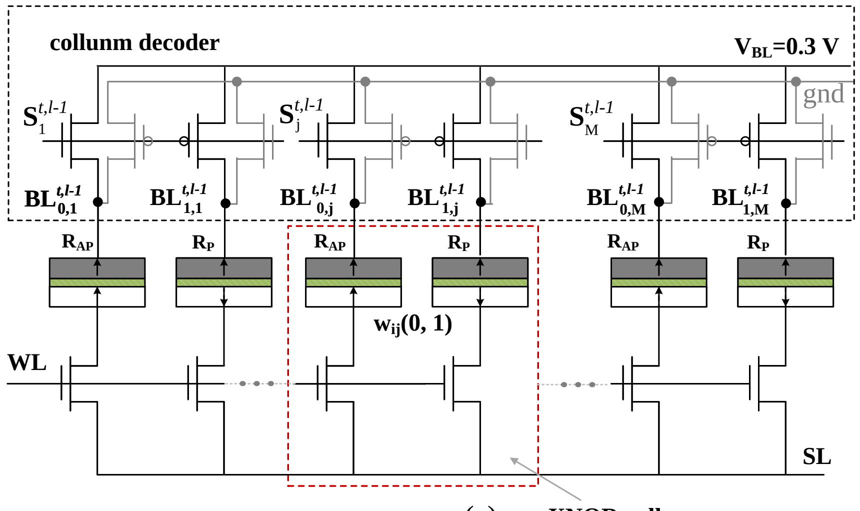
found in [108, 109]. As seen in Fig. 19(a), for a single XNOR bitcell, the binarized weight (0, 1) is encoded by the MTJ states (R_AP and R_P), and the presynaptic spike is encoded to a pair of BL voltages as follows

$$s_j^{t,l-1} = \begin{cases} 0 & \text{if } BL_{0,j}^{t,l} = 0(V), BL_{1,j}^{t,l} = V_{BL} \\ 1 & \text{if } BL_{0,j}^{t,l} = V_{BL}, BL_{1,j}^{t,l} = 0(V) \end{cases} \tag{29}$$

The BL driver encodes incoming spikes using a pair of complementary transistors (an n-channel MOS (NMOS) and a p-channel MOS (PMOS)). The SL voltage represents the output of a single XNOR operation (see Fig. 19(b)). Fig. 19(a) shows the circuit implementation for a single-row XNOR-based BSNN using STT-MRAM.

Each high-load WL is driven by a buffer (an 4-stage inverter chain) that guarantees the fast transition and stable level value during the MAC operation on the memory row. Additionally, from [85], the MAC product in the output of the i-*th* row connection is represented by the merged SL voltage $V_{SL,i}^{t,l}$ (i.e., all bitcells in a row share the same SL), which is equal to [85]

$$V_{SL,i}^{t,l} = V_{BL}\left(\frac{M-K}{M}\cdot\frac{R_{bitcell}}{R_{AP}+R_{access}} + \frac{K}{M}\cdot\frac{R_{bitcell}}{R_P+R_{access}}\right) \tag{30}$$

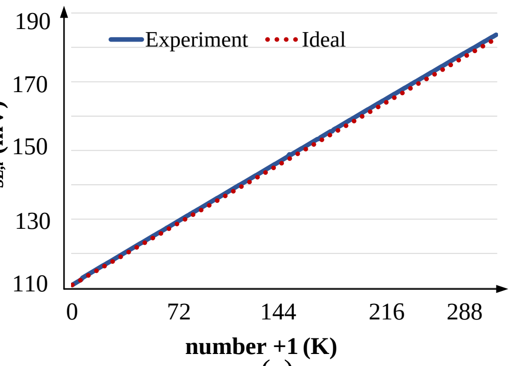


(a)

Equivalent to XNOR operation for a 2T-2R bitcell

MTJ_0, MTJ_1 / BL_0, BL_1	AP, P (0)	P, AP (1)
V_{BL}, 0 (1)	Low (0)	High (1)
0, V_{BL} (0)	High (1)	Low (0)

(b)



(c)

Figure 19. (a) A single STT-MRAM row based on 2T-2R STT-MRAM bitcells for realizing binarized MAC operations, (b) the SL voltage level corresponding to the XNOR operation for a single 2T-2R bitcell and (c) the dependence of the SL voltage on the number of (+1) values among the XNOR outputs (K) of the circuit simulation for a row of 288 bitcells.

where $R_{bitcell}$ is the overall resistance seen from the SL terminal of the bitcell,

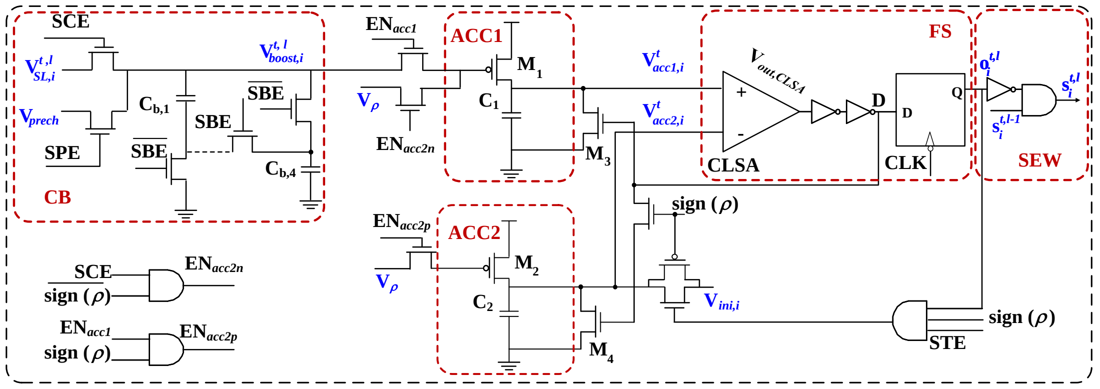


Figure 20. The IF neuron and SEW circuit of the proposed XNOR-based BSNN inference with STT-MRAM synapses.

which is data-independent and equal to $(R_{AP}+R_{access})||(R_P+R_{access})$; K denotes the number of XNOR outputs (i.e., $\overline{w_{ij}^{u,l} \oplus s_j^{t,l-1}}$) equal to +1 across the entire row of M bitcells. The SL voltage linearly depends on K and ranges from $V_{BL} \cdot \frac{R_{bitcell}}{R_{AP}+R_{access}}$ $(K=0)$ to $V_{BL} \cdot \frac{R_{bitcell}}{R_P+R_{access}}$ $(K=M)$.

In Fig. 19(c), we plot the circuit simulation of $V_{SL,i}^{t,l}$ with respect to K. The simulation results show that $V_{SL,i}^{t,l}$, which ranges from 110 mV to 184 mV, is linearly dependent on K. This confirms that the MAC calculation (26) can be performed within a single in-memory access phase. In the following, we introduce an approach for implementing an IF neuron mechanism (the model in (26)) using circuit computation in the charge domain, whose input is the SL voltage $V_{SL,i}^{t,l}$ from the MAC operation.

4.2.3 IF neuron and Spike-Element-Wise Circuit Designs

Fig. 20 shows the proposed implementation of the IF neuron and SEW circuit architecture. The IF neuron circuit consists of two charge-based accumulations (ACC1 and ACC2), a capacitive voltage booster (CB), and firing and shaping (FS) circuits. The charge-based accumulations are used to update the membrane potential and the dynamic threshold. As discussed earlier, the result of an MAC operation is represented in the form of a voltage at the merged SL $V_{SL,i}^{t,l}$. Subsequently, this voltage must be accumulated in every time step, followed by the IF model in (26). However, since $V_{SL,i}^{t,l}$ varies from 110 mV to 184 mV, it must

be amplified to an adequate level to limit the charging current. If $V_{SL,i}^{t,l}$ is used directly to control the charging current (the drain current of the PMOS M1), the accumulated charge on C1 can quickly reach the saturation level (i.e., $V_{DD} \times C$, where C is the capacitance of C1) because the M1 transistor is almost at the full-driving state. The amplification process is performed by the CB circuit introduced in [110], considering the amplification factor (G) does not require high precision, and the CB circuit is very compact and energy efficient. As seen in Fig. 21, $V_{SL,i}^{t,l}$ is sampled before being fed into the booster circuit, the sampling time τ_{sample} for this array size is chosen to be 1 ns (when SCE=1). The capacitors in the CB are precharged by V_{prech} up to the middle level of $V_{SL,i}^{t,l}$ (150 mV) to optimize the timing, as detailed in [110]. The precharging time $\tau_{precharge}$ is set to 0.5 ns (when SPE=1). After boosting, $V_{boost,i}^{t,l}$ is maintained for the duration of τ_{boost}=4.5 ns (when SBE=1). During that time, when the signal EN_{acc1} is activated, $V_{boost,i}^{t,l}$ is connected to the gate of M1 transistor for charging C_1 within a fixed duration of time τ_{charge}=1 ns. The additional amount of charge in C1 hence is proportional to $V_{boost,i}^{t,l}$ and to $V_{SL,i}^{t,l}$. The voltage level across C1 is equal to $V_{acc1,i}^{t} = \frac{Q_{acc1,i}^{t}}{C}$, which indirectly represents the accumulation of $V_{SL,i}^{t,l}$ at time step t. Similarly, ACC2 is utilized for dynamic threshold accumulation. Assume that ρ¿0, and V_ρ represents the value of ρ in the voltage domain. As shown inFig. 21, when EN_{acc2p} is activated, V_ρ is used to charge C2 through M2 ($EN_{acc2p} \equiv EN_{acc1}$). The amount of additional charge corresponds to the voltage increment at the output $V_{acc2,i}^{t}$ of capacitor C2. In such a way, $V_{acc2,i}^{t}$ represents the dynamic threshold accumulation corresponding to the model in (26) .
Finally, $V_{acc1,i}^{t}$ and $V_{acc2,i}^{t}$ are fed into the current latched sense amplifier (CLSA) [111] circuit for a voltage level comparison. If the firing condition in (26) is satisfied $V_{acc1,i}^{t} > V_{acc2,i}^{t}$, a spike is generated in the output of the CLSA ($V_{out,CLSA}$), as an example shows in Fig. 21. Subsequently, $V_{out,CLSA}$ is shaped by two inverters before being fed into a D-flip-flop (D-FF) for the postsynaptic generation of spike $o_i^{t,l}$. The frequency of the clock (CLK) signal determines the postsynaptic spike period (T_{spike}=6 ns). Additionally, after two inverters, a signal D, which is $V_{out,CLSA}$ delayed by two inverters, is used for resetting ACC1 and ACC2 before starting the next step operation. To reset the dynamic threshold (after firing and during initialization), C2 is precharged by $V_{ini,i}$, which represents the threshold

