Spiking Neural Networks in Spintronic Computational RAM

HÜSREV CILASUN, SALONIK RESCH, ZAMSHED I. CHOWDHURY, ERIN OLSON, MASOUD ZABIHI, ZHENGYANG ZHAO, THOMAS PETERSON, KESHAB K. PARHI, JIAN-PING WANG, SACHIN S. SAPATNEKAR, and ULYA R. KARPUZCU, University of Minnesota, Twin Cities

Spiking Neural Networks (SNNs) represent a biologically inspired computation model capable of emulating neural computation in human brain and brain-like structures. The main promise is very low energy consumption. Classic Von Neumann architecture based SNN accelerators in hardware, however, often fall short of addressing demanding computation and data transfer requirements efficiently at scale. In this article, we propose a promising alternative to overcome scalability limitations, based on a network of in-memory SNN accelerators, which can reduce the energy consumption by up to 150.25× when compared to a representative ASIC solution. The significant reduction in energy comes from two key aspects of the hardware design to minimize data communication overheads: (1) each node represents an in-memory SNN accelerator based on a spintronic Computational RAM array, and (2) a novel, De Bruijn graph based architecture establishes the SNN array connectivity.

CCS Concepts: • **Computer systems organization → Other architectures;**

Additional Key Words and Phrases: Processing in memory, non-volatile memory, spiking neural networks, computational random access memory

ACM Reference format:
Hüsrev Cılasun, Salonik Resch, Zamshed I. Chowdhury, Erin Olson, Masoud Zabihi, Zhengyang Zhao, Thomas Peterson, Keshab K. Parhi, Jian-Ping Wang, Sachin S. Sapatnekar, and Ulya R. Karpuzcu. 2021. Spiking Neural Networks in Spintronic Computational RAM. *ACM Trans. Arch. Code Optim.* 18, 4, Article 59 (September 2021), 21 pages.
https://doi.org/10.1145/3475963

1 INTRODUCTION

Spiking Neural Networks (SNNs) are neural networks mimicking the event-driven neural transmission in the brain. As a biologically inspired computational model capable of emulating neural computation in human brain and brain-like structures [17, 45], SNNs pose as an ideal computing medium for event-driven processing [46]. SNNs can perform a variety of computational tasks with significantly lower power consumption, such as prosthetic brain-machine interface control [12] or classification tasks like speech recognition [51].

This work was supported in part by NSF grant no. SPX-1725420.
Authors. address: H. Cılasun, S. Resch, Z. I. Chowdhury, E. Olson, M. Zabihi, Z. Zhao, T. Peterson, K. K. Parhi, J.-P. Wang, S. S. Sapatnekar, and U. R. Karpuzcu, University of Minnesota, Twin Cities, 200 Union St SE, Minneapolis, Minnesota, 55455; emails: {cilas001, resc0059, chowh005, olso6834, zabih003, zhaox526, pete9290, parhi, jpwang, sachin, ukarpuzc}@umn.edu.
Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
© 2021 Association for Computing Machinery.
1544-3566/2021/09-ART59 $15.00
https://doi.org/10.1145/3475963

Large-scale SNNs needed to perform useful computational tasks inevitably come with increased data access and parallelism demand that often exceeds capabilities of traditional hardware. When neuron count increases significantly, more space is required to store synaptic parameters. Data access and retrieval becomes a burden at the same time, mainly because any spiking information emerging in neurons should be accessible to all other neurons as an input.

As large-scale SNN computations are massively parallel and induce high-intensity memory accesses for data retrieval, the energy consumption skyrockets at scale. To put this into perspective, SpiNNaker requires 90 kW of power [34] for processing 1 billion neurons in real time, which would scale to 7.7 MW for 86 billion [26] neuron count in human brain even under optimistic linear scaling assumptions. This falls short by more than six orders of magnitude difference compared to the often cited 20-W power consumption in human brain [29], which clearly shows that scaling is quite costly. This is why true in-memory computing substrates such as the recently proposed spintronic **Computational RAM (CRAM)** [48], enabling massively parallel and reconfigurable logic operations *in situ* without compromising energy efficiency, represent especially promising platforms for SNN hardware, which forms the focus of this article. Our data-centric design features a network of accelerator arrays, where each node represents an in-memory SNN accelerator based on a CRAM array, and a novel, De Bruijn graph based architecture establishes SNN array connectivity to minimize data communication overheads. In a nutshell, our contributions are as follows:

- We present a novel, communication-centric SNN accelerator that minimizes data transfer overheads by (i) using non-volatile in-memory accelerator arrays at the node level and (ii) introducing a scalable topology for array connectivity.
- We explore the accelerator design space and cover algorithm mapping details along with sensitivity to technology parameters.
- The proposed accelerator design can reduce energy consumption significantly (by up to more than two orders of magnitude) when compared to a recent representative ASIC implementation.

We will start our discussion with background information covering related work in Section 2, and basics of CRAM and the SNN model in Section 3. Section 4 details the proposed SNN architecture based on CRAM. After quantitative evaluation in Section 5, we conclude the article in Section 6.

2 RELATED WORK

Recent efforts [25] focus on flexible and efficient hardware emulation of different biologically accurate SNN models, whereas others exploit the computational power resulting from the high number of relatively simpler neurons in SNNs. Low-energy hardware SNN accelerators such as ODIN [16] try to minimize the energy per spiking event. SNN hardware solutions include IBM's TrueNorth [30], Intel's Loihi [10], University of Manchester's SpiNNaker [17], and Human Brain Project's BrainScaleS [41]. The present highest number of spiking neurons in a hardware SNN implementation is 1 billion real-time neurons in SpiNNaker [21]—only a fraction of the average neuron count of 86 billion in human brain [26].

Various solutions based on traditional CMOS as well as emerging spintronic and resistive technologies are proposed to overcome this bottleneck [32]. These proposals typically only address a single step of SNN computation, which takes the form of weight multiplication by spike trains in a crossbar setting for recent **Magnetic Tunneling Junction (MTJ)** based spintronic SNN accelerators [6, 22, 42, 50]. Flexibility is also a concern, as the SNN model as well as parameter resolution is hardwired. Spintronic (MTJ based) devices have also been proposed as analog SNN building blocks

[54]; however, analog implementations by construction suffer from process variability constraints [47]. Several other implementations utilize customized sense amplifiers to perform computations near memory, including FeFETs [36], SOT-MRAM based in-memory graph processing [1], and STT-MRAM based binary neural networks [35]. Unfortunately, these designs introduce additional complexity to the peripheral circuitry and do not completely eliminate the memory access overhead. This results in a suboptimal design as tighter coupling between the computation and the memory is still possible. Other implementations include various other works [2, 3, 22, 31].

It is important to note that RRAM and SRAM based universal true in-memory computing approaches also exist, yet MTJ based SHE CRAM is more energy efficient. For example, compared to the SRAM and RRAM based 3-input majority gate implementations in the work of Gupta et al. [18], CRAM in the work of Zabihi et al. [53] consumes 4.6× and 4.48× less energy, respectively. RRAM and MTJ based technologies have a clear advantage over SRAM due to their non-volatility. When compared to MTJ, however, RRAM suffers from low endurance, which limits practicality. Up to 1E+20 cycles of endurance are reported for MTJs, whereas RRAM endurance remains below 1E+10 [49]. As a representative example, MAGIC [6, 23, 24], a similar in-memory computing platform to CRAM (in terms of logic operation principle) yet with predominantly RRAM based implementations, suffers from these limitations.

3 BACKGROUND

We will first cover CRAM basics (Section 3.1) and continue with the feedforward Leaky Integrate-and-Fire SNN model (Section 3.2). Although only the feedforward model is useful for many computational tasks, applications such as brain simulation need the parameters to be learned online. To this end, we also include pairwise **Spike-Time Dependent Plasticity (STDP)** learning algorithm for updating parameters during computation (Section 3.3).

3.1 CRAM Basics

Essentially, CRAM [48] augments conventional spintronic memory arrays with compute capability, thereby enabling seamless memory access. As long as there is no computation, CRAM reduces to an ordinary memory. When computation is enabled, CRAM performs logic operations directly inside the memory array. **Spin Torque Transfer (STT)** and **Spin-Hall Effect (SHE)** or Spin-Orbit Torque (SOT) based CRAM variants exists [8, 38, 52, 53]. CRAM can perform one logic operation (Boolean gate) in a column at a time, but all (or a desired subset of) columns can perform the same operation in parallel. Each cell in a column can serve as an input or output to a Boolean gate. Logic operations are reconfigurable in time and space. At the same time, multiple CRAM arrays can perform the same computation (across all of their columns) in parallel.

CRAM's main storage element is an MTJ, which comprises a fixed-polarity magnetic layer, a variable-polarity magnetic layer, and an insulating layer in between. When polarities of the magnetic layers (mis)match, the MTJ is in (Anti)Parallel-(A)P state that corresponds to logic (1) 0, exhibiting a (high) low resistance.

