frontiers in Neuroscience

ORIGINAL RESEARCH
published: 29 August 2018
doi: 10.3389/fnins.2018.00583

Neural and Synaptic Array Transceiver: A Brain-Inspired Computing Framework for Embedded Learning

Georgios Detorakis[1], Sadique Sheik[2], Charles Augustine[3], Somnath Paul[3], Bruno U. Pedroni[4], Nikil Dutt[1,5], Jeffrey Krichmar[1,5], Gert Cauwenberghs[4] and Emre Neftci[1,5]**

[1] Department of Cognitive Sciences, University of California, Irvine, Irvine, CA, United States, [2] Biocircuits Institute, University of California, San Diego, La Jolla, CA, United States, [3] Intel Corporation-Circuit Research Lab, Hillsboro, OR, United States, [4] Department of Bioengineering and Institute for Neural Computation, University of California, San Diego, La Jolla, CA, United States, [5] Department of Computer Science, University of California, Irvine, Irvine, CA, United States

OPEN ACCESS

Edited by:
Jonathan C. Tapson,
Western Sydney University, Australia

Reviewed by:
Mostafa Rahimi Azghadi,
James Cook University, Australia
Damien Querlioz,
Centre National de la Recherche Scientifique (CNRS), France

***Correspondence:**
Georgios Detorakis
gdetorak@uci.edu
Emre Neftci
eneftci@uci.edu

Specialty section:
This article was submitted to Neuromorphic Engineering, a section of the journal Frontiers in Neuroscience

Received: 20 March 2018
Accepted: 03 August 2018
Published: 29 August 2018

Citation:
Detorakis G, Sheik S, Augustine C, Paul S, Pedroni BU, Dutt N, Krichmar J, Cauwenberghs G and Neftci E (2018) Neural and Synaptic Array Transceiver: A Brain-Inspired Computing Framework for Embedded Learning. Front. Neurosci. 12:583. doi: 10.3389/fnins.2018.00583

Embedded, continual learning for autonomous and adaptive behavior is a key application of neuromorphic hardware. However, neuromorphic implementations of embedded learning at large scales that are both flexible and efficient have been hindered by a lack of a suitable algorithmic framework. As a result, most neuromorphic hardware are trained off-line on large clusters of dedicated processors or GPUs and transferred *post hoc* to the device. We address this by introducing the neural and synaptic array transceiver (NSAT), a neuromorphic computational framework facilitating flexible and efficient embedded learning by matching algorithmic requirements and neural and synaptic dynamics. NSAT supports event-driven supervised, unsupervised and reinforcement learning algorithms including deep learning. We demonstrate the NSAT in a wide range of tasks, including the simulation of Mihalas-Niebur neuron, dynamic neural fields, event-driven random back-propagation for event-based deep learning, event-based contrastive divergence for unsupervised learning, and voltage-based learning rules for sequence learning. We anticipate that this contribution will establish the foundation for a new generation of devices enabling adaptive mobile systems, wearable devices, and robots with data-driven autonomy.

Keywords: Neuromorphic computing, neuromorphic algorithms, three-factor learning, on-line learning, event-based computing, spiking neural networks

1. INTRODUCTION

Brain-inspired computing paradigms can lead to massively distributed technologies that compute on extremely tight power budgets, while being robust to ambiguities in real-world sensory information and component failures. To devise such technology, *neuromorphic* electronic systems strive to mimic key building blocks of biological neural networks and dynamics (Mead, 1989) in custom digital (Furber et al., 2014; Merolla et al., 2014) or mixed signal (Schemmel et al., 2010; Benjamin et al., 2014; Qiao et al., 2015) CMOS technologies.

Recent progress has significantly advanced the systematic synthesis of dynamical systems onto neural substrates and their neuromorphic counterparts. For instance, the configuration of spiking neural networks for inference tasks have been solved using frameworks such as the neural engineering framework (Eliasmith and Anderson, 2004) and STICK (Lagorce and Benosman, 2015), direct mapping of pre-designed (Neftci et al., 2013) or pre-trained neural networks (Cao et al., 2015).