θ_i^l (see (26)) in the voltage domain. According to Fig. 20, that C2 is precharged when both the STE and output of the D-FF are equal to 1, where STE is a periodic signal that enables the threshold precharging circuit for a fixed duration within a time step. In the normal working mode, the STE is activated when a generated postsynaptic spike occurs. For example, $V_{acc2,i}^t$ is precharged to $V_{ini,i}$ ($14.7mV$) within the duration τ_{ini} (0.5 ns) after firing, as seen in Fig. 21. During the initialization process, the D-FF output is manually set to '1' to preset the threshold. After IF processing, the output of the D-FF is fed into the SEW circuit to perform residual connection according to the model in (21). Specifically, $o_i^{t,l}$ is added with the spike from the previous layer $s_i^{t,l-1}$ using a single inverter-AND gate, as shown in Fig. 20. Note that in some rare cases, if ρ <0 (sign (ρ)$\equiv$ 0) (see (26)), EN_{acc2p} is deactivated, and EN_{acc2n} is active. In such cases,V_ρ accumulates in ACC1 instead of ACC2. In other words, the output of ACC2 is fixed as $V_{ini,i}$. This switching mechanism recalls our discussion about ρ in subsection 4.2.
The detailed charge-based accumulation, analysis and circuit implementation of the core functions are described below. As shown in Fig. 20, the charge-based accumulation architecture consists of a PMOS transistor and a capacitor. According to the well-knownα-power law model [112], the charging current I_{ds}^t passing through transistors M1 and M2 is equal to

$$I_{ds}^t = \frac{\mu \cdot C_o x \cdot W}{L}(V_{GS} - V_{TH})^\alpha \tag{31}$$

where α is the model power index, which ranges from 1-2 depending on the adopted technology. In our chosen technology (a 65-nm CMOS), the transistor is considered a short channel; i.e., theoretically, α ≈1.

We further experimentally verify that the charging current is quasi-linear and dependent on V_{GS} in the range of interest (V_{GS} from -702 mV to -443 mV). This fact is very important and allows the proposed model in (26) to be directly mapped to the circuit solution. Note that the value of $V_{SL,i}^{t,l}$ and its boosted value $V_{boost,i}^{t,l}$=$G \cdot V_{SL,i}^{t,l}$ (G ≈3.6) are constants within a time step duration. The charging current I_{ds}^t in (31) can therefore be considered unchanged during the accumulation time. Thus, the amount of charge accumulated in C_1 in the time

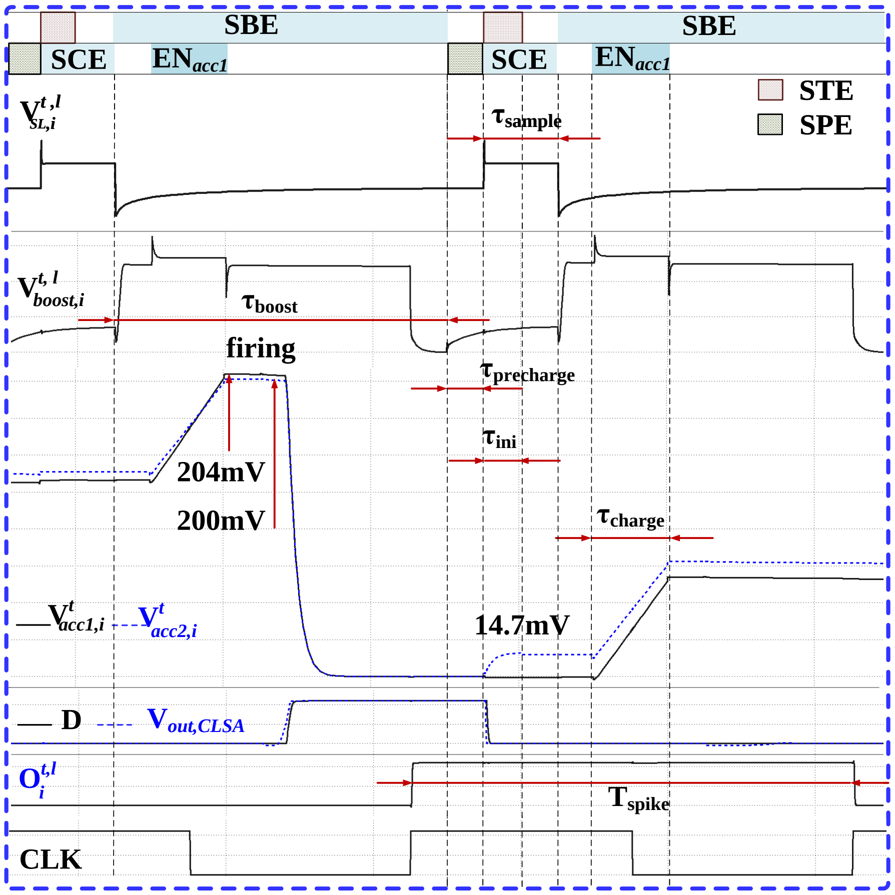


Figure 21. The IF neuron circuit simulation waveform within 2 time steps for a row with 288 bitcells.

step from t-1 to t is equal to

$$\begin{aligned}\triangle Q_{acc1,i} &= \int_{t-1}^{t} I_{ds}^{t}\, dt \\ &= \frac{\mu \cdot C_o x \cdot W \cdot \tau_{charge}}{L}(G \cdot V_{SL,i}^{t,l} - V_{DD} - V_{TH}) = \beta \cdot V_{SL,i}^{t,l} + \varphi.\end{aligned} \tag{32}$$

Here, $\beta = G \cdot \frac{\mu \cdot C_{ox} \cdot W \cdot \tau_{charge}}{L}$ and $\varphi = -\frac{\mu \cdot C_{ox} \cdot W \cdot \tau_{charge}}{L}(V_{DD} + V_{TH})$. Accordingly, the amount of charge in C_1 at time step t equals to

$$Q_{acc1,i}^{t} = Q_{acc1,i}^{t-1} + \triangle Q_{acc1,i} \tag{33}$$

By replacing $V_{acc1,i}^{t} = \frac{Q_{acc1,i}^{t}}{C}$ and $\triangle V_{acc1,i}^{t} = \frac{\triangle Q_{acc1,i}}{C}$ (in (33)), the accumulated voltage at the second plate of the capacitor equals

$$V_{acc1,i}^{t} = V_{acc1,i}^{t-1} + \triangle V_{acc1,i}^{t} \tag{34}$$

Note that $\triangle V_{acc1,i}^{t}$ represents the voltage increment in ACC1 when $V_{SL,i}^{t,l}$ is applied to the input analog accumulation. For a dynamic threshold, the circuit implementation is almost the same, but $V_{SL,i}^{t,l}$ is replaced with V_{ρ} in (32)-(34). We have

$$V_{acc2,i}^{t} = V_{acc2,i}^{t-1} + \frac{\beta}{C} \cdot V_{\rho} + \frac{1}{C} \cdot \varphi \tag{35}$$

The dynamic threshold can be reformulated as follows

$$V_{acc2,i}^{t} = V_{acc2,i}^{t-1} + \triangle V_{acc2,i}^{t} \tag{36}$$

where $\triangle V_{acc2,i}^{t} = \frac{\beta}{C} \cdot V_{\rho} + \frac{1}{C} \cdot \varphi$. The equations for $V_{acc1,i}^{t}$ and $V_{acc2,i}^{t}$ in (34) and (36), respectively, in the circuit domain hence can be mapped to the model in (26).

4.2.4 The effect of Non-linearity and Process Variation

As introduced in (34) and (36), the IF model in (26) can be directly mapped to the charge-based accumulation circuit, which is introduced in Fig. 20. However, this model suffers from inevitable nonlinearity, and process variation comes from both the CMOS and MTJ devices. These effects essentially degrade the IF accuracy, as well as the overall system performance. In this section, we quantify this nonideal impact based on actual circuit simulations, aiming to have a realistic evaluation of the proposed BSNN model at the system level in the subsequent section. To capture the effect of process variation, we run Monte Carlo simulations for a synaptic array (a row of STT-MRAM) in Fig. 19 (a) and the proposed IF neuron circuit in Fig. 20. The variation in the CMOS device is set according to the provided by foundry models. For the MTJ, the variability in the MTJ resistance is approximately 5% according to [113]. The accumulated output $\triangle V_{acc1,i}^{t}$ depends on K (the number of XNOR outputs that are equal to +1), as presented in subsection 4.2.1. Each IMC row is designed for 288 bitcells that later fit with the BSNN inference model. Monte Carlo simulations have been performed for 289 cases of K. Fig. 22 plots the results of mapping K to $\triangle V_{acc1,i}^{t}(K)$ under

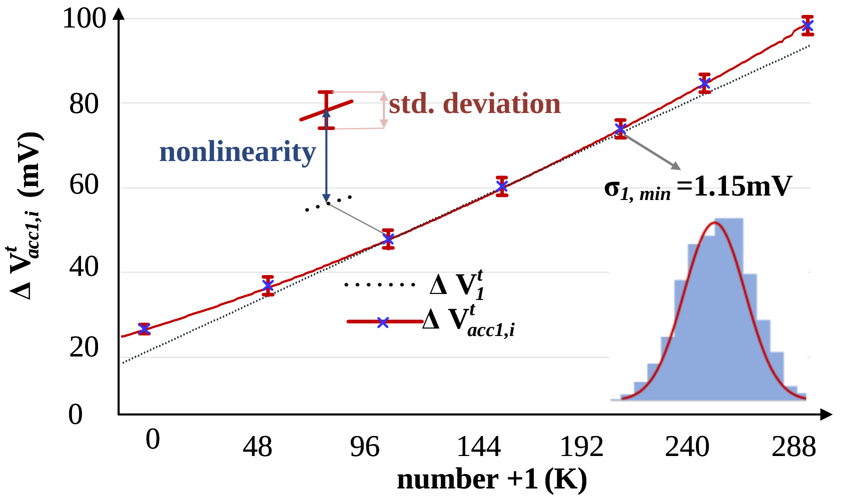


Figure 22. The effects of nonlinearity and variations on $\triangle V^t_{acc1,i}$ (K) with nonlinear errors δ_1 (K) and standard deviations σ_1 (K) with respect to the number of XNOR outputs (equal to +1 (K) in a row with 288 bitcells).