Cell structures for STT- and SHE-CRAM are provided in Figure 1(a) and Figure 1(b), respectively, which depict a sample of three cells in a column. CRAM features even and odd bitlines (BLE/O), which are used to sense (change) the state of the MTJ in read (write) mode. For logic operations, BLE/O is used determine which MTJ serves as an input or output. Logic lines (LL) connect input and output cells to perform Boolean operations. Wordlines (WL) are used to select the rows for both memory and logic operations. Specifically, to perform a logic gate in Figure 1(a), first the output MTJ is preset to a known logic value depending on the type of the gate. If inputs reside in even rows, the output should be in an odd row, and vice versa. By imposing a predefined voltage difference between BLE and BLO, a current is induced through (a parallel connection of) inputs and

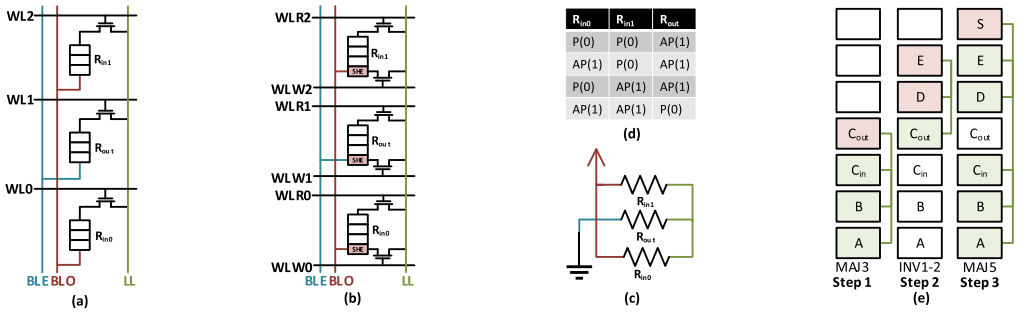


Fig. 1. (a) 1T1M STT-CRAM. (b) SHE-CRAM. (c) Equivalent resistive network for a NAND gate. (d) NAND truth table. (e) Three-step full adder implementation (with each box depicting a cell). MAJ3 and MAJ5 are 3-input and 5-input majority gates. INV1−2 is an inverter that writes the output to two different cells simultaneously. This is the time evolution of the very same column over three steps.

the output in series. The magnitude and direction of this current depends on the voltage difference between BLE and BLO, which also is a function of the type of the gate to be performed. For a given voltage difference, the current through the output evolves as a function of the parallel equivalent resistance of the inputs, and may or may not be enough to switch the output state. This is the main principle behind how CRAM implements truth tables. The equivalent resistive network for the universal NAND gate is given in Figure 1(c) along with the truth table in Figure 1(d). CRAM supports a rich set of (universal) gates beyond NAND, each characterized by specific bitline voltage and output preset values. More complex logic functions such as full adders can be synthesized by a sequence of basic gate operations as shown in Figure 1(e) [7].

SHE-CRAM, as shown in Figure 1(b), however, features a four terminal cell element that enables faster memory and logic operations while consuming significantly less energy. This is because SHE-CRAM separates read and write paths, making independent optimization of each possible (which otherwise impose conflicting constraints). Accordingly, SHE-CRAM has separate read (WLR) and write (WLW) wordlines, and two access transistors per cell. SHE-CRAM hence has a higher area footprint compared to the STT-CRAM, which is compensated by lower energy and faster operation. Memory and logic operations follow the same principles otherwise.

CRAM features two levels of parallelism: *column level parallelism*, where the same logic operations are performed in multiple columns simultaneously, and *array level parallelism*, which enables multiple STT/SHE-CRAM arrays to perform computation at the same time. As all computation directly takes place within the memory array (i.e., there is no need to transfer data out of the array, not even to periphery, for logic processing), CRAM represents an ideal in-memory processing platform. To ease illustration, in the following, we will use *transposed* CRAM arrays, and abstract out the interleaved placement where inputs (outputs) reside in even (odd) rows or vice versa.

3.2 Leaky Integrate-and-Fire SNN Model

Although there exist more complex and biologically accurate models for SNNs, the *Leaky Integrate-and-Fire* model, as described in the work of Davies et al. [10], is widely used due to its relative simplicity. Figure 2 highlights the computational steps involved in calculating the output (*spike train* or spike function) of neuron n_i from the spike trains received at its input. A neuron transmits information (in the form of a spike train) to others only if its *membrane potential* exceeds a threshold. And a neuron's membrane potential itself evolves as a function of the spike trains received at its input, as we will discuss briefly. Table 1 provides the definitions for all model parameters.

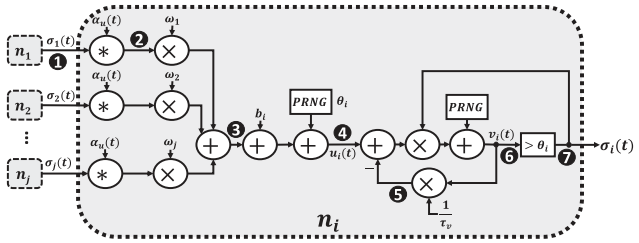


Fig. 2. Leaky Integrate-and-Fire model [10]. ∗, ×, +, >, and *PRNG* represent convolution, multiplication, addition, comparison, and pseudo-random number generation. Table 1 provides the definitions for all model parameters.

Table 1. Leaky Integrate-and-Fire SNN Model Parameter Definitions

Parameter	Definition
t	Discretized time variable
$\sigma_j(t)$	Spike train at the output of neuron j
τ_u	Synaptic time constant
τ_v	Neural time constant
θ_i	Spiking threshold constant of neuron i
ω_{ij}	Weight variable between neurons i and j
d_{ij}	Delay variable between neurons i and j
b_i	Bias variable of neuron i
$u_i(t)$	Synaptic response current of neuron i
$v_i(t)$	Membrane potential of neuron i
$A_{\pm}$	Time constant
$t^{pre/post}$	Presynaptic/postsynaptic spike time
$\alpha_u(t)$	$\frac{1}{\tau_u} e^{-\frac{t}{\tau_u}} H(t)$
$\mathcal{F}(t)$	$e^{-\frac{t}{\tau_u}} H(t)$
L_f	#Entries in the lookup table of $\alpha_u(t)$
S	Weight bit length
t_{max}	Maximum time difference for STDP
$r(t)$	Pseudorandom noise
N_{max}	Maximum number of input spikes

Each connection from neuron j to neuron i is characterized by a weight ω_{ij} and a delay value d_{ij} (corresponding to the actual transmission delay in brain). Each spike train (or spike function) can be expressed as $\sigma(t) = \sum_k \delta(t - t_k)$, where δ is the unit impulse function; t, the discrete time variable; and t_k, the time difference corresponding to the k^{th} spike.

The weighted sum of filtered input spike trains with bias is called *synaptic response current*, as depicted in Equation (1), where $u_i(t)$ represents the synaptic response current of neuron i:

$$u_i(t) = \sum_{j \neq i} \omega_{ij} (\alpha_u * \sigma_j)(t) + b_i \text{ where } \alpha_u(t) = \frac{1}{\tau_u} e^{-\frac{t}{\tau_u}} H(t - d_{ij}). \quad (1)$$

$H(t)$ here is the unit step function; τ_u, a time constant; and b_i, the bias. In the summation provided in (1), the arrival of the spikes from each input (i.e., presynaptic) neuron j, σ_j, is delayed by d_{ij}. In other words, the input neuron j is only taken into consideration after time characterized by d_{ij}

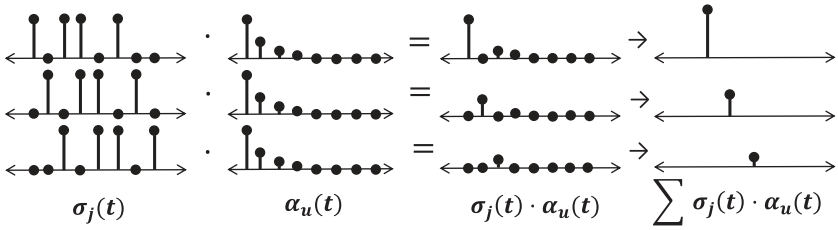


Fig. 3. Time evolution for convolution of input spikes (σ_j) with the filter α_u.

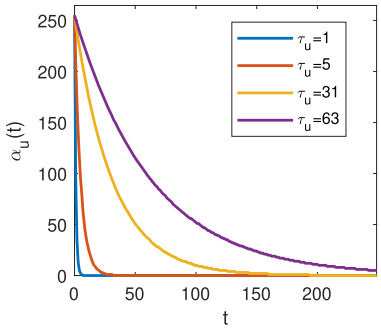


Fig. 4. $\alpha_u(t)$ function for representative values of the synaptic time constant τ_u.

elapses. Here the convolution is implemented as a dot product between shifted samples of input spikes (σ_j) and the filter (α_u). For each time instance, adding samples of the dot product result together effectively corresponds to a new sample of the convolution. This operation is depicted in Figure 3.

Update function for membrane potential is given by Equation (2), where $v_i(t)$ captures the membrane potential of neuron i:

$$\dot{v}_i(t) = \left(-\frac{1}{\tau_v} v_i(t) + u_i(t)\right) \sigma_i(t). \tag{2}$$

τ_v here is a time constant. The neuron i spikes if $\dot{v}_i(t)$ exceeds the threshold value θ_i, and if the neuron spikes, $v_i(t)$ is reset to zero.

Figure 4 shows $\alpha_u(t)$ function considering representative values of the synaptic time constant τ_u. It is important to note that the function decays quickly, which enables logic optimizations for savings in array utilization, as we will cover in the following sections.

Variations of this basic algorithm include the Random Sampling algorithm from Loihi [10], which our design is based on. Support for such variations is enabled by introducing an increment of the synaptic response current and the membrane potential by a pseudorandom number, which we will encapsulate in the following discussion in a noise term $r(t)$ (and which is generated by *PRNG* blocks in Figure 2).

3.3 Pairwise STDP Based Parameter Learning

Learning entails adjusting synaptic weights (ω_{ij}) and delays (d_{ij}) to dynamically optimize the SNN for solving a specific problem. It models the actual synaptic changes that brain adapts to, to accommodate new constraints.

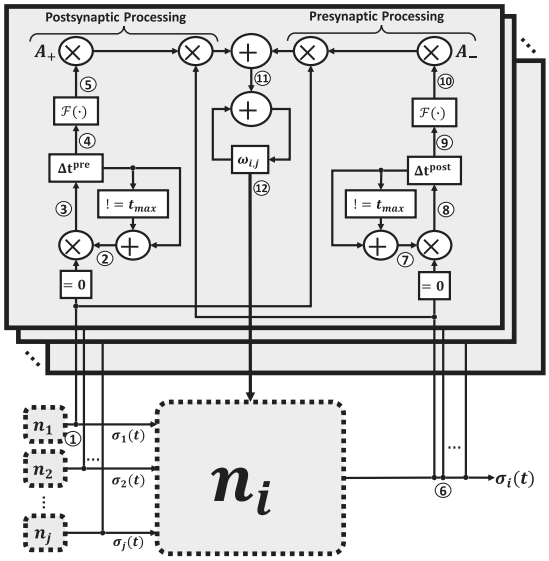


Fig. 5. Pairwise STDP based parameter learning.