Many of these approaches were successfully ported to neuromorphic hardware (Neftci et al., 2013; Qiao et al., 2015; Esser et al., 2016). While these solutions are promising from an energetic point of view in inference tasks, they heavily rely on computers (GPU or CPU) for their configuration and largely abandon adaptive and autonomous behavior capabilities in the presence of intrinsic and extrinsic variations. These critical features can be introduced through embedded synaptic plasticity and learning "on-the-fly." However, learning using data streaming to the neuromorphic device presents significant challenges. One challenge is technological: synaptic plasticity requires high memory bandwidth, but the realization of adequate high density memory co-located with the neuron is costly using current technologies. While emerging memory technologies are poised to solve this problem, the solutions remain difficult to control and lack precision. Another challenge is algorithmic: the co-location of memory with the neuron leads to significant algorithmic challenge: state-of-the-art algorithms in machine learning rely on information that is temporally and spatially global when implemented on a neural substrate. Finally, the hardware implementation of learning involves a hard commitment to the plasticity dynamics, but doing so in a way that is both hardware-friendly and capable of learning a wide range of tasks is a significant modeling challenge. Our recent work in neuromorphic algorithms demonstrated that most algorithmic challenges can be solved (Eliasmith et al., 2012; Lagorce and Benosman, 2015), and can potentially result in learning systems that require a thousandfold less power than mainstream technologies (Neftci et al., 2016, 2017b; Mostafa, 2017), while matching or surpassing the accuracy of dedicated machine learning accelerators, and operating on-line. In addition, neuromorphic learning-enabled devices are expected to have similar energy per operation figures with learning-enabled artificial neural networks, such as binary neural networks (Neftci, 2018). Furthermore, it has been shown that neural networks with binary activations are a class of spiking neural networks (without states or dynamics) (Neftci, 2018), implying that the proposed framework is capable of implementing neural networks without binary activations as well.

One outstanding question is whether one can formulate a general event-based learning rule that is general and capable of learning a wide range of tasks while being efficiently realizable using existing memory technologies.

This article presents one such system, called Neural and Synaptic Array Transceiver (NSAT), and demonstrates proof-of-concept learning applications. Extreme efficiency in data-driven autonomy hinges on the establishment of (i) energy-efficient computational building blocks and (ii) algorithms that build on these blocks. NSAT is a spiking neural network architecture designed on these assumptions, using neural building blocks that are constructed from algorithmic principles and an event-based architecture that emphasizes locally dense and globally sparse communication (Park et al., 2017).

To achieve extreme efficiency in dedicated implementations, the NSAT framework consists of neural cores that take advantage of tractable linear neural model dynamics, multiplier-less design, fixed-width representation and event-driven communication, while being able to simulate a wide range of neural and plasticity dynamics. Each NSAT core is composed of state components that can be flexibly coupled to form multi-compartment generalized integrate-and-fire neurons, allowing the implementation of several existing neural models (**Figure 1**). The state components forming the neuron can be interpreted as somatic potential, dendritic potential, synaptic currents, neuromodulator concentration or calcium currents, depending on its interactions with other state components or pre-synaptic neurons. The communication between cores and event-driven sensors is routed via inter-core spike events.

While several neuromorphic VLSI circuits for synaptic learning exist (Arthur and Boahen, 2006; Pfeil et al., 2012; Azghadi et al., 2015; Qiao et al., 2015), our framework is novel in that it is equipped with a flexible and scalable event-based plasticity rule that is tightly guided by algorithmic considerations and matched to the neuron model. Scalability is achieved using only forward lookup access of the synaptic connectivity table (Pedroni et al., 2016), permitting scalable, memory-efficient implementation compared to other implementations requiring reverse table lookups or memory-intensive architectures such as crossbar arrays. Flexibility in the learning dynamics is achieved using a reconfigurable event-based learning dynamics compatible with three-factor rules (Urbanczik and Senn, 2014), consistent with other established plasticity dynamics such as STDP (Bi and Poo, 1998; Markram et al., 2012), membrane-voltage based rules (Clopath et al., 2010), calcium based dynamics (Shouval et al., 2002; Graupner and Brunel, 2012), and reinforcement learning (Florian, 2007).

NSAT is a framework intended to guide the design of an optimized digital architecture, which we outline in the sections 2 and 3. To set sail toward hardware implementations of the NSAT framework and assist algorithmic co-design efforts, we wrote cNSAT, a multi-thread software simulator of the NSAT framework that is behaviorally accurate with