nonlinear and variation effects. The capacitances C in ACC1 and ACC2 are set to 150 fF and 100 fF for all capacitors in the booster circuit. The PMOS sizes in ACC1 and ACC2 are set to W=8×W_{min}, L=7× L_{min}. The NMOS size in the array is set to W=4×W_{min}, L = L_{min}. From Fig. 22, we can see the effect of nonlinearity on $\triangle V^t_{acc1,i}(K)$ (the solid line). As shown in Fig. 22, $\triangle V^t_{acc1,i}(K)$ is quasi-linearly dependent on K. From the simulation data, the difference $\delta_1(K) = \triangle V^t_{acc1,i}(K) - \triangle V^t_1(K)$ between the ideal linear value $\triangle V^t_1(K)$ (the dashed line) and the simulated $\triangle V^t_{acc1,i}(K)$ is negligible in the middle but becomes significant near the boundary. For example, $\delta_{1,max}$ (0)=7.27 mV, and $\delta_{1,min}$ (144)=0.01 mV. In Fig. 22, we also present the statistical analysis of $\triangle V^t_{acc1,i}(K)$. The statistical results show that the variations of $\triangle V^t_{acc1,i}(K)$ can be approximately fit to Gaussian distributions with a standard deviation $\sigma_1(K)$. The results indicate that $\sigma_1(K)$ is not the same for every K. This effect is understandable from the circuit perspective, where the working points of the M1 transistor for different values of K are not the same. For example, $\sigma_1(K)$ reaches its maximum value at 3.60 mV for K=2 and its minimum value at 1.15 mV

for K=221. The noise profile, represented by a normal distribution $n(0,\sigma_1(K))$ is added to the ideal value $\triangle V_{acc1,i}^t$. The actual voltage level in the output of ACC1 can be approximated as

$$\triangle V_{acc1,i}^t = \triangle V_1^t(K) + \delta_1(K) + n(0, \sigma_1(K)) \tag{37}$$

A similar analysis and model are conducted for $\triangle V_{acc2,i}^t$. The difference is that the variation $\triangle V_{acc2,i}^t$ comes from the ACC2 circuit itself without contributions from the synapse circuits (i.e., the memory array). We have

$$\triangle V_{acc2,i}^t = \triangle V_2^t(K) + \delta_2(K) + n(0, \sigma_2(K)) \tag{38}$$

Where $\triangle V_{2,\rho}^t$ is the ideal linear; $\sigma_2(\rho)$ and $n(0,\sigma_2(\rho))$ are the ρ-dependent nonlinear errors and random variation of $\triangle V_{acc2,i}^t$. Finally, the nonlinear and variation effects on $\triangle V_{acc1,i}^t$ and $\triangle V_{acc2,i}^t$ are added into the accumulations in (34) and (36). The effects of nonlinearity and process variation on the full circuit simulation are studied in the following subsection.

In this subsection, we conduct a full circuit simulation for an IF neuron circuit, which is directly connected with the M synapses of 288 STT memory cells. We extract the trained network parameters (i.e., weights, BN parameters, and threshold) from the PyTorch model and feed them to the circuit model, which is encapsulated in the HSPICE netlist. The network hyperparameters are set so that the kernel size is 3×3 and the number of input channels is 32, corresponding to an IMC array size of 1×288. Fig. 23 shows an example waveform of the CLSA output ($V_{out,CLSA}$) for a single time step considering the process variation effect. From the figure, the shift delay can fluctuate from $\tau_{delay,1}$($\approx$170 ps) to $\tau_{delay,2}$ ($\approx$740 ps). However, this variation barely affects the output spike because $V_{out,CLSA}$ is shaped and latched in a fairly stable position. Furthermore, for multiple time steps, the process variation also affects the spike position and the total number of spikes. Fig. 24 presents the postsynaptic spike waveform for T=8× T_{spike} corresponding to 288 input bitstrings. From multiple Monte Carlo samples, it is very common that the positions of the spikes in the output pattern $o_{i,simulated}^{t,l}$ are shifted back and forth from the positions of the expected software simulation output $o_{i,expected}^{t,l}$. However, since we use rate encoding, this effect does not affect the final result, which is determined by the total number of spikes within T period. It can also be observed that the output pattern may miss or introduce one

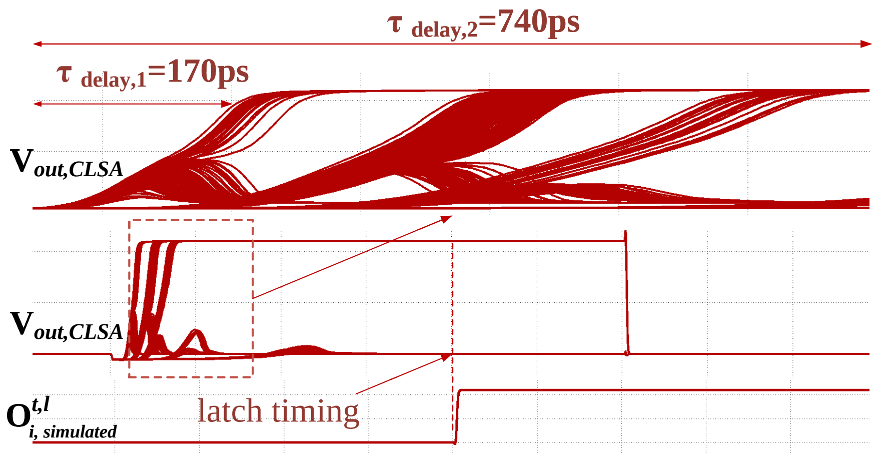


Figure 23. The effect of process variations on the output of the CLSA within a single time step.

spike, as shown in Fig. 24(b) and 24(c). Nevertheless, the undesirable missing or addition of one spike has little impact on the final result also thanks to the use of the rate encoding. In rare cases, we also observe missing or introduced two spikes, but that is the maximum number of incorrect spikes observed thus far in our simulations.

To quantify the impact of process variation on the robustness of the neuron implementation, we extracted the mean square error (MSE) of the output bitstreams from the multiple Monte Carlo simulations as follows

$$MSE = \frac{1}{N_{Monte}} \sum_{j=1}^{N_{Monte}} \frac{1}{T^2} \cdot \left(\sum_{j=1}^{T} o_{ij,expected}^{t,l} - \sum_{j=1}^{T} o_{ij,expected}^{t,l} \right) \tag{39}$$

where N_{Monte} is the number of Monte Carlo simulations. According to (39), the MSE for 500-Monte Carlo simulations in this experiment has a value of 1.45%. Thus, process variation essentially has an impact on the classification accuracy, but this impact is small and can be reasonably accepted. The results obtained from the above analysis could be a solid evidence for the SNN fault-tolerant nature. In fact, it is not possible to simulate the whole network at the circuit level with massive input patterns. However, as we have mentioned earlier, the statistical error models of $\triangle V_{acc1,i}^{t}$ and $\triangle V_{acc2,i}^{t}$ in (37) and (38) can be exactly

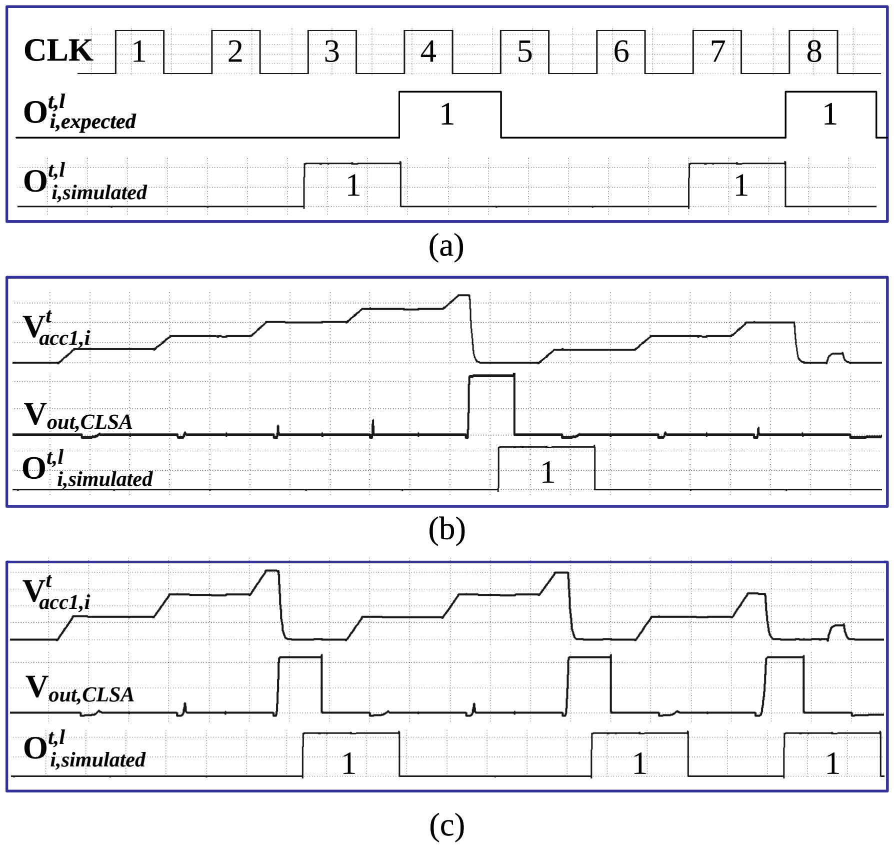


Figure 24. The effect of process variations on postsynaptic spike generation.

injected into the system model to realistically estimate the system accuracy. The details of the model and evaluation process are discussed in the next section.

4.3 System-Level Evaluation

4.3.1 Setup BSNN Traning Model

To evaluate the performance of the BSNN at the system level, we use the training method proposed in subsection 4.2. The network structures and major parameters are shown in Table 8. The training and inference models are written in the Python language using the PyTorch library. We use the MNIST and CIFAR-10 [114] datasets for training and evaluation. The networks are trained for 300 (50) epochs with a batch size of 128 (100) for the CIFAR-10 (MNIST) dataset.