The STDP model features a simple weight learning rule, as described in other works [9, 10]. It can be summarized as follows:

$$\Delta\omega_{i,j} = \begin{cases} A_{-}\mathcal{F}\left(t - t_i^{post}\right), & \text{on presynaptic spike} \\ A_{+}\mathcal{F}\left(t - t_j^{pre}\right), & \text{on postsynaptic spike.} \end{cases} \tag{3}$$

Presynaptic and postsynaptic spikes correspond to spike events (trains) received at the input and fired at the output, respectively. $\mathcal{F}(t) = e^{-\frac{t}{\tau}}H(t)$ and τ, A_{-}, and A_{+} are constants, as defined in Table 1. $\Delta\omega_{i,j}$ is added to the weight $\omega_{i,j}$ in each update. Figure 5 provides a block diagram description of the basic STDP algorithm. We define $\Delta t^{pre(post)}$ as $t - t_{j(i)}^{pre(post)}$. The algorithm continuously checks whether the current neuron or an input (presynaptic) neuron spikes. If so, it resets the $\Delta t^{pre(post)}$ counter by multiplying its former value by zero. Otherwise, it checks for overflow and increments the $\Delta t^{pre(post)}$, which is then fed to $\mathcal{F}(\cdot)$ function and multiplied by $A_{+(-)}$. The same STDP algorithm described in Figure 5 can be used to update synaptic delay parameters in addition to the synaptic weights without loss of generality.

4 SNN IN CRAM

4.1 Leaky Integrate-and-Fire-Model in Memory

CRAM supports universal Boolean gates and hence can handle any type of computation including the Leaky Integrate-and-Fire Model described in Section 3.2. The mapping is straightforward, as shown in Figure 6, where each neuron gets processed inside a single CRAM array to exploit array-level parallelism.

4.1.1 Initialization. Our design incorporates several lookup tables and dedicated storage for parameters and constants, which are initialized before feedforward computations start. The initialization is a one-time procedure, as the corresponding values do not change during computation. We also initialize an S-bit *local delay* value for each synapse, which serves modeling synaptic delays between neurons.

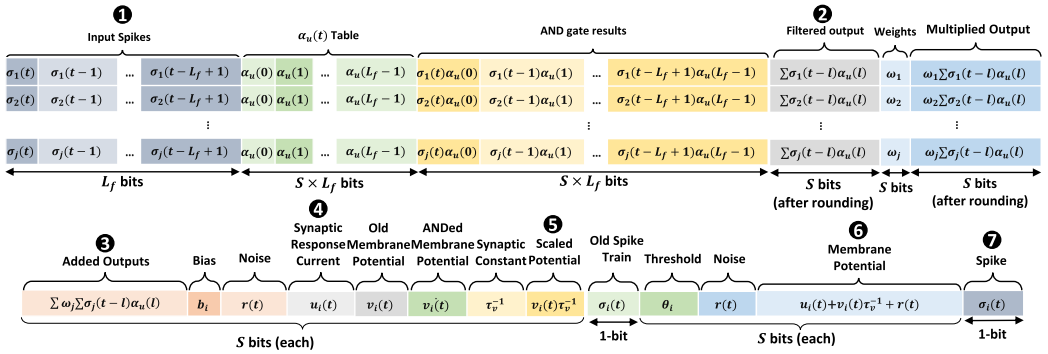


Fig. 6. Data layout (transposed, with each row corresponding to a column of CRAM). The steps follow the steps of the algorithm from Figure 2.

Table 2. CRAM Array Utilization by Constants, Parameters, and Lookup Table for Different S and L_fs

S	L_f	Array Size	Utilization
1−4	32	256 × 256	14.06%−56.25%
5−8	32	512 × 512	35.15%−56.25%
9−16	32	1024 × 1024	31.64%−56.25%
1−2	64	256 × 256	26.55%−53.13%
3−4	64	512 × 512	39.84%−53.13%
5−8	64	1024 × 1024	33.2%−53.13%
9−16	64	2048 × 2048	29.88%−49.8%

Precomputed α_u values reside in a lookup table. α_u lookup table consists of L_f different S-bit values (Table 1). Similarly, constant $\frac{1}{\tau_v}$ is initialized once, as well as weights ω_{ij}, b_i, and θ_i values. For the α_u lookup table, Table 2 shows the utilization for different CRAM array sizes for various values of S and L_f. We choose the array size so that the utilization does not exceed approximately half of the array size. Thereby, enough space is left to perform the arithmetic/logic operations in remaining CRAM space. L_f values of 32 and 64 allow the decay in $\alpha_u(t)$ (as captured by Figure 4) to be represented with sufficient accuracy without compromising the memory footprint. Also note that the actual number of active rows depends on the presynaptic spike activity.

4.1.2 Dataflow. Once a spike train of presynaptic neurons is received, it is written in the memory as shown in the first column in Step ❶. All of the (previously received and similarly placed) spikes and the $\alpha_u(s)$ values are multiplied where $s \in [0, L_f - 1]$ and L_f is the predetermined filter length. This filtering operation effectively corresponds to the convolution ($*$) from Equation (1). Since all of the values involved are binary, this operation reduces to $S \times L_f$ AND operations where S is the bit length of each entry in the $\alpha_u(s)$ lookup table, which keeps precomputed values of the $\alpha_u(s)$ function. Once ANDs are computed, the resulting L_f entries are added together and rounded (Section 4.1.3) to S bits to obtain the column denoted by ❷ in Figure 6.

Then, the resulting S bit values are multiplied by S bit weights, which corresponds to S^2 full adder operations and results in the column marked by Step ❸. The $2S$ multiplication outcome is next rounded to S bits—that is, a $(S-1)$-bit rounding factor is added to it and the result is truncated to S bits. Addition is again performed as a cascade of full adder operations. Truncation

has practically no overhead as it translates into simply ignoring the unused bits. Equation (1) gives rise to the operations we covered so far, spanning Step ❶ to Step ❸.

In the following step, all rows in Step ❸ are added by reading half of the rows and writing them back to the adjacent rows. After each addition, a rounding operation is performed. This operation takes $\log_2 N_{max}$ steps, where N_{max} is the predetermined maximum number of presynaptic neurons. Once the addition is done, the result is reduced to a single row as shown in Step ❹. After the bias and the noise is added to the outcome from Step ❹, synaptic response current shown in Step ❺ is obtained. We keep older membrane potential in the same row, which is next scaled—by multiplication with ${\tau_v}^{-1}$. Equation (2) spans Step ❹ to Step ❻. The current membrane potential is obtained after current synaptic response current, scaled old membrane potential, the noise, and the scaled old spike train is added as shown in Step ❻. Current membrane potential then overwrites the old membrane potential. Finally, current spike value is calculated by thresholding the current membrane potential in Step ❻, which corresponds to the comparison of S bits. Current spike value is updated in Step ❼ and copied to the old spike train column. To reset the membrane potential, we invert the spike bit and AND it with the membrane potential. The output spike is then read and broadcast as we will explain in Section 4.2.

Before starting computation, the S-bit local delay value in all columns is incremented and compared to d_{ij} (for the corresponding synaptic connection). If the comparison yields true, the local delay value is reset to zero. The comparison output is then read by the array controller and used as column enable. This results in a practical synaptic delay implementation, as well as energy savings, as the disabled columns do not participate in computations described previously to perform Equations (1) and (2).

4.1.3 Low-Level Operations. For addition operations, we use the full adder from Figure 1. Multiplication also uses the same full adder design N^2 times for N-bit numbers. Multiplication is implemented in standard fashion since bit lengths are relatively low. In the convolution with $\alpha_u(t)$, the multiplications reduce to AND operations since the spikes are binary. Comparators are implemented as a subtraction operation, followed by using the sign bit in case of magnitude comparison, or a series of AND operations when equality comparison is needed. We increase the bit length conservatively until Step ❸ and then perform rounding by adding rounding factors in each arithmetic operation, which ensures that no overflow happens. Rounding to S bits entails adding a $(S-1)$ bit long rounding factor and then keeping only the S most significant bits.

Figure 7 demonstrates a 2×2-bit binary multiplication in SHE-CRAM array, which represents the key building block for multiplication operations and can be scaled to any number of bits.

Loihi [27] features a Pseudorandom Noise Generator to support several variants of the basic algorithm such as neural sampling. As a proof of concept, we demonstrate how to implement a Pseudorandom Noise Generator in CRAM using a basic linear feedback shift register in Figure 8. First, an XOR gate is applied as a cascade of four NAND gates and then COPY operations are performed as many as the length of the feedback polynomial. For an example 9-bit polynomial, this operation takes 13 cycles. We denote the generated random number with $r(t)$ and update it upon each use.

Our lookup table based implementation of $\alpha_u(t)$ function and the inevitably limited bit length for parameters such as weights, by construction, affect synaptic accuracy. Section 5 provides a quantitative characterization of the accuracy impact of precision.