Table 8. The network structures for two different datasets

Datasets	Architecture
MNIST (2 Conv layers)	32 Conv (rate encoding) – AvrPool2-32 Conv -AvrPool2-128FC1-512FC2-10FC3
CIFAR-10 (7 Conv layers)	32 Conv (rate encoding)- 32 Conv- 32 Conv-32 Conv- 32 Conv- 32 Conv - 256 Conv -AvrPool4 -512FC1-10FC2
CIFAR-10(7 Conv layers with SEW[a])	Input layer $l = g$ (Output layer l-1, Output layer l-2) l=3-7

[a]SEW: the networks with residual connections [103]

The base learning rate is set to 0.3, and the stochastic gradient descent (SGD) optimizer has a momentum of 0.9. The learning rate is scheduled with a decay factor of 10 at 50%, 70%, and 90% of the total epochs. For the CIFAR-10 dataset, we adopt BSNNs using 7 Conv layers for both networks (with and without the residual connections). Since the Conv layers occupy most of the computational workload and latency (more than 90%) [115, 116] in the deep neural network, only the hidden Conv layers are binarized in the BSNN to balance the accuracy with that of the conventional SNN [89]. Therefore, in our work, the IMC XNOR-based STT subarrays substitute for all hidden Conv layers in the BSNN evaluation.

4.3.2 Mapping the BSNN to the Circuit Model

Fig. 25 shows the mapping of a BSNN Conv layer to the STT subarray. The main calculation workload to shift from presynaptic spikes to postsynaptic spikes is done by the IMC and IF circuits described in subsection 4.2. As seen in the figure, we unroll each sliding window calculation for the given input into a vector of presynaptic spikes. Then, the unrolled vector is passed to an $N \times M$ subarray through M BL decoders. The kernel is mapped and stored in the M bitcells to perform convolution between unrolled spikes and the kernel weights. If N output channels are generated simultaneously, the $N \times M$ subarray performs one sliding window, as illustrated in Fig. 25. Furthermore, intra-layer parallelism can be

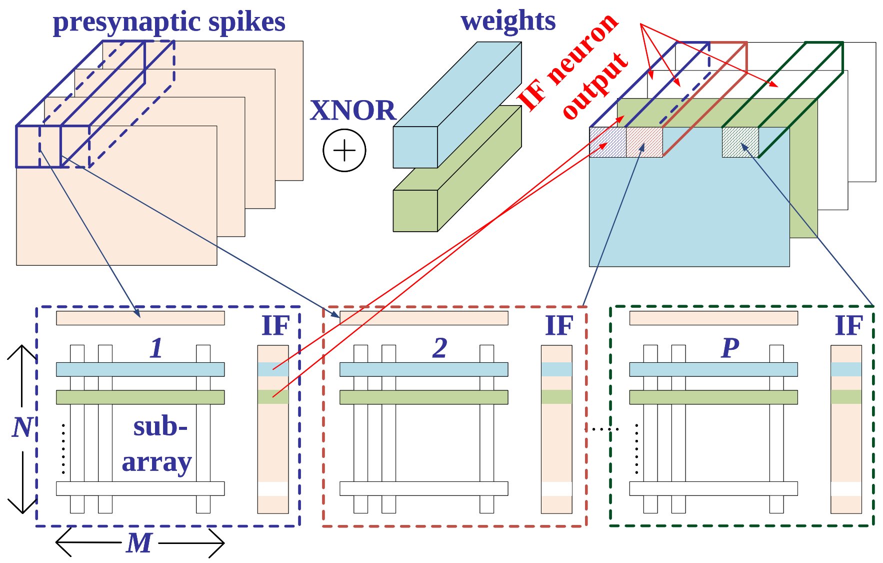


Figure 25. Mapping BSNN Conv layers to STT-MRAM subarrays.

realized by utilizing P subarrays that calculate P parallel sliding windows [117]. P hence indicates the level of parallelism, which reflects the tradeoff between the hardware cost and the speed of computation. For the practical implementation in this work, each Conv layer has N =32 input and output channels, and a kernel size of 3×3 corresponds to a subarray size of 32×288 (M=288).

4.3.3 Evaluation Classification Accuracy

We investigate the impact of time steps on the classification accuracy for both the MNIST and CIFAR-10 datasets. The classification accuracies of the BSNN inferences with six network configurations are shown in Table 9.

From the table, using more time steps essentially improves accuracy. Specifically, for networks without SEW, increasing the number of time steps from 4 to 8 results in an accuracy increase of 0.64% (3.24%) for MNIST (CIFAR-10). Additionally, it is clear that the networks with SEW achieve 2.52% (1.82%) better accuracy with 4 (8) time steps for CIFAR-10 than the conventional networks [106]. These results confirm that the training method using a surrogate gradient with SEW significantly reduces the required number of time steps compared with that of the ANN-SNN conversion method [107]. Furthermore, the nonideal

Table 9. Classification accuracies of BSNN model on the MNIST, CIFAR-10 datasets for four and eight time steps

Datasets	Architecture	Accuracy %	Time steps
MNIST	2 Conv layers	97.50	4
MNIST	2 Conv layers	98.14	8
CIFAR-10	7 Conv layers	79.70	4
CIFAR-10	7 Conv layers	82.94	8
CIFAR-10	7 Conv layers +SEW	82.22	4
CIFAR-10	7 Conv layers +SEW	84.76	8

charge increments in (37) and (38) are injected into the Python BSNN inference model to evaluate the effect of process variation on the classification accuracy. This is completed by replacing the linear models of the membrane potential and threshold with the actual models with an incorporated nonlinearity bias and a Gaussian random quantity , which are characterized by the Monte Carlo simulations in subsection 4.2. The models are evaluated on the test set multiple times (a 100-variation netlist) with different variation seeds. The means (μ_{BSNN}) and standard deviations (σ_{BSNN}) are reported in Table 10.

Table 10. The effect of process variation on the classification accuracy of the BSNN model for 8 time steps

Datasets	Architecture	Accuracy % No variation	Accuracy % With variation μ_{BSNN} σ_{BSNN}
MNIST	2 Conv layers	98.14	97.92/0.23%
CIFAR-10	7 Conv layers +SEW	84.76	83.85/0.03%

From the table, the mean classification accuracy is slightly reduced by 0.22% (0.91%) for MNIST (CIFAR-10) with 8 time steps compared to the reported accuracies for the models without variation. Additionally, the accuracies evaluated with different configurations are not much different, with standard deviations of 0.23% (MNIST) and 0.03% (CIFAR-10). In the extreme case at the

$6\times\sigma_{BSNN}$ point, the classification accuracy is degraded by 1.38% (MNIST) and 0.18% (CIFAR-10). Overall, these results permit us to conclude that process variation has a minor impact on classification accuracy and the proposed BSNN exhibits a very good level of fault tolerance.

4.3.4 Energy, Throughput, and Area of the Subarray

In the proposed STT-BSNN architecture, the energy per spiking operation is provided by the energy for synapses and for the IF neuron circuit:

$$E_{spike} = E_{synapses} + E_{neuron} = E_{wl} + E_{bitcell} + E_{neuron} \tag{40}$$

where $E_{synapses}$ is the energy consumed by the synapse operations of the IMC circuit (i.e., the MAC operations). This includes the precharged energy required for the high-load word lines E_{wl} and the energy consumed by the M bitcells $E_{bitcell}$. The latter essentially accounts for the main portion of the total energy, which is proportional to the sampling time and the accumulated current drawn from the BL source voltage V_{BL}. E_{neuron} is the energy spent on the IF neuron circuit, which includes the energy needed for the CB, ACC1, ACC2 and FS subcircuits in Fig. 20. Since these circuits are all charge-based circuits, they consume energy only during switching, i.e., without any direct currents. Therefore, their energy proportions are small compared to $E_{synapses}$, which normally accounts for the main contribution in E_{spike}. Specifically, for an IMC array row of size 288, the $E_{synapses}$ for 288 synaptic elements is found to be 1.58 pJ (where $E_{wl} = 0.064\ pJ$ and $E_{bitcell} = 1.52\ pJ$), where the optimal sampling time is set to 1 ns when using the precharged technique for the booster [31].E_{neuron} accounts for only 0.052 pJ, which results in a total spiking operation energy of E_{spike}=1.63 pJ. In this work, we define the number of operations to be equal to the size of the MAC function. This means that 288 operations are executed within one time step. Therefore, the energy efficiency E_{eff} $(TOPS/W)$ for an IMC array row of size 288 is estimated to be 288/1.63=176.6 $TOPS/W$. The rough estimation of the subarray area is equal to 608 μm^2 for a single-row implementation with 288 bitcells. The estimation area for a neuron is equal to 115 μm^2 For a subarray of size 32×288, the number of operations is equal to 32×288=9216 (operations) over 8 time steps with a period of T_{spike}=6 ns. The throughput efficiency $T_{subarray}$ is equal to

Table 11. Comparison with previous works

	IJCNN'19 [89]	TNNSL'20 [93]	VLSI'20 [95]	Our work
synapse	MTJ	MTJ	MTJ	MTJ
neuron	MTJ	Digital	MTJ	Analog
technology	45 nm	28 nm	N/A	65 nm
network type	BSNN	SNN	BSNN	BSNN
structure	3 Conv	FC layers	2 Conv	2/7 Conv
neuron	sigmoid	IF	Poisson	IF
weight	+1/-1	+1/0[c]	+1/-1	+1/0
energy/operation(fJ)	36[b]	8.87	N/A	5.48
area/neuron(F^2)[a]	N/A	$\approx 15 \times 10^5$	6×10^3	32×10^3
acuuracy MNIST/ CIFAR-10	N/A 70.3 %	91.5 % N/A	$\approx$ 97.4 % N/A	97.92 % 83.85 %

[a]The area is normalized to F^2, with F is the technology feature size.
[b]The maximum energy consumption per spiking event for a synapse, as reported in [118, 119].
[c]The full-precision weights are converted into stochastic bits (+1/0) in each time step.

(9216 /8$\times$ 6)=192 $GOPS$. If P sliding windows are processed simultaneously, the throughput efficiency increases by P times (P$\times T_{subarray}$).