4.2 Routing and Connectivity

As one CRAM array is allocated to process each neuron, the connectivity between neurons directly translates into the connections between arrays to transfer spikes. Ideally, each neuron in SNN would be connected to all of the other neurons. However, such a fully connected network

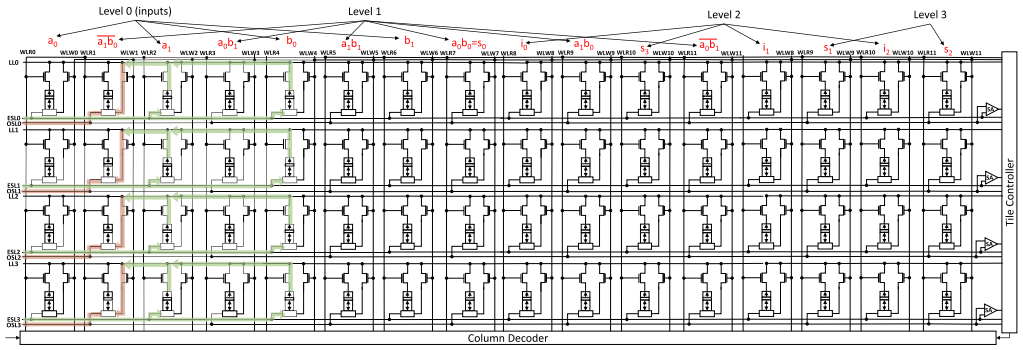
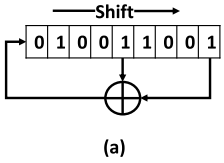


Fig. 7. 2×2-bit binary multiplication (of a_1a_0 and b_1b_0) in a SHE-based CRAM array proceeds as follows. A controller array provides the driver line values (LL, ESL, OSL, WLR, and WLW). Using these driver values fetched from the controller array, AND, NAND, and INV gates are performed in a cascaded manner. Before the computation starts, output cells at each logic depth level (Levels 1, 2, and 3) are preset to appropriate values depending on the gate type to be performed at that particular level. The top portion of the figure marks the Boolean depth level of the cells in the logic network along with the bits (in red) generated by each column. Since Level 0 resides in even columns, Level 1 should reside in odd columns, following basic SHE CRAM semantics. Steps of the first logic gate operation, NAND(a_1, b_0) are highlighted with green and red arrows. In all rows, the currents passing through a_1 and b_0 cells get combined in the LL, and pass through the SHE channel in the cells marked with column label $\overline{a_1b_0}$. This essentially computes $\overline{a_1b_0}$ operation as the name suggests. a_0b_1, a_1b_1, a_0b_0, a_1b_0, and $\overline{a_0b_1}$ are calculated in a similar fashion. Then, intermediate value i_0 is obtained by NANDing the first two level 1 cells. Intermediate value i_1 is obtained by INV operation on a_0b_0, and i_2 is obtained by ANDing last two level 1 cells. Finally, s_1 is calculated by NANDing i_0 and i_2, and s_2 is calculated by ANDing i_1 and a_1b_1, thereby producing the 4-bit multiplication result $s_3s_2s_1s_0$.

requires $\binom{N}{2}$ connections between neurons (i.e., arrays), which is not feasible in hardware when the number of neurons is in the order of billions, due to the limited area. To eliminate the connectivity burden, a possible solution is using a 2D network as implemented in other works [10, 17, 30]. In the work of Furber et al. [17], the 2D topology is further extended to a 3D Torus alongside additional diagonal connections and emergency routes. Such implementations involve each spike event to be transmitted as a packet in the mesh. This and similar solutions with more involved networks are subject to congestion in the packet traffic, eventually leading to packet drops.

Instead, we use the **Generalized De Bruijn Graph (GDBG)** topology [11] to connect CRAM arrays. GDBG has been used in High Performance Computing, and it is shown to be the near-optimal for load-balanced networks [15]. The first advantage of GDBG is that each vertex has four edges at most, which is comparable to a 2D mesh. The second advantage is that the shortest path between any two vertices is log_2N for a graph with N vertices.

Figure 9 provides a comparison of GDBG to representative 2D Mesh and Torus variants using the tabulation given in the work of Chae et al. [5]. Here the diameter is defined as the number of edges between the farthest vertices in the graph, and the bisection width is the number of edges in the largest cross section. The cost is defined as the overall number of edges. Although GDBG reduces the diameter significantly, this comes at the expense of higher layout complexity. However, overall asymptotic cost is not affected. Although the cost is similar when it is defined as the number of edges in the graph, the length of the edges is not necessarily the same in all alternatives. 2D Mesh only has unit size connections, whereas 2D Torus has long connections at the edges. SpiNNaker's Torus has longer edges in general, since it has 6-connectivity and skip connections. Although the maximum link length in GDBG is less than the Torus, it has a range of link lengths,



(a)

Time	Memory Layout													Operation
t_n	b1	b2	b3	b4	b5	b6	b7	b8	b9	t	a0	a1	a2	PRESET(t,a0,a1,a2)
t_{n+1}	b1	b2	b3	b4	b5	b6	b7	b8	b9	t	a0	a1	a2	NAND(b5,b9,a0)
t_{n+2}	b1	b2	b3	b4	b5	b6	b7	b8	b9	t	a0	a1	a2	NAND(b5,a0,a1)
t_{n+3}	b1	b2	b3	b4	b5	b6	b7	b8	b9	t	a0	a1	a2	NAND(b9,a0,a2)
t_{n+4}	b1	b2	b3	b4	b5	b6	b7	b8	b9	t	a0	a1	a2	NAND(a1,a2,t)
t_{n+5}	b1	b2	b3	b4	b5	b6	b7	b8	b9	t	a0	a1	a2	COPY(b8,b9)
t_{n+6}	b1	b2	b3	b4	b5	b6	b7	b8	b9	t	a0	a1	a2	COPY(b7,b8)
t_{n+7}	b1	b2	b3	b4	b5	b6	b7	b8	b9	t	a0	a1	a2	COPY(b6,b7)
t_{n+8}	b1	b2	b3	b4	b5	b6	b7	b8	b9	t	a0	a1	a2	COPY(b5,b6)
t_{n+9}	b1	b2	b3	b4	b5	b6	b7	b8	b9	t	a0	a1	a2	COPY(b4,b5)
t_{n+10}	b1	b2	b3	b4	b5	b6	b7	b8	b9	t	a0	a1	a2	COPY(b3,b4)
t_{n+11}	b1	b2	b3	b4	b5	b6	b7	b8	b9	t	a0	a1	a2	COPY(b2,b3)
t_{n+12}	b1	b2	b3	b4	b5	b6	b7	b8	b9	t	a0	a1	a2	COPY(b1,b2)
t_{n+13}	b1	b2	b3	b4	b5	b6	b7	b8	b9	t	a0	a1	a2	COPY(t,b1)

(b)

Fig. 8. (a) Linear Feedback Shift Register (LFSR) based pseudorandom number generation using feedback polynomial x^9+x^5+1. (b) CRAM implementation of the given LFSR. Input (output) cells are highlighted with green (red). Here the cells b0-b9 are the past values in the LFSR, a0-a2 are the ancillary cells for intermediate values, and t is the new XOR value to be computed. t_n to t_{n+13} denote the time instances for the computation. For this operation, output cells are preset to logic 1 first. Then a cascade of four NAND operations compute XOR of b5 and b9 and write the result to the t cell, which effectively corresponds to calculating x^9+x^5 part of the feedback polynomial. Then a series of COPY operations shift the input vector and write the new t value into the b1 cell, which is the +1 part of the characteristic polynomial.

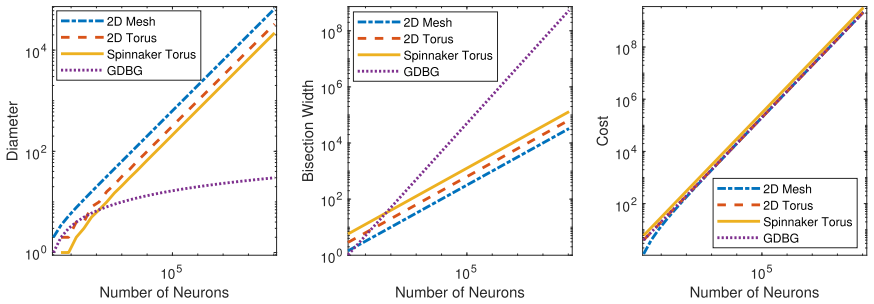


Fig. 9. Diameter, bisection width, and cost comparison for GDBG, 2D Mesh, and Torus variants.

which might make routing harder. Still, GDBGs can be routed with two routing layers [19], and algorithms for optimal routing are discussed in the work of Liu and Lee [28]. As a Network-on-Chip topology, GDBG results in less latency and area, which typically translates into lower power consumption [39].

As under GDBG the shortest path between any two vertices is log_2N, each neuron can reach any other neuron in (at most) log_2N steps. Figure 10 depicts an example for 16 neurons (labeled with 1-16), which reveals the similarity to the connectivity of Singleton's FFT [43]. Each edge corresponds to sets of wires between neurons.

In our design, each CRAM array, which corresponds to a neuron, is connected to one another in GDBG topology. Each connection consists of N_{max} wires, which is equal to the maximum number of input (presynaptic) neurons. Once the computation is done, the routing operations are initiated. The routing consists of log_2N steps. If the current step $c \in [1, log_2N]$ is smaller or equal to N_{max}, then each array (i.e., neuron) concatenates the input spike trains and transmits them

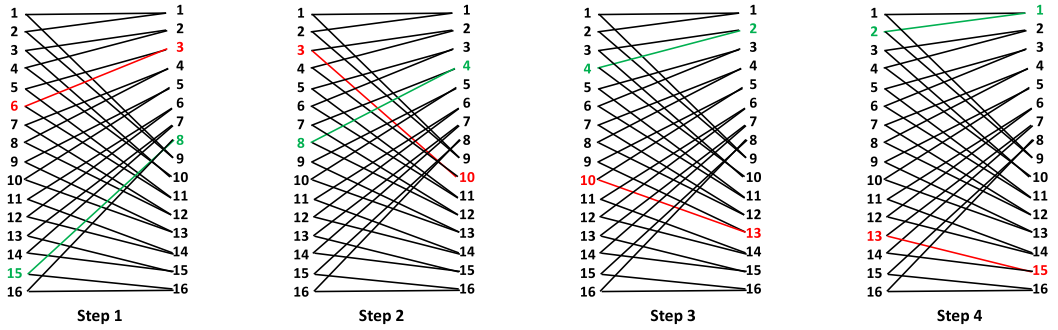


Fig. 10. Example worst-case paths on a 16-neuron network in GDBG topology, from neuron 15 to neuron 1 (highlighted in green) and from neuron 6 to neuron 15 (highlighted in red).