4.3.5 Comparision with Related Works

Table 11 summarizes a comparison with previous work on IMC architectures for SNNs. Although there are many similar works [89, 93, 94, 95] on this topic, we have selected the most relevant studies using spintronic memory and spiking networks for comparison. Regarding accuracy, using a deeper network and direct training method, the accuracy level of BSNN in this work is $\approx$13.8% higher than that in [89], evaluated using CIFAR-10 dataset. The implementation in [93] with spintronic synapse and digital neuron also achieves lower accuracy (by 6.3% for MNIST dataset) compared to our work. This is partly because their

Table 12. Comparison with previous works with different technologies

	TCAS'20 [120]	L-SSC'21 [121]	TrueNorth [122]	Our work
synapse	SRAM	SRAM	Digital	MTJ
neuron	Digital	Analog	Digital	Analog
technology	90 nm	65 nm	28nm	65 nm
network type	BSNN	SNN	T-SNN	BSNN
energy eff.($TOPS/W$)	89.49	0.99	N/A	176.6
energy/operation(fJ)	184	N/A	2700	5.48
Frequency(MHz)	37.5	500	N/A	166
acuuracy MNIST/ CIFAR-10	92.30 % N/A	98.96 % N/A	97.6 % 89.32 %	97.92 % 83.85 %

results are reported for a fully connected network. An all-spin SNN in [95] exhibits similar accuracy for MNIST dataset, though their spiking rate is much lower than ours. The reason is that their method requires a large sampling time to convert synaptic current into a spike duty cycle in each time step. Regarding the energy, the energy consumption per synapse calculated for our BSNN is ∼6.5 times lower than that of the MTJ synapse reported in [89], not considering their synapse is implemented in a smaller technology node (45nm). Similarly, our work is still 1.6× more energy-efficient than the reported synapse energy in [93], even though their work is implemented on a 28nm CMOS, theoretically ≈2 times more energy-efficient than the same 65nm implementation. Finally, for area comparison reported in F^2 (F is the technology feature size), the area per neuron of our model is essentially much better than digital implementation [93]. Still, it is not as good as all spintronic one in [95]. That advantage comes with the clear trade-offs in energy and latency, as mentioned earlier. Table 12 shows a comparison table of recently presented spiking neural network accelerators with different technologies. In [120], a stochastic bit enabled BSNN is presented to mitigate the hardware complexity for rate-encoding for on-chip learning. However, the dominated computation in [120] is implemented with digital circuits, leading to high power consumption compared to our work. The authors in [121] implemented a reconfigurable IMC macro, supporting all instructions for SNN inference. The area is roughly estimated in this work considering the necessity of this performance metric since such design in this emerging technology is not fully available for taping out at the current state. However, the IMC macro in [121] suffers the limitation of size. In detail, the array size equal (restricted to the input channels of Conv layers is 14 with 3x3 kernel size) might not be adopted for deep spiking structures. Furthermore, this work's peak energy efficiency achieves up to 0.99 TOPS/W, which is still lower than our IMC macro due to the representation of the full-precision weighs. In comparison to [122], our work has advantages in terms of energy efficiency but lower classification accuracy.

5 Conclusion

We have shown the hardware implementation of the RBF using stochastic computing based on the definition of a limit of an exponent. It is shown that this method is experimentally superior to competing classical approaches in term of accuracy. Future work will be towards stochastic computing for the RBF kernel for full neural networks. The results show that our work outperforms the two-dimensional finite-state machine-based implementation, roughly 54% less hardware cost. Regarding the critical path delay, our design is less than 12% than others on average. Besides, the power consumtion is 70% lower than two-dimensional finite-state machine-based implementation.

In the 2-nd part of the thesis, an approached computation of sigmoid($2ax$) and tanh(ax) in a bipolar format based Bernstein polynomial has been proposed. The results showed that 90% improvement of accuracy had been achieved while maintaining a comparable hardware cost compared to the state-of-the-art.

The 3-rd part of the thesis presents an in-memory BSNN based on STT-MRAM for low-power, low-latency on-edge AI applications. We propose a direct BSNN training approach using a surrogate gradient with residual connections that achieves high classification accuracy with much fewer time steps than the number of steps required in prior works. Furthermore, we propose a full-circuit solution for IMC MAC operations based on an STT-MRAM array, which allows ultrafast vector multiplication to be performed within one memory access phase. Furthermore, we propose a dynamic threshold approach for the IF circuit that mimics the neuron spiking behavior and consumes very low power. The BSNN system model is then re-evaluated using realistic circuit parameters and exact circuit simulations. The results indicate that device mismatches and nonlinearity essentially affect the misclassification accuracy of the model. Nonetheless, the accuracy degradation is insignificant, and the proposed BSNN still offers decent performance in comparison with other methods. The results show that the proposed design can achieve a performance of 176.6 TOPS/W for the IMC subarray of 1×288. The classification accuracy is 97.92% (83.85%) for the MNIST (CIFAR-10) dataset. The impacts of the device nonidealities and process variations are also thoroughly covered in the analysis. The proposed design approach with a practical circuit solution could potentially pave the way for ultralow-power DNNs to be applied in on-edge AI applications. We are working forward to improve the architecture and design to fit deeper and larger networks.

Acknowledgements

First and foremost, I am extremely grateful to my supervisor, Prof. Yasuhiko Nakashima, for his invaluable advice, continuous support, and patience during my Ph.D. study. His immense knowledge and great experience have encouraged me in my academic research and daily life. I would also like to thank Assoc. Prof. Renyuan ZHANG and Dr. Kien Trinh for their technical support of my study. Besides, I would like to thank Prof. Yuichi Hayashi for serving on my thesis committee and giving valuable evaluations.
Next, I would like to thank the Computing Architecture Laboratory, Division of Information Science, Nara Institute of Science and Technology (NAIST) for supporting me in my career. I also would like to thank the Japanese Ministry of Education, Culture, Sport, Science, and Technology (MEXT) for giving me the scholarship to study in Japan.
Finally, I would like to thank my mother for sacrificing so much in her life to help me live mine.

References

[1] Bhattarabhorn Wattanacheep and Orachat Chitsobhuk. Camera pose estimation using cnn. In *2020 the 3rd International Conference on Control and Computer Vision*, page 84–88, 2020.

[2] M. P. Bhuyan, S. K. Sarma, and M. Rahman. Natural language processing based stochastic model for the correctness of assamese sentences. In *2020 5th International Conference on Communication and Electronics Systems (ICCES)*, pages 1179–1182, 2020.

[3] Yong Xu, Jun Du, Li-Rong Dai, and Chin-Hui Lee. A regression approach to speech enhancement based on deep neural networks. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 23(1):7–19, 2015.

[4] Yiran Chen, Yuan Xie, Linghao Song, Fan Chen, and Tianqi Tang. A survey of accelerator architectures for deep neural networks. *Engineering*, 6(3):264–274, 2020.

[5] Sun-Ting Tsai, En-Jui Kuo, and Pratyush Tiwary. Learning molecular dynamics with simple language model built upon long short-term memory neural network. *Nature communications*, 11(1):1–11, 2020.

[6] Jiecao Yu, Andrew Lukefahr, David Palframan, Ganesh Dasika, Reetuparna Das, and Scott Mahlke. Scalpel: Customizing dnn pruning to the underlying hardware parallelism. *ACM SIGARCH Computer Architecture News*, 45(2):548–560, 2017.

[7] Li Zhou, Mohammad Hossein Samavatian, Anys Bacha, Saikat Majumdar, and Radu Teodorescu. Adaptive parallel execution of deep neural networks on heterogeneous edge devices. In *Proceedings of the 4th ACM/IEEE Symposium on Edge Computing*, pages 195–208, 2019.

[8] Jiasi Chen and Xukan Ran. Deep learning with edge computing: A review. *Proceedings of the IEEE*, 107(8):1655–1674, 2019.

[9] Sourya Dey, Diandian Chen, Zongyang Li, Souvik Kundu, Kuan-Wen Huang, Keith M. Chugg, and Peter A. Beerel. A highly parallel fpga implementation of sparse neural network training. In *2018 International Conference on ReConFigurable Computing and FPGAs (ReConFig)*, pages 1–4, 2018.

[10] Sourya Dey, Kuan-Wen Huang, Peter A. Beerel, and Keith M. Chugg. Pre-defined sparse neural networks with hardware acceleration. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 9(2):332–345, 2019.

[11] Jeremy Kepner, Simon Alford, Vijay Gadepally, Michael Jones, Lauren Milechin, Ryan Robinett, and Sid Samsi. Sparse deep neural network graph challenge. In *2019 IEEE High Performance Extreme Computing Conference (HPEC)*, pages 1–7, 2019.

[12] Ali Keshavarzi, Kai Ni, Wilbert Van Den Hoek, Suman Datta, and Arijit Raychowdhury. Ferroelectronics for edge intelligence. *IEEE Micro*, 40(6):33–48, 2020.

[13] Erwei Wang, James J Davis, Ruizhe Zhao, Ho-Cheung Ng, Xinyu Niu, Wayne Luk, Peter YK Cheung, and George A Constantinides. Deep neural network approximation for custom hardware: Where we've been, where we're going. *ACM Computing Surveys (CSUR)*, 52(2):1–39, 2019.

[14] Armin Alaghi and John P Hayes. Survey of stochastic computing. *ACM Transactions on Embedded computing systems (TECS)*, 12(2s):1–19, 2013.

[15] Kerstin Beer, Dmytro Bondarenko, Terry Farrelly, Tobias J Osborne, Robert Salzmann, Daniel Scheiermann, and Ramona Wolf. Training deep quantum neural networks. *Nature communications*, 11(1):1–6, 2020.

[16] Saurabh Jain, Longyang Lin, and Massimo Alioto. Broad-purpose in-memory computing for signal monitoring and machine learning workloads. *IEEE Solid-State Circuits Letters*, 3:394–397, 2020.

[17] Ankit Mondal and Ankur Srivastava. Energy-efficient design of mtj-based neural networks with stochastic computing. *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, 16(1):1–27, 2019.

[18] Ankit Mondal and Ankur Srivastava. Energy-efficient design of mtj-based neural networks with stochastic computing. *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, 16(1):1–27, 2019.