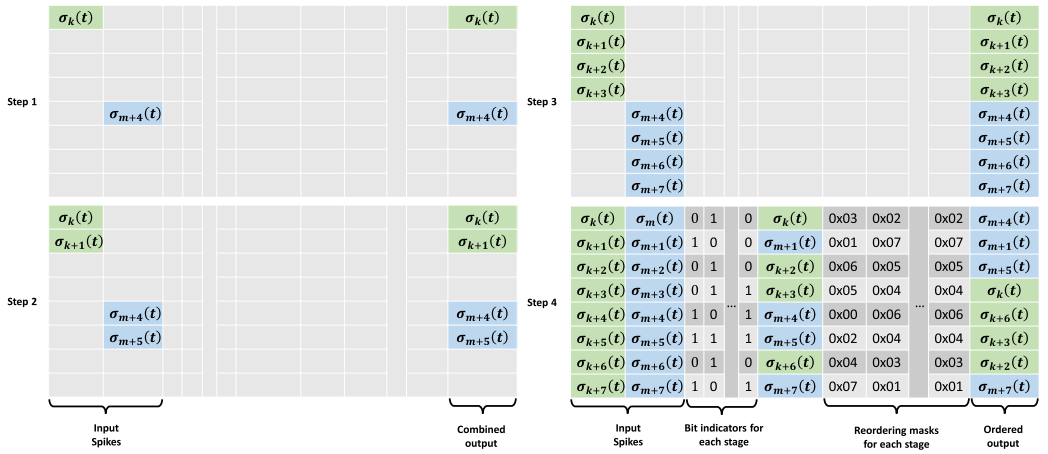


Fig. 11. Routing steps for an example 16-neuron (-array) architecture with a maximum of eight input spikes.

to the connected arrays for the next step. This operation involves only reading incoming spike trains and writing them in the predetermined locations. Therefore, no extra logic is involved if $c \leq \log_2 N_{max}$. However, if $c > \log_2 N_{max}$, each array combines the input spike trains using stored address values. Each array therefore reads $\log_2 N_{max}$-bit address values for each spike and copies the spike train to the corresponding addresses. Hence, upon initialization, each array has to store $(\log_2 N_{max})(\log_2 N - \log_2 N_{max})$ bits of address values. After routing is complete, the next spike computation starts, as discussed in Section 4.1.

Figure 11 demonstrates four routing steps of an example 16-neuron architecture: $N_{max} = 8$ and $N = 16$. In Step 1 ($c = 1$), the two spikes (highlighted in green and blue) coming from two input neurons are combined and transmitted. In Step 2 ($c = 2$), the number of incoming spikes doubles and combined spikes are forwarded to Step 3. When $c = 3$, each incoming spike train has a size of 4, and they are forwarded to the next step. In Step 4, however, $c = 4$ is greater than $\log_2 N_{max} = 3$ so half of the incoming spike trains should be discarded and the remaining spikes should be forwarded. For this purpose, we use a bit indicator for each step, as shown in Figure 11. The bit indicators select one of the input spikes (highlighted in green and blue). The selected input spike train is read and then written to the ordered output in the corresponding address specified in the current step's reordering (bit)mask. For this example, there is only one set of reordering

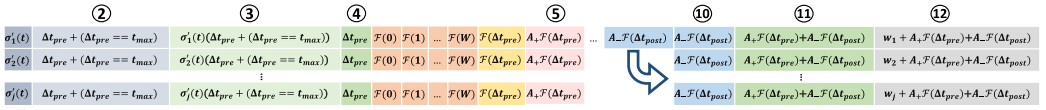


Fig. 12. Data layout for the STDP engine in CRAM (transposed).

masks and bit indicators; however, in general, there can be $\log_2 N - \log_2 N_{max}$ different sets of $(\log_2 N_{max})$-bit masks and bit indicator sets.

Our GDBG based design is efficient since we need $2N$ connections for N neurons instead of $\binom{N}{2}$; we avoid traffic handling that is required to implement NoC based architectures; and we can synchronize the whole system in deterministic time as each routing cycle entails a fixed amount of routing steps, which is $\log_2 N$ for N neurons.

4.3 STDP Learning Engine

4.3.1 Initialization. Initialization steps for the STDP engine are similar to the feedforward case (Section 4.1.1). Note that $\mathcal{F}(\cdot)$ is only a scaled version of $\alpha_u(t)$ (Table 1). Therefore, we re-use the lookup table for α_u and scale any fetched value from it by multiplying the fetched value with $\frac{1}{\tau_u}$. We also initialize all parameters and constants from Equation (3) such $A_{\pm}$.

4.3.2 Dataflow. In the STDP engine, every time spike distribution is completed, the presynaptic (postsynaptic) part of the operations is executed, as shown in the left (right) half in Figure 5, where the input corresponds to the spikes and the output is the weight. This effectively performs Equation (3). Note that the data dependency on presynaptic and postsynaptic spikes does not induce additional memory accesses since they are already stored in the same array in our mapping from Section 4.1.

In the transposed layout, ① corresponds to the presynaptic spikes. ② is the incremented time difference since the arrival of the latest presynaptic spike, and ③ resets the Δt^{pre} if the input neuron spikes. Then ⑤ is obtained by a lookup table implementation of $\mathcal{F}(\cdot)$ function that is fed by ④. These steps are similar for ⑦, ⑧, ⑨, and ⑩ except that these calculations are only performed in one row and the output is copied to all rows as shown in the layout in Figure 12. Finally, $\Delta\omega$ is added to ω as shown in ⑪ and ⑫.

4.3.3 Low-Level Operations. Addition and multiplication operations in the STDP are similar to the feedforward case as discussed in Section 4.1.3. Comparators are implemented as a subtraction operation as described in Section 4.1.3.

5 EVALUATION

To evaluate our design, we perform energy and performance analysis based on the low-level elementary operations such as Boolean gate implementations in the context of the proposed CRAM-based SNN architecture. Configuration parameters for the simulations are given in Table 3. For experiments, we have four different configurations where STT-M and SHE-M correspond to the current conservative estimates while STT-F and SHE-F reflect near-term future expectations for STT and SHE based MTJs. The near-term expectations are based on the configurations in other works [37, 38, 52, 53] along with the work of Jan et al. [20]. For array characteristics such as read/write timing, we use NVSim [14]. Each array has an *array controller* that is responsible for driving logic lines, bitlines, and wordlines. We take the overhead resulting from driving such lines

Table 3. Configuration Parameters

Parameter	STT-M, SHE-M	STT-F, SHE-F
P state resistance	3.15 $k\Omega$	7.34 $k\Omega$
AP state resistance	7.34 $k\Omega$	76.39 $k\Omega$
Switching Time	3 ns [33, 40]	1 ns [20, 52]
Switching Current	40 μA [40]	3 μA [52]
Bulk preset current limit	30 mA	

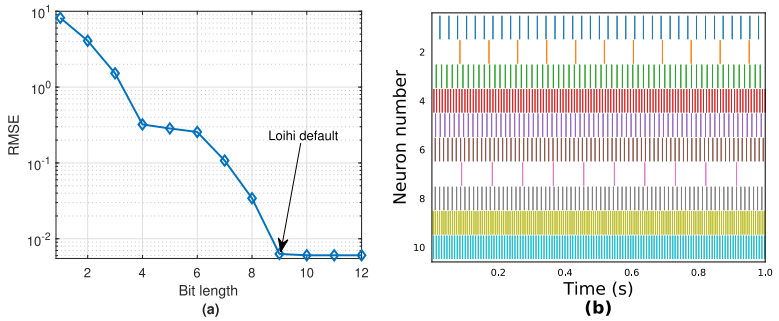


Fig. 13. (a) RMSE vs. $\log_2 L_f$ bit length for an example 10-neuron configuration in Loihi. (b) The spike-time diagram.

into consideration.[1] For synaptic events, we stick to a conservative setup where each presynaptic neuron spikes in each timestep and each neuron is connected to the maximum possible number of neurons (F). Since we base our neuron model on Loihi's implementation, we compare our results with the energy and time figures reported in the work of Davies et al. [10] and Lines et al. [27], using $L = 1024$ and $F = 1024$ for 1-bit synaptic weights (in their design, L is the neuron count per core; F, the postsynaptic fanout).

For accuracy analysis, we use Loihi library of Nengo framework [4] with Loihi configuration. Modifying Nengo Loihi's implementation, we analyze the spiking accuracy for our lookup table based limited precision design. This analysis is only used for Leaky Integrate-and-Fire feedforward implementation. Figure 13 shows how RMSE error changes with intermediate computation bit length compared to the baseline. Bit-precision analysis performed in Figure 13(a) does not necessarily translate to accuracy degradation and demonstrates the energy/time-accuracy tradeoff.[2] Figure 13(b) features a 10-neuron spike-time diagram used for the analysis.

Table 4 summarizes the results for 1 billion neurons, where the maximum number of presynaptic neurons is 1024, bit length is 1, and the filter (lookup table) size is 64. Each CRAM array has 1024×512 cells. Although all CRAM configurations can provide enough performance to implement SNN operations within a biologically plausible time budget (i.e., a spiking rate of several kHz),

[1]The controller can be implemented in many different ways. We assume a very conservative controller design, where control bits for the driver lines are preprogrammed into separate control arrays. During the execution, this controller array broadcasts the configuration parameters to multiple arrays. Providing a startup pulse, controller array rows are activated one by one.

[2]Note that this analysis only includes a low-level characterization of the proposed architecture. We directly use an existing SNN algorithm without any essential algorithmic changes to drive our hardware design; therefore, higher-level characteristics, including accuracy (for a given weight bit-length), stays the same when compared to our baseline, by construction. That said, extension to higher-level algorithmic examples, such as the ones discussed in the work of Srinivasan et al. [44] and Diehl and Cook [13] would be straightforward.

Table 4. Evaluation Results

Metric	STT-M	STT-F	SHE-M	SHE-F
Execution Time (μs)	6.342	0.825	6.342	0.825
Maximum Spiking Rate (KHz)	157.7	1212.3	157.7	1212.3
Energy (J)	133.37	2.53	31.36	0.25
EDP (μJs)	845.89	2.09	198.82	0.21

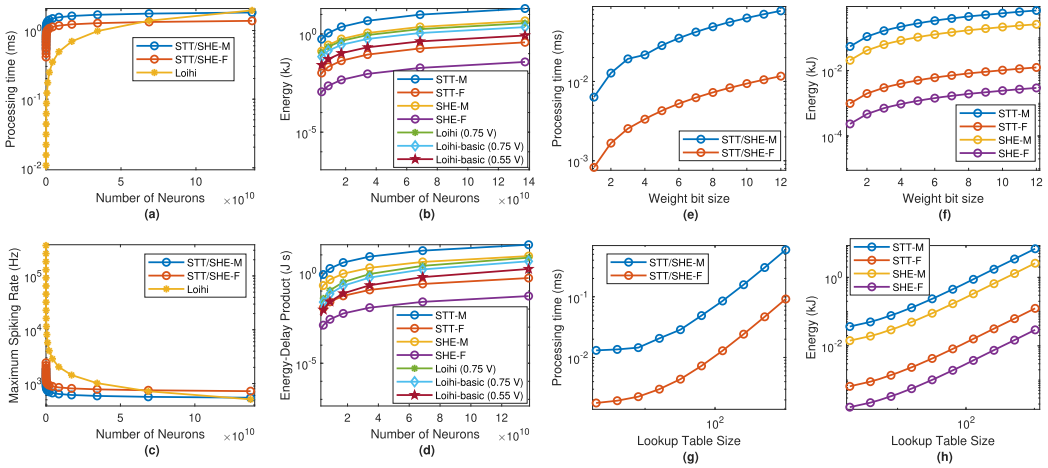


Fig. 14. Sensitivity to number of neurons, weight bit length, and α_u table size.

SHE-F configuration provides the lowest energy consumption and fastest operation. Note that although frequency scaling is an option, it is desirable to have a less than a millisecond latency so that real-time brain simulation is possible. Here the spiking rate is the maximum attainable operating frequency within the time budget imposed by our design constraints.

Figure 14 shows execution time, maximum spiking rate, energy, and **energy-delay-product (EDP)** for the case where maximum number of presynaptic neurons is 1024 and the bit length is 9, which is the maximum weight size that Loihi supports. Figure 14(a) and (c) are complementary figures for different CRAM variants compared to Loihi for similar parameters. Although Loihi is faster for a larger number of neurons because of its faster computation, CRAM variants surpass Loihi when neuron count approaches 100 billion thanks to our routing architecture. Figure 14(b) and (d) capture energy and EDP's sensitivity to neuron counts in several Loihi models with different voltages. For very large neuron counts, the best-performing CRAM variants have lower energy consumption than Loihi and higher energy efficiency (EDP). We also sweep bit length of weights and table size for α_u, as shown in Figure 14(e) through (h). Figure 14(e) and 14(f) show processing time and energy consumption for different weight bit lengths, and Figure 14(g) and (h) capture the processing time and energy for various lookup table sizes in the feedforward design. In all cases, STT/SHE-F or SHE-F configuration provide the lowest processing time and energy when compared to other CRAM variants, emphasizing the effect of the cell technology.

Overall, when compared to the best-performing Loihi baseline (0.55V) with a fanout of 1024, and 1-bit synaptic weights, SHE-F configuration consumes 24.2× less energy. At the same time, SHE-F is 3.72× more energy efficient (in terms of EDP) for the feedforward implementation as captured in Figure 14.

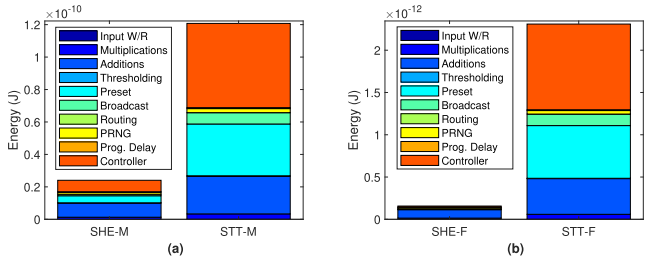


Fig. 15. Overall energy breakdown of operations in a single instance of spiking events.

Table 5. Energy, Latency, and EDP for Single Spike Operation

Synaptic Parameter	SHE-F	Loihi
Spike Operation Energy	157.07 fJ	23.6 pJ
Spike Operation Time	499.86 ns	3.5 ns
Spike Operation EDP	7.85e-20 Js	8.26 e-20 Js
STDP Update Energy	0.369 pJ	120 pJ
STDP Update Time	1321.7 ns	6.1 ns
STDP Update EDP	4.88e-19 Js	7.32 e-19 Js

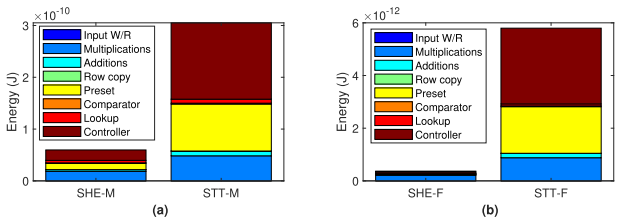


Fig. 16. Single-spike energy breakdown of STDP weight update operation.

Figure 15 shows the energy breakdown for a single spike operation with 1 bit weight. We observe that preset and additions dominate the overall energy followed by multiplication (i.e., AND) operations, pointing to further hardware optimization opportunities.

Table 5 summarizes low-level performance parameters of our best case CRAM-SNN design for 1-bit weights where $L_f = 32$ for feedforward computations. Also tabulated are Loihi's corresponding low-level performance parameters for comparison. STDP operations are performed at 10-bit precision. $L_f = 32$ is chosen to deliver enough accuracy for $\alpha_u(t)$ without compromising logic complexity to maintain a low energy consumption. A 10-bit precision suffices to provide enough bandwidth for updating up to 9 bit weights, which is meaningful for a fair comparison. Although being slower for low-level operations, our design consumes significantly less energy, which leads to a lower EDP for a single spike operation with a single presynaptic connection for 1 bit weights. That said, Loihi supports a variety of models and in general is a more capable architecture. Therefore, Table 5 is only provided to demonstrate where our implementation stands compared to Loihi for a single configuration of Leaky Integrate-and-Fire and STDP models.

Energy breakdown of STDP operations are given in Figure 16. Similar to the feedforward case, energy consumption is mostly dominated by preset, addition, and multiplication operations in STDP weight update. Figure 17 shows the STDP processing time and energy for larger bit lengths.

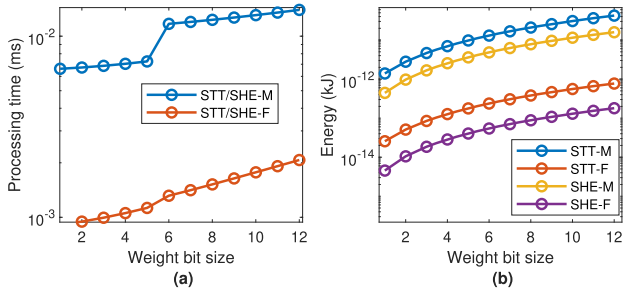


Fig. 17. (a) Processing time vs. weight bit length. (b) STDP update energy vs. weight bit length.

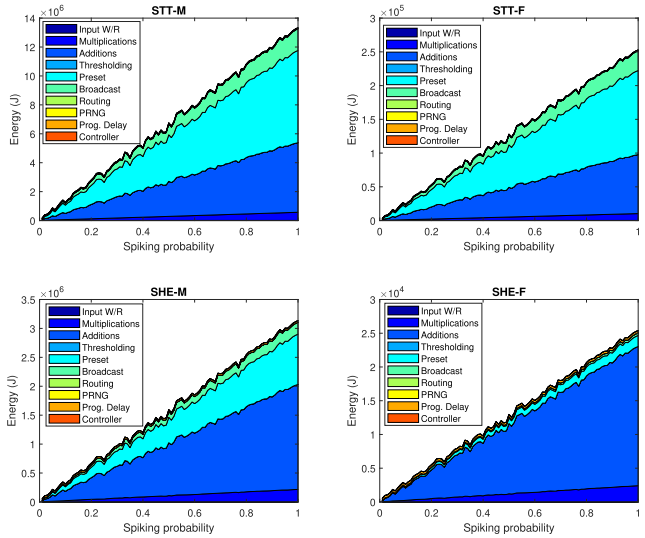


Fig. 18. Energy breakdown vs. spiking probability for different CRAM variants.

Processing times in Figure 17(a) irregularly increase around 6-bits as CRAM preset operations become overwhelming; however, this does not effect the energy consumption pattern in Figure 17(b) where the SHE-F configuration outperforms the other CRAM variants. Low-level performance results further show that our gains are not only due to the proposed connectivity scheme but also because of the low energy, massively parallel logic operations CRAM enables.

Figure 18 shows the energy breakdown for increasing spiking probabilities for different CRAM configurations. Although presets require a large amount of energy for STT-M, share of presets decrease and adder overhead dominates the overall energy consumption moving to SHE-F. Several energy figures, such as routing, are not affected by spiking probability, whereas the rest is proportional to it.

Both in Figure 15 and Figure 16, the controller overhead seems overwhelming. This is because we assume an unrealistically conservative controller overhead, and these figures only characterize the neuron operation with only one presynaptic neuron. In fact, this overhead diminishes when number of presynaptic neurons increase, as illustrated in Figure 18.