[19] G. Maor, X. Zeng, Z. Wang, and Y. Hu. An fpga implementation of stochastic computing-based lstm. In *2019 IEEE 37th International Conference on Computer Design (ICCD)*, pages 38–46, 2019.

[20] H. Xiong, M. Abu bakar, and G. He. Hardware implementation of an improved stochastic computing based deep neural network using short sequence length. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 67(11):2667–2671, 2020.

[21] A. Ardakani, F. Leduc-Primeau, N. Onizawa, T. Hanyu, and W. J. Gross. Vlsi implementation of deep neural network using integral stochastic computing. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 25(10):2688–2699, 2017.

[22] N. Onizawa, W. J. Gross, and T. Hanyu. Stochastic-computing based brainwave lsi towards an intelligence edge. In *2019 26th IEEE International Conference on Electronics, Circuits and Systems (ICECS)*, pages 434–437, 2019.

[23] B. Li, M. H. Najafi, and D. J. Lilja. An fpga implementation of a restricted boltzmann machine classifier using stochastic bit streams. In *2015 IEEE 26th International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, pages 68–69, 2015.

[24] J. Yu, K. Kim, J. Lee, and K. Choi. Accurate and efficient stochastic computing hardware for convolutional neural networks. In *2017 IEEE International Conference on Computer Design (ICCD)*, pages 105–112, 2017.

[25] Z. Li, J. Li, A. Ren, R. Cai, C. Ding, X. Qian, J. Draper, B. Yuan, J. Tang, Q. Qiu, and Y. Wang. Heif: Highly efficient stochastic computing-based inference framework for deep neural networks. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 38(8):1543–1556, 2019.

[26] T. Hirtzlin, B. Penkovsky, M. Bocquet, J. Klein, J. Portal, and D. Querlioz. Stochastic computing for hardware implementation of binarized neural networks. *IEEE Access*, 7:76394–76403, 2019.

[27] Brian R Gaines. Stochastic computing. In *Proceedings of the April 18-20, 1967, spring joint computer conference*, pages 149–156, 1967.

[28] Renyuan Zhang, Tati Erlina, Tinh Van Nguyen, and Yasuhiko Nakashima. Hybrid stochastic computing circuits in continuous statistics domain. In *2020 IEEE 33rd International System-on-Chip Conference (SOCC)*, pages 225–230. IEEE, 2020.

[29] Keshab K Parhi. Analysis of stochastic logic circuits in unipolar, bipolar and hybrid formats. In *2017 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1–4. IEEE, 2017.

[30] M Hassan Najafi, David J Lilja, and Marc Riedel. Deterministic methods for stochastic computing using low-discrepancy sequences. In *2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 1–8. IEEE, 2018.

[31] Weikang Qian, Xin Li, Marc D Riedel, Kia Bazargan, and David J Lilja. An architecture for fault-tolerant computation with stochastic logic. *IEEE transactions on computers*, 60(1):93–105, 2010.

[32] Sina Asadi, M Hassan Najafi, and Mohsen Imani. A low-cost fsm-based bit-stream generator for low-discrepancy stochastic computing. In *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 908–913. IEEE, 2021.

[33] Amir Ardakani, Arash Ardakani, and Warren J Gross. Fault-tolerance of binarized and stochastic computing-based neural networks. In *2021 IEEE Workshop on Signal Processing Systems (SiPS)*, pages 52–57. IEEE, 2021.

[34] B.D. Brown and H.C. Card. Stochastic neural computation. i. computational elements. *IEEE Transactions on Computers*, 50(9):891–905, 2001.

[35] Tati Erlina, Renyuan Zhang, and Yasuhiko Nakashima. A feasibility study of multi-domain stochastic computing circuit. *IEICE Transactions on Electronics*, 2020.

[36] Van-Tinh Nguyen, Tieu-Khanh Luong, Han Le Duc, and Van-Phuc Hoang. An efficient hardware implementation of activation functions using stochastic computing for deep neural networks. In *2018 IEEE 12th International Symposium on Embedded Multicore/Many-core Systems-on-Chip (MCSoC)*, pages 233–236, 2018.

[37] Keshab K. Parhi and Yin Liu. Computing arithmetic functions using stochastic logic by series expansion. *IEEE Transactions on Emerging Topics in Computing*, 7(1):44–59, 2019.

[38] Tieu-Khanh Luong, Van-Tinh Nguyen, Anh-Thai Nguyen, and Emanuel Popovici. Efficient architectures and implementation of arithmetic functions approximation based stochastic computing. In *2019 IEEE 30th International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, volume 2160, pages 281–287. IEEE, 2019.

[39] Tati Erlina, Renyuan Zhang, Yasuhiko Nakashima, et al. A programmable approximate calculation unit employing time-encoded stochastic computing elements. In *2019 Seventh International Symposium on Computing and Networking Workshops (CANDARW)*, pages 91–96. IEEE, 2019.

[40] Peng Li, David J Lilja, Weikang Qian, Kia Bazargan, and Marc Riedel. The synthesis of complex arithmetic computation on stochastic bit streams using sequential logic. In *Proceedings of the International Conference on Computer-Aided Design*, pages 480–487, 2012.

[41] Yin Liu and Keshab K Parhi. Computing hyperbolic tangent and sigmoid functions using stochastic logic. In *2016 50th Asilomar Conference on Signals, Systems and Computers*, pages 1580–1585. IEEE, 2016.

[42] Yu-Hsin Chen, Tushar Krishna, Joel S Emer, and Vivienne Sze. Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks. *IEEE journal of solid-state circuits*, 52(1):127–138, 2016.

[43] Bert Moons, Roel Uytterhoeven, Wim Dehaene, and Marian Verhelst. 14.5 envision: A 0.26-to-10tops/w subword-parallel dynamic-voltage-accuracy-frequency-scalable convolutional neural network processor in 28nm fdsoi. In *2017 IEEE International Solid-State Circuits Conference (ISSCC)*, pages 246–247. IEEE, 2017.

[44] Dongjoo Shin, Jinmook Lee, Jinsu Lee, and Hoi-Jun Yoo. 14.2 dnpu: An 8.1 tops/w reconfigurable cnn-rnn processor for general-purpose deep neural networks. In *2017 IEEE International Solid-State Circuits Conference (ISSCC)*, pages 240–241. IEEE, 2017.

[45] P Narayanan, S Ambrogio, A Okazaki, K Hosokawa, H Tsai, A Nomura, T Yasuda, C Mackin, SC Lewis, A Friz, et al. Fully on-chip mac at 14nm enabled by accurate row-wise programming of pcm-based weights and parallel vector-transport in duration-format. In *2021 Symposium on VLSI Technology*, pages 1–2. IEEE, 2021.

[46] Avilash Mukherjee, Kumar Saurav, Prashant Nair, Sudip Shekhar, and Mieszko Lis. A case for emerging memories in dnn accelerators. In *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 938–941. IEEE, 2021.

[47] Saurabh Jain, Longyang Lin, and Massimo Alioto. Broad-purpose in-memory computing for signal monitoring and machine learning workloads. *IEEE Solid-State Circuits Letters*, 3:394–397, 2020.

[48] Bin Zhang, Weilin Chen, Jianmin Zeng, Fei Fan, Junwei Gu, Xinhui Chen, Lin Yan, Guangjun Xie, Shuzhi Liu, Qing Yan, et al. 90% yield production

of polymer nano-memristor for in-memory computing. *Nature communications*, 12(1):1–11, 2021.

[49] Yin Wang, Hongwei Tang, Yufeng Xie, Xinyu Chen, Shunli Ma, Zhengzong Sun, Qingqing Sun, Lin Chen, Hao Zhu, Jing Wan, et al. An in-memory computing architecture based on two-dimensional semiconductors for multiply-accumulate operations. *Nature communications*, 12(1):1–8, 2021.

[50] Fei Xue, Xin He, Zhenyu Wang, José Ramón Durán Retamal, Zheng Chai, Lingling Jing, Chenhui Zhang, Hui Fang, Yang Chai, Tao Jiang, et al. Giant ferroelectric resistance switching controlled by a modulatory terminal for low-power neuromorphic in-memory computing. *Advanced Materials*, 33(21):2008709, 2021.

[51] Hongyang Jia, Murat Ozatay, Yinqi Tang, Hossein Valavi, Rakshit Pathak, Jinseok Lee, and Naveen Verma. 15.1 a programmable neural-network inference accelerator based on scalable in-memory computing. In *2021 IEEE International Solid-State Circuits Conference (ISSCC)*, volume 64, pages 236–238. IEEE, 2021.

[52] Nikolaos Vasileiadis, Vasileios Ntinas, Iosif-Angelos Fyrigos, Rafailia-Eleni Karamani, Vassilios Ioannou-Sougleridis, Pascal Normand, Ioannis Karafyllidis, Georgios Ch Sirakoulis, and Panagiotis Dimitrakis. A new 1p1r image sensor with in-memory computing properties based on silicon nitride devices. In *2021 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1–5. IEEE, 2021.

[53] Martino Dazzi, Abu Sebastian, Thomas Parnell, Pier Andrea Francese, Luca Benini, and Evangelos Eleftheriou. Efficient pipelined execution of cnns based on in-memory computing and graph homomorphism verification. *IEEE Transactions on Computers*, 70(6):922–935, 2021.

[54] Mohsen Riahi Alam, M Hassan Najafi, and Nima Taheri Nejad. Exact stochastic computing multiplication in memristive memory. *IEEE Design & Test*, 2021.

[55] Corey Lammie, Jason K Eshraghian, Wei D Lu, and Mostafa Rahimi Azghadi. Memristive stochastic computing for deep learning parameter optimization. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 68(5):1650–1654, 2021.

[56] Ruiqi Guo, Zhiheng Yue, Xin Si, Te Hu, Hao Li, Limei Tang, Yabing Wang, Leibo Liu, Meng-Fan Chang, Qiang Li, et al. 15.4 a 5.99-to-691.1 tops/w tensor-train in-memory-computing processor using bit-level-sparsity-based optimization and variable-precision quantization. In *2021 IEEE International Solid-State Circuits Conference (ISSCC)*, volume 64, pages 242–244. IEEE, 2021.

[57] Guodong Yin, Yi Cai, Juejian Wu, Zhengyang Duan, Zhenhua Zhu, Yongpan Liu, Yu Wang, Huazhong Yang, and Xueqing Li. Enabling lower-power charge-domain nonvolatile in-memory computing with ferroelectric fets. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 2021.

[58] Jiahao Song, Yuan Wang, Minguang Guo, Xiang Ji, Kaili Cheng, Yixuan Hu, Xiyuan Tang, Runsheng Wang, and Ru Huang. Td-sram: Time-domain-based in-memory computing macro for binary neural networks. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 2021.

[59] Jian Meng, Li Yang, Xiaochen Peng, Shimeng Yu, Deliang Fan, and Jae-Sun Seo. Structured pruning of rram crossbars for efficient in-memory computing acceleration of deep neural networks. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 68(5):1576–1580, 2021.

[60] Gihun Choe, Wonbo Shim, Panni Wang, Jae Hur, Asif Islam Khan, and Shimeng Yu. Impact of random phase distribution in ferroelectric transistors-based 3-d nand architecture on in-memory computing. *IEEE Transactions on Electron Devices*, 68(5):2543–2548, 2021.

[61] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Binarized neural networks. *Advances in neural information processing systems*, 29, 2016.

[62] Matthieu Courbariaux, Itay Hubara, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Binarized neural networks: Training deep neural networks with weights and activations constrained to+ 1 or-1. *arXiv preprint arXiv:1602.02830*, 2016.

[63] Eriko Nurvitadhi, David Sheffield, Jaewoong Sim, Asit Mishra, Ganesh Venkatesh, and Debbie Marr. Accelerating binarized neural networks: Comparison of fpga, cpu, gpu, and asic. In *2016 International Conference on Field-Programmable Technology (FPT)*, pages 77–84. IEEE, 2016.

[64] Nicholas J Fraser, Yaman Umuroglu, Giulio Gambardella, Michaela Blott, Philip Leong, Magnus Jahre, and Kees Vissers. Scaling binarized neural networks on reconfigurable logic. In *Proceedings of the 8th Workshop and 6th Workshop on Parallel Programming and Run-Time Management Techniques for Many-core Architectures and Design Tools and Architectures for Multicore Embedded Computing Platforms*, pages 25–30, 2017.

[65] Yaman Umuroglu, Nicholas J Fraser, Giulio Gambardella, Michaela Blott, Philip Leong, Magnus Jahre, and Kees Vissers. Finn: A framework for fast, scalable binarized neural network inference. In *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pages 65–74, 2017.

[66] Mohammad Ghasemzadeh, Mohammad Samragh, and Farinaz Koushanfar. Rebnet: Residual binarized neural network. In *2018 IEEE 26th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pages 57–64. IEEE, 2018.

[67] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. Xnor-net: Imagenet classification using binary convolutional neural networks. In *European conference on computer vision*, pages 525–542. Springer, 2016.

[68] Rui Liu, Xiaochen Peng, Xiaoyu Sun, Win-San Khwa, Xin Si, Jia-Jing Chen, Jia-Fang Li, Meng-Fan Chang, and Shimeng Yu. Parallelizing sram arrays with customized bit-cell for binary neural networks. In *2018*

55th ACM/ESDA/IEEE Design Automation Conference (DAC), pages 1–6. IEEE, 2018.

[69] Hyungjun Kim, Hyunmyung Oh, and Jae-Joon Kim. Energy-efficient xnor-free in-memory bnn accelerator with input distribution regularization. In *Proceedings of the 39th International Conference on Computer-Aided Design*, pages 1–9, 2020.

[70] Xin Si, Yung-Ning Tu, Wei-Hsing Huang, Jian-Wei Su, Pei-Jung Lu, Jing-Hong Wang, Ta-Wei Liu, Ssu-Yen Wu, Ruhui Liu, Yen-Chi Chou, et al. 15.5 a 28nm 64kb 6t sram computing-in-memory macro with 8b mac operation for ai edge chips. In *2020 IEEE International Solid-State Circuits Conference-(ISSCC)*, pages 246–248. IEEE, 2020.

[71] Chengshuo Yu, Taegeun Yoo, Tony Tae-Hyoung Kim, Kevin Chai Tshun Chuan, and Bongjin Kim. A 16k current-based 8t sram compute-in-memory macro with decoupled read/write and 1-5bit column adc. In *2020 IEEE Custom Integrated Circuits Conference (CICC)*, pages 1–4. IEEE, 2020.

[72] Amogh Agrawal, Adarsh Kosta, Sangamesh Kodge, Dong Eun Kim, and Kaushik Roy. Cash-ram: Enabling in-memory computations for edge inference using charge accumulation and sharing in standard 8t-sram arrays. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 10(3):295–305, 2020.

[73] Xiaoyu Sun, Shihui Yin, Xiaochen Peng, Rui Liu, Jae-sun Seo, and Shimeng Yu. Xnor-rram: A scalable and parallel resistive synaptic architecture for binary neural networks. In *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1423–1428. IEEE, 2018.

[74] Xiaoyu Sun, Xiaochen Peng, Pai-Yu Chen, Rui Liu, Jae-sun Seo, and Shimeng Yu. Fully parallel rram synaptic array for implementing binary neural network with (+ 1,- 1) weights and (+ 1, 0) neurons. In *2018 23rd Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 574–579. IEEE, 2018.

[75] Nuo Xu, Yang Lu, Weiyi Qi, Zhengping Jiang, Xiaochen Peng, Fan Chen, Jing Wang, Woosung Choi, Shimeng Yu, and Dae Sin Kim. Stt-mram design technology co-optimization for hardware neural networks. In *2018 IEEE International Electron Devices Meeting (IEDM)*, pages 15–3. IEEE, 2018.

[76] Yuhan Shi, Sangheon Oh, Zhisheng Huang, Xiao Lu, Seung H Kang, and Duygu Kuzum. Performance prospects of deeply scaled spin-transfer torque magnetic random-access memory for in-memory computing. *IEEE Electron Device Letters*, 41(7):1126–1129, 2020.

[77] Simon Van Beek, Siddharth Rao, Shreya Kundu, Woojin Kim, Barry J O'Sullivan, Stefan Cosemans, Farukh Yasin, Robert Carpenter, Sebastien Couet, Shamin H Sharifi, et al. Edge-induced reliability & performance degradation in stt-mram: an etch engineering solution. In *2021 IEEE International Reliability Physics Symposium (IRPS)*, pages 1–5. IEEE, 2021.

[78] Vivek Parmar, Manan Suri, Kazutaka Yamane, Taeyoung Lee, Nyuk Leong Chung, and Vinayak Bharat Naik. Mram-based ber resilient quantized edge-ai networks for harsh industrial conditions. In *2021 IEEE 3rd International Conference on Artificial Intelligence Circuits and Systems (AICAS)*, pages 1–4. IEEE, 2021.

[79] Hao Cai, Juntong Chen, Yongliang Zhou, and Weisheng Zhao. Towards energy-efficient stt-mram design with multi-modes reconfiguration. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 2021.

[80] Shubham Jain, Ashish Ranjan, Kaushik Roy, and Anand Raghunathan. Computing in memory with spin-transfer torque magnetic ram. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 26(3):470–483, 2017.

[81] Liang Chang, Xin Ma, Zhaohao Wang, Youguang Zhang, Yuan Xie, and Weisheng Zhao. Pxnor-bnn: In/with spin-orbit torque mram preset-xnor operation-based binary neural networks. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 27(11):2668–2679, 2019.

[82] Chih-Cheng Chang, Ming-Hung Wu, Jia-Wei Lin, Chun-Hsien Li, Vivek Parmar, Heng-Yuan Lee, Jeng-Hua Wei, Shyh-Shyuan Sheu, Manan Suri, Tian-Sheuan Chang, et al. Nv-bnn: An accurate deep convolutional neural network based on binary stt-mram for adaptive ai edge. In *2019 56th ACM/IEEE Design Automation Conference (DAC)*, pages 1–6. IEEE, 2019.

[83] Salonik Resch, S Karen Khatamifard, Zamshed Iqbal Chowdhury, Masoud Zabihi, Zhengyang Zhao, Jian-Ping Wang, Sachin S Sapatnekar, and Ulya R Karpuzcu. Pimball: Binary neural networks in spintronic memory. *ACM Transactions on Architecture and Code Optimization (TACO)*, 16(4):1–26, 2019.

[84] Shifan Gao, Bing Chen, Yiming Qu, and Yi Zhao. Mram acceleration core for vector matrix multiplication and xnor-binarized neural network inference. In *2020 International Symposium on VLSI Technology, Systems and Applications (VLSI-TSA)*, pages 153–154. IEEE, 2020.

[85] Thi-Nhan Pham, Quang-Kien Trinh, Ik-Joon Chang, and Massimo Alioto. Stt-mram architecture with parallel accumulator for in-memory binary neural networks. In *2021 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1–5. IEEE, 2021.

[86] Wenzhe Guo, Mohammed E Fouda, Ahmed M Eltawil, and Khaled Nabil Salama. Neural coding in spiking neural networks: A comparative study for robust neuromorphic systems. *Frontiers in Neuroscience*, 15:212, 2021.

[87] Abhronil Sengupta, Maryam Parsa, Bing Han, and Kaushik Roy. Probabilistic deep spiking neural systems enabled by magnetic tunnel junction. *IEEE Transactions on Electron Devices*, 63(7):2963–2970, 2016.

[88] Abhronil Sengupta, Gopalakrishnan Srinivasan, Deboleena Roy, and Kaushik Roy. Stochastic inference and learning enabled by magnetic tunnel junctions. In *2018 IEEE International Electron Devices Meeting (IEDM)*, pages 15–6. IEEE, 2018.

[89] Akhilesh Jaiswal, Amogh Agrawal, Indranil Chakraborty, Deboleena Roy, and Kaushik Roy. On robustness of spin-orbit-torque based stochastic sig-

moid neurons for spiking neural networks. In *2019 International Joint Conference on Neural Networks (IJCNN)*, pages 1–6. IEEE, 2019.

[90] Hsin-Pai Cheng, Wei Wen, Chunpeng Wu, Sicheng Li, Hai Helen Li, and Yiran Chen. Understanding the design of ibm neurosynaptic system and its tradeoffs: A user perspective. In *Design, Automation Test in Europe Conference Exhibition (DATE), 2017*, pages 139–144, 2017.