To demonstrate that CRAM improvement is not only due to the technology, we compare the energy figures of a hypothetical GDBG based Loihi implementation to the existing 2D mesh

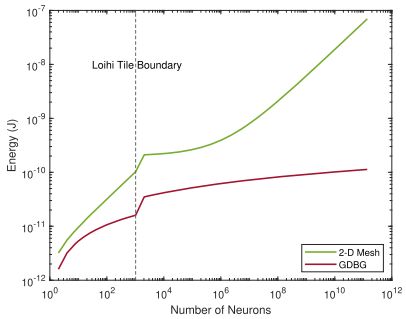


Fig. 19. Spike energy comparison of hypothetical GDBG and default hierarchical 2D Mesh implementation for Loihi. This limit study ignores the link latencies in both 2D Mesh and GDBG, and only considers the routing energy overhead due to the chosen topology.

based Loihi in Figure 19. The comparison takes existing core and tile boundaries and resulting limitations into consideration, and shows that GDBG implementation would improve the default Loihi significantly.

A quantitative comparison to crossbar based designs would not be necessarily more fair since the crossbar typically only addresses one part of the computation (i.e., weight multiplication with spikes and addition), as is the case for RESPARC [2] or STT-RAM based [22]. A similar pattern applies to SpinalFlow [31], where a large portion of the operations are offloaded to the logic in the crossbar periphery. SNN models are usually complex, and many alternative approaches exist even for the most commonly used algorithms. Furthermore, more biologically accurate model behavior is often simplified, and notions such as filtering, membrane potential feedback, variable delays, refractory period, and online learning are not implemented or offloaded to peripheral logic. The main advantage of our design is that logic operations are performed purely in-memory regardless of the model—therefore, if the model is flexible (i.e., can be changed or adjusted according to the user needs), the modification only entails changing the content of the controller array that stores driver line values. This way, no fixed control logic is necessary, and no external logic is involved throughout the computation.

6 CONCLUSION

We present a novel, networked SNN hardware, where each node represents a CRAM-based SNN accelerator processing a neuron, and a GDBG based topology establishes SNN array connectivity to minimize data communication overheads. We thereby achieve a limited full connectivity by using only $2N$ connections in a N neuron network, instead of $\binom{N}{2}$, while exploiting the massive intra- and inter-array parallelism and energy efficiency of CRAM for logic operations. Our best configuration results in 150.25× less energy consumption and similar EDP for feedforward operations, and 324.89× less energy consumption and 1.5× lower EDP for learning (via STDP) when compared to the alternative Loihi architecture. We also evaluate the effect of technology boost as well as GDBG based improvement separately, and show that both concepts contribute to the performance of the overall architecture.

REFERENCES

[1] Shaahin Angizi, Jiao Sun, Wei Zhang, and Deliang Fan. 2019. GraphS: A graph processing accelerator leveraging SOT-MRAM. In *Proceedings of the 2019 Design, Automation, and Test in Europe Conference and Exhibition (DATE'19)*. IEEE, Los Alamitos, CA, 378–383.

[2] Aayush Ankit, Abhronil Sengupta, Priyadarshini Panda, and Kaushik Roy. 2017. Resparc: A reconfigurable and energy-efficient architecture with memristive crossbars for deep spiking neural networks. In *Proceedings of the 54th Annual Design Automation Conference*. 1–6.

[3] Anakha V. Babu, Osvaldo Simeone, and Bipin Rajendran. 2020. SpinAPS: A high-performance spintronic accelerator for probabilistic spiking neural networks. arXiv:2008.02189.
[4] Trevor Bekolay, James Bergstra, Eric Hunsberger, Travis DeWolf, Terrence C. Stewart, Daniel Rasmussen, Xuan Choo, Aaron Voelker, and Chris Eliasmith. 2014. Nengo: A Python tool for building large-scale functional brain models. *Frontiers in Neuroinformatics* 7 (2014), 48.
[5] S.-H. Chae, Jong Kim, Dongseung Kim, S. Hong, and Sunggu Lee. 1995. DTN: A new partitionable torus topology. In *Proceedings of the International Conference on Parallel Processing*. 84–91.
[6] Mei-Chin Chen, Abhronil Sengupta, and Kaushik Roy. 2018. Magnetic skyrmion as a spintronic deep learning spiking neuron processor. *IEEE Transactions on Magnetics* 54, 8 (2018), 1–7.
[7] Z. Chowdhury, J. D. Harms, S. K. Khatamifard, M. Zabihi, Y. Lv, A. P. Lyle, S. S. Sapatnekar, U. R. Karpuzcu, and J. Wang. 2018. Efficient in-memory processing using spintronics. *IEEE Computer Architecture Letters* 17, 1 (Jan. 2018), 42–46. https://doi.org/10.1109/LCA.2017.2751042
[8] Z. I. Chowdhury, S. K. Khatamifard, Z. Zhao, M. Zabihi, S. Resch, M. Razaviyayn, J. Wang, S. Sapatnekar, and U. R. Karpuzcu. 2019. Spintronic in-memory pattern matching using computational RAM (CRAM). *IEEE Journal on Exploratory Solid-State Computational Devices and Circuits* PP, 99 (2019), 1. https://doi.org/10.1109/JXCDC.2019.2951157
[9] Khanh N. Dang and Abderazek Ben Abdallah. 2019. An efficient software-hardware design framework for spiking neural network systems. In *Proceedings of the 2019 International Conference on Internet of Things, Embedded Systems, and Communications (IINTEC'19)*.
[10] Mike Davies, Narayan Srinivasa, Tsung-Han Lin, Gautham Chinya, Yongqiang Cao, Sri Harsha Choday, Georgios Dimou, et al. 2018. Loihi: A neuromorphic manycore processor with on-chip learning. *IEEE Micro* 38, 1 (2018), 82–99.
[11] N. G. De Bruijn. 1946. A combinatorial problem. *Proceedings of the Koninklijke Nederlandse Academie van Wetenschappen* 49 (1946), 758–764. https://ci.nii.ac.jp/naid/10019660672/en/
[12] Julie Dethier, Paul Nuyujukian, Chris Eliasmith, Terrence C. Stewart, Shauki A. Elasaad, Krishna V. Shenoy, and Kwabena A. Boahen. 2011. A brain-machine interface operating with a real-time spiking neural network control algorithm. In *Advances in Neural Information Processing Systems*. 2213–2221.
[13] Peter U. Diehl and Matthew Cook. 2015. Unsupervised learning of digit recognition using spike-timing-dependent plasticity. *Frontiers in Computational Neuroscience* 9 (2015), 99.
[14] X. Dong, C. Xu, Y. Xie, and N. P. Jouppi. 2012. NVSim: A circuit-level performance, energy, and area model for emerging nonvolatile memory. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 31, 7 (July 2012), 994–1007. https://doi.org/10.1109/TCAD.2012.2185930
[15] P. Faizian, M. A. Mollah, X. Yuan, Z. Alzaid, S. Pakin, and M. Lang. 2018. Random regular graph and generalized De Bruijn graph with *k*-shortest path routing. *IEEE Transactions on Parallel and Distributed Systems* 29, 1 (Jan. 2018), 144–155. https://doi.org/10.1109/TPDS.2017.2741492
[16] Charlotte Frenkel, Martin Lefebvre, Jean-Didier Legat, and David Bol. 2018. A 0.086-mm2 12.7-pJ/SOP 64k-synapse 256-neuron online-learning digital spiking neuromorphic processor in 28-nm CMOS. *IEEE Transactions on Biomedical Circuits and Systems* 13, 1 (2018), 145–158.
[17] Steve B. Furber, Francesco Galluppi, Steve Temple, and Luis A. Plana. 2014. The Spinnaker project. *Proceedings of the IEEE* 102, 5 (2014), 652–665.
[18] Saransh Gupta, Mohsen Imani, and Tajana Rosing. 2019. Exploring processing in-memory for different technologies. In *Proceedings of the 2019 Great Lakes Symposium on VLSI*. 201–206.
[19] Mohammad Hosseinabady, Mohammad Reza Kakoee, Jimson Mathew, and Dhiraj K. Pradhan. 2008. De Bruijn graph as a low latency scalable architecture for energy efficient massive NoCs. In *Proceedings of the 2008 Conference on Design, Automation, and Test in Europe*. IEEE, Los Alamitos, CA, 1370–1373.
[20] Guenole Jan, Luc Thomas, Son Le, Yuan-Jen Lee, Huanlong Liu, Jian Zhu, Ru-Ying Tong, et al. 2014. Demonstration of fully functional 8Mb perpendicular STT-MRAM chips with sub-5ns writing for non-volatile embedded memories. In *Proceedings of the 2014 Symposium on VLSI Technology (VLSI-Technology'14): Digest of Technical Papers*. IEEE, Los Alamitos, CA, 1–2.
[21] Xin Jin, Mikel Lujan, Luis A. Plana, Sergio Davies, Steve Temple, and Steve B. Furber. 2010. Modeling spiking neural networks on SpiNNaker. *Computing in Science & Engineering* 12, 5 (2010), 91–97.
[22] Shruti R. Kulkarni, Deepak Vinayak Kadetotad, Shihui Yin, Jae-Sun Seo, and Bipin Rajendran. 2019. Neuromorphic hardware accelerator for snn inference based on SST-RAM crossbar arrays. In *Proceedings of the 2019 26th IEEE International Conference on Electronics, Circuits, and Systems (ICECS'19)*. IEEE, Los Alamitos, CA, 438–441.
[23] Shahar Kvatinsky, Dmitry Belousov, Slavik Liman, Guy Satat, Nimrod Wald, Eby G. Friedman, Avinoam Kolodny, and Uri C. Weiser. 2014. MAGIC—Memristor-aided logic. *IEEE Transactions on Circuits and Systems II: Express Briefs* 61, 11 (2014), 895–899.

[24] Naveen Murali G, F. Lalchhandama, Kamalika Datta, and Indranil Sengupta. 2018. Modelling and simulation of non-ideal MAGIC NOR gates on memristor crossbar. In *Proceedings of the 2018 8th International Symposium on Embedded Computing and System Design (ISED'18)*. IEEE, Los Alamitos, CA, 124–128.

[25] Dayeol Lee, Gwangmu Lee, Dongup Kwon, Sunghwa Lee, Youngsok Kim, and Jangwoo Kim. 2018. Flexon: A flexible digital neuron for efficient spiking neural network simulations. In *Proceedings of the 2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA'18)*. IEEE, Los Alamitos, CA, 275–288.

[26] Roberto Lent, Frederico A. C. Azevedo, Carlos H. Andrade-Moraes, and Ana V. O. Pinto. 2012. How many neurons do you have? Some dogmas of quantitative neuroscience under revision. *European Journal of Neuroscience* 35, 1 (2012), 1–9.

[27] Andrew Lines, Prasad Joshi, Ruokun Liu, Steve McCoy, Jonathan Tse, Yi-Hsin Weng, and Mike Davies. 2018. Loihi asynchronous neuromorphic research chip. In *Proceedings of the 2018 24th IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC'18)*. IEEE, Los Alamitos, CA, 32–33.

[28] Guoping Liu and Kyungsook Y. Lee. 1993. Optimal routing algorithms for generalized De Bruijn digraphs. In *Proceedings of the 1993 International Conference on Parallel Processing (ICPP'93)*, Vol. 3. IEEE, Los Alamitos, CA, 167–174.

[29] Henry Markram. 2012. The human brain project. *Scientific American* 306, 6 (2012), 50–55.

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

[31] Surya Narayanan, Karl Taht, Rajeev Balasubramonian, Edouard Giacomin, and Pierre-Emmanuel Gaillardon. 2020. SpinalFlow: An architecture and dataflow tailored for spiking neural networks. In *Proceedings of the 2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA'20)*. IEEE, Los Alamitos, CA, 349–362.

[32] D. E. Nikonov and I. A. Young. 2019. Benchmarking delay and energy of neural inference circuits. *IEEE Journal on Exploratory Solid-State Computational Devices and Circuits* 5, 2 (2019), 75–84.

[33] Hiroki Noguchi, Kazutaka Ikegami, Keiichi Kushida, Keiko Abe, Shogo Itai, Satoshi Takaya, Naoharu Shimomura, et al. 2015. 7.5 A 3.3 ns-access-time 71.2 μW/MHz 1Mb embedded STT-MRAM using physically eliminated read-disturb scheme and normally-off memory architecture. In *Proceedings of the 2015 IEEE International Solid-State Circuits Conference (ISSCC'15): Digest of Technical Papers*. IEEE, Los Alamitos, CA, 1–3.

[34] Eustace Painkras, Luis A. Plana, Jim Garside, Steve Temple, Francesco Galluppi, Cameron Patterson, David R. Lester, Andrew D. Brown, and Steve B. Furber. 2013. SpiNNaker: A 1-W 18-core system-on-chip for massively-parallel neural network simulation. *IEEE Journal of Solid-State Circuits* 48, 8 (2013), 1943–1953.

[35] Yu Pan, Peng Ouyang, Yinglin Zhao, Wang Kang, Shouyi Yin, Youguang Zhang, Weisheng Zhao, and Shaojun Wei. 2018. A multilevel cell STT-MRAM-based computing in-memory accelerator for binary convolutional neural network. *IEEE Transactions on Magnetics* 54, 11 (2018), 1–5.

[36] Dayane Reis, Michael Niemier, and X. Sharon Hu. 2018. Computing in memory with FeFETs. In *Proceedings of the International Symposium on Low Power Electronics and Design*. 1–6.

[37] Salonik Resch, S. Karen Khatamifard, Zamshed I. Chowdhury, Masoud Zabihi, Zhengyang Zhao, Husrev Cilasun, Jian-Ping Wang, Sachin S. Sapatnekar, and Ulya R. Karpuzcu. 2020. MOUSE: Inference in non-volatile memory for energy harvesting applications. In *Proceedings of the 2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO'20)*. IEEE, Los Alamitos, CA, 400–414.

[38] Salonik Resch, S. Karen Khatamifard, Zamshed Iqbal Chowdhury, Masoud Zabihi, Zhengyang Zhao, Jian-Ping Wang, Sachin S. Sapatnekar, and Ulya R. Karpuzcu. 2019. PIMBALL: Binary neural networks in spintronic memory. *ACM Transactions on Architecture and Code Optimization* 16, 4 (Oct. 2019), Article 41, 26 pages. https://doi.org/10.1145/3357250

[39] Reza Sabbaghi-Nadooshan, Mehdi Modarressi, and Hamid Sarbazi-Azad. 2008. The 2D DBM: An attractive alternative to the simple 2D mesh topology for on-chip networks. In *Proceedings of the 2008 IEEE International Conference on Computer Design*. IEEE, Los Alamitos, CA, 486–490.

[40] Daisuke Saida, Saori Kashiwada, Megumi Yakabe, Tadaomi Daibou, Naoki Hase, Miyoshi Fukumoto, Shinji Miwa, et al. 2016. Sub-3 ns pulse with sub-100 μA switching of 1x–2x nm perpendicular MTJ for high-performance embedded STT-MRAM towards sub-20 nm CMOS. In *Proceedings of the 2016 IEEE Symposium on VLSI Technology*. IEEE, Los Alamitos, CA, 1–2.

[41] S. Schmitt, J. Klaehn, G. Bellec, A. Gruebl, M. Guettler, A. Hartel, S. Hartmann, et al. 2017. Neuromorphic hardware in the loop: Training a deep spiking network on the BrainScaleS wafer-scale system. In *Proceedings of the 2017 International Joint Conference on Neural Networks (IJCNN'17)*. 2227–2234. https://doi.org/10.1109/IJCNN.2017.7966125

[42] Abhronil Sengupta, Aparajita Banerjee, and Kaushik Roy. 2016. Hybrid spintronic-CMOS spiking neural network with on-chip learning: Devices, circuits, and systems. *Physical Review Applied* 6, 6 (2016), 064003.

[43] R. Singleton. 1967. A method for computing the fast Fourier transform with auxiliary memory and limited high-speed storage. *IEEE Transactions on Audio and Electroacoustics* 15, 2 (1967), 91–98.

[44] Gopalakrishnan Srinivasan, Priyadarshini Panda, and Kaushik Roy. 2018. STDP-based unsupervised feature learning using convolution-over-time in spiking neural networks for energy-efficient neuromorphic computing. *ACM Journal on Emerging Technologies in Computing Systems* 14, 4 (2018), 1–12.

[45] Terrence Stewart, Feng-Xuan Choo, and Chris Eliasmith. 2012. Spaun: A perception-cognition-action model using spiking neurons. In *Proceedings of the Annual Meeting of the Cognitive Science Society*, Vol. 34.

[46] Evangelos Stromatias, Miguel Soto, Teresa Serrano-Gotarredona, and Bernabé Linares-Barranco. 2017. An event-driven classifier for spiking neural networks fed with synthetic or dynamic vision sensor data. *Frontiers in Neuroscience* 11 (2017), 350.

[47] E. I. Vatajelu and L. Anghel. 2017. Fully-connected single-layer STT-MTJ-based spiking neural network under process variability. In *Proceedings of the 2017 IEEE/ACM International Symposium on Nanoscale Architectures (NANOARCH'17)*. 21–26. https://doi.org/10.1109/NANOARCH.2017.8053727

[48] Jian-Ping Wang and Jonathan D. Harms. 2015. General structure for computational random access memory (CRAM). US Patent 9,224,447.

[49] Jian-Ping Wang, Sachin S. Sapatnekar, Chris H. Kim, Paul Crowell, Steve Koester, Supriyo Datta, Kaushik Roy, et al. 2017. A pathway to enable exponential scaling for the beyond-CMOS era. In *Proceedings of the 54th Annual Design Automation Conference*. 1–6.

[50] Kezhou Yang, Akul Malhotra, Sen Lu, and Abhronil Sengupta. 2020. All-spin Bayesian neural networks. *IEEE Transactions on Electron Devices* 67, 3 (2020), 1340–1347.

[51] Shihui Yin, Deepak Kadetotad, Bonan Yan, Chang Song, Yiran Chen, Chaitali Chakrabarti, and Jae-Sun Seo. 2017. Low-power neuromorphic speech recognition engine with coarse-grain sparsity. In *Proceedings of the 2017 22nd Asia and South Pacific Design Automation Conference (ASP-DAC'17)*. IEEE, Los Alamitos, CA, 111–114.

[52] M. Zabihi, Z. I. Chowdhury, Z. Zhao, U. R. Karpuzcu, J. Wang, and S. S. Sapatnekar. 2019. In-memory processing on the spintronic CRAM: From hardware design to application mapping. *IEEE Transactions on Computers* 68, 8 (Aug. 2019), 1159–1173. https://doi.org/10.1109/TC.2018.2858251

[53] M. Zabihi, Z. Zhao, D. Mahendra, Z. I. Chowdhury, S. Resch, T. Peterson, U. R. Karpuzcu, J. Wang, and S. S. Sapatnekar. 2019. Using Spin-Hall MTJs to build an energy-efficient in-memory computation platform. In *Proceedings of the 20th International Symposium on Quality Electronic Design (ISQED'20)*. 52–57. https://doi.org/10.1109/ISQED.2019.8697377

[54] Deming Zhang, Lang Zeng, Youguang Zhang, Weisheng Zhao, and Jacques Olivier Klein. 2016. Stochastic spintronic device based synapses and spiking neurons for neuromorphic computation. In *Proceedings of the 2016 IEEE/ACM International Symposium on Nanoscale Architectures (NANOARCH'16)*. IEEE, Los Alamitos, CA, 173–178.

Received November 2020; revised May 2021; accepted July 2021