[91] Surya Narayanan, Ali Shafiee, and Rajeev Balasubramonian. Inxs: Bridging the throughput and energy gap for spiking neural networks. In *2017 International Joint Conference on Neural Networks (IJCNN)*, pages 2451–2459, 2017.

[92] Luis A Camuñas-Mesa, Bernabé Linares-Barranco, and Teresa Serrano-Gotarredona. Implementation of a tunable spiking neuron for stdp with memristors in fdsoi 28nm. In *2020 2nd IEEE International Conference on Artificial Intelligence Circuits and Systems (AICAS)*, pages 94–98. IEEE, 2020.

[93] Samuel N Pagliarini, Sudipta Bhuin, Mehmet Meric Isgenc, Ayan Kumar Biswas, and Larry Pileggi. A probabilistic synapse with strained mtjs for spiking neural networks. *IEEE transactions on neural networks and learning systems*, 31(4):1113–1123, 2019.

[94] Akhilesh Jaiswal, Sourjya Roy, Gopalakrishnan Srinivasan, and Kaushik Roy. Proposal for a leaky-integrate-fire spiking neuron based on magnetoelectric switching of ferromagnets. *IEEE Transactions on Electron Devices*, 64(4):1818–1824, 2017.

[95] Ming-Hung Wu, Ming-Shun Huang, Zhifeng Zhu, Fu-Xiang Liang, Ming-Chun Hong, Jiefang Deng, Jeng-Hua Wei, Shyh-Shyuan Sheu, Chih-I Wu, Gengchiau Liang, et al. Compact probabilistic poisson neuron based on back-hopping oscillation in stt-mram for all-spin deep spiking neural network. In *2020 IEEE Symposium on VLSI Technology*, pages 1–2. IEEE, 2020.

[96] Van-Tinh Nguyen, Tieu-Khanh Luong, Renyuan Zhang, and Yasuhiko Nakashima. A compact and accuracy-reconfigurable univariate rbf kernel based on stochastic logic. In *2020 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1–5. IEEE, 2020.

[97] Van-Tinh Nguyen, Tieu-Khanh Luong, Emanuel Popovici, Quang-Kien Trinh, Renyuan Zhang, and Yasuhiko Nakashima. An accurate and compact hyperbolic tangent and sigmoid computation based stochastic logic. In *2021 IEEE International Midwest Symposium on Circuits and Systems (MWSCAS)*, pages 386–390. IEEE, 2021.

[98] Van-Tinh Nguyen, Qung-Kien Tring, Renyuan Zhang, and Yasuhiko Nakashima. Xnor-bsnn: In-memory computing model for deep binarized spiking neural network. In *International Conference on High Performance Big Data and Intelligent Systems, (accepted for oral presentation)*. IEEE, 2021.

[99] Van-Tinh Nguyen, Quang-Kien Trinh, Renyuan Zhang, and Yasuhiko Nakashima. Stt-bsnn: An in-memory deep binary spiking neural network based on stt-mram. *IEEE Access*, 9:151373–151385, 2021.

[100] Youngeun Kim and Priyadarshini Panda. Revisiting batch normalization for training low-latency deep spiking neural networks from scratch. *arXiv preprint arXiv:2010.01729*, 2020.

[101] Yujie Wu, Lei Deng, Guoqi Li, Jun Zhu, Yuan Xie, and Luping Shi. Direct training for spiking neural networks: Faster, larger, better. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 1311–1318, 2019.

[102] Hanle Zheng, Yujie Wu, Lei Deng, Yifan Hu, and Guoqi Li. Going deeper with directly-trained larger spiking neural networks. *arXiv preprint arXiv:2011.05280*, 2020.

[103] Wei Fang, Zhaofei Yu, Yanqi Chen, Tiejun Huang, Timothée Masquelier, and Yonghong Tian. Deep residual learning in spiking neural networks. *arXiv preprint arXiv:2102.04159*, 2021.

[104] Muath Abu Lebdeh, Heba Abunahla, Baker Mohammad, and Mahmoud Al-Qutayri. An efficient heterogeneous memristive xnor for in-memory computing. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 64(9):2427–2437, 2017.

[105] Tifenn Hirtzlin, Bogdan Penkovsky, Marc Bocquet, Jacques-Olivier Klein, Jean-Michel Portal, and Damien Querlioz. Stochastic computing for hardware implementation of binarized neural networks. *IEEE Access*, 7:76394–76403, 2019.

[106] Hong-Han Lien, Chung-Wei Hsu, and Tian-Sheuan Chang. Vsa: Reconfigurable vectorwise spiking neural network accelerator. In *2021 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1–5. IEEE, 2021.

[107] Yixuan Wang, Yang Xu, Rui Yan, and Huajin Tang. Deep spiking neural networks with binary weights for object recognition. *IEEE Transactions on Cognitive and Developmental Systems*, 2020.

[108] Quang Kien Trinh, Sergio Ruocco, and Massimo Alioto. Voltage scaled stt-mrams towards minimum-energy write access. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 6(3):305–318, 2016.

[109] TRINH QUANG KIEN. Stt-mrams circuit techniques for enhanced robustness in low power embedded applications. 2017.

[110] Quang Kien Trinh, Sergio Ruocco, and Massimo Alioto. Novel boosted-voltage sensing scheme for variation-resilient stt-mram read. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 63(10):1652–1660, 2016.

[111] Michiel Van Elzakker, Ed van Tuijl, Paul Geraedts, Daniel Schinkel, Eric AM Klumperink, and Bram Nauta. A 10-bit charge-redistribution adc consuming 1.9 uw at 1 ms/s. *IEEE Journal of Solid-State Circuits*, 45(5):1007–1015, 2010.

[112] Takayasu Sakurai and A Richard Newton. Alpha-power law mosfet model and its applications to cmos inverter delay and other formulas. *IEEE Journal of solid-state circuits*, 25(2):584–594, 1990.

[113] E Chen, D Apalkov, Z Diao, A Driskill-Smith, D Druist, D Lottis, V Nikitin, X Tang, S Watts, S Wang, et al. Advances and future prospects of spin-transfer torque random access memory. *IEEE Transactions on Magnetics*, 46(6):1873–1878, 2010.

[114] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

[115] Yan Chen, Jing Zhang, Yuebing Xu, Yingjie Zhang, Renyuan Zhang, and Yasuhiko Nakashima. An efficient reram-based inference accelerator for convolutional neural networks via activation reuse. *IEICE Electronics Express*, pages 16–20190396, 2019.

[116] Lei Jiang, Minje Kim, Wujie Wen, and Danghui Wang. Xnor-pop: A processing-in-memory architecture for binary convolutional neural networks in wide-io2 drams. In *2017 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*, pages 1–6, 2017.

[117] Linghao Song, Xuehai Qian, Hai Li, and Yiran Chen. Pipelayer: A pipelined reram-based accelerator for deep learning. In *2017 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 541–552, 2017.

[118] Gopalakrishnan Srinivasan, Abhronil Sengupta, and Kaushik Roy. Magnetic tunnel junction enabled all-spin stochastic spiking neural network. In *Design, Automation Test in Europe Conference Exhibition (DATE), 2017*, pages 530–535, 2017.

[119] Abhronil Sengupta Srinivasan, Gopalakrishnan and Kaushik Roy. Magnetic tunnel junction based long-term short-term stochastic synapse for a spiking neural network with on-chip stdp learning. *Scientific reports*, pages 1–13, 2016.

[120] Minsuk Koo, Gopalakrishnan Srinivasan, Yong Shim, and Kaushik Roy. sbsnn: Stochastic-bits enabled binary spiking neural network with on-chip learning for energy efficient neuromorphic computing at the edge. *IEEE*

Transactions on Circuits and Systems I: Regular Papers, 67(8):2546–2555, 2020.

[121] Amogh Agrawal, Mustafa Ali, Minsuk Koo, Nitin Rathi, Akhilesh Jaiswal, and Kaushik Roy. Impulse: A 65-nm digital compute-in-memory macro with fused weights and membrane potential for spike-based sequential learning tasks. volume 4, pages 137–140, 2021.

[122] Paul A Merolla, John V Arthur, Rodrigo Alvarez-Icaza, Andrew S Cassidy, Jun Sawada, Filipp Akopyan, Bryan L Jackson, Nabil Imam, Chen Guo, Yutaka Nakamura, et al. A million spiking-neuron integrated circuit with a scalable communication network and interface. *Science*, 345(6197):668–673, 2014.

List of publications

Peer review journal papers

1. Van-Tinh Nguyen, Kien-Quang Trinh, Zhang Renyuan, Yasuhiko Nakashima; "STT-BSNN: An In-Memory Deep Binary Spiking Neural Network Based on STT-MRAM"; IEEE Access, Vol. 9, pp.151373-151385 , Nov. 2021.

Peer review conference papers

1. Van-Tinh Nguyen,Erlina Tati,Zhang Renyuan, Yasuhiko Nakashima; "A Programmable Approximate Calculation Unit Employing Time-Encoded Stochastic Computing Elements"; 2019 Seventh International Symposium on Computing and Networking Workshops (CANDARW), pp.91–96, Nov. 2019.

2. Van-Tinh Nguyen, Zhang Renyuan, Yasuhiko Nakashima; "A Compact and Accuracy-Reconfigurable Univariate RBF Kernel Based on Stochastic Logic"; 2020 IEEE International Symposium on Circuits and Systems (ISCAS), pp.1–5 , Oct. 2020.

3. Van-Tinh Nguyen, Zhang Renyuan, Yasuhiko Nakashima; "An Accurate and Compact Hyperbolic Tangent and Sigmoid Computation Based Stochastic Logic"; 2021 IEEE International Midwest Symposium on Circuits and Systems (MWSCAS), pp.386–390 , Aug. 2021.

4. Van-Tinh Nguyen,Kien-Quang Trinh, Zhang Renyuan, Yasuhiko Nakashima; "XNOR-BSNN: In-Memory Computing Model for Deep Binarized Spiking Neural Network"; 2021 International Conference on High Performance Big Data and Intelligent Systems,accepted for oral presentation, Dec.2021.

5. Zhang Renyuan,Erlina Tati, Van-Tinh Nguyen, Yasuhiko Nakashima; "Hybrid Stochastic Computing Circuits in Continuous Statistics Domain"; 2020 IEEE 33rd International System-on-Chip Conference (SOCC), pp. 225-230, Sept. 2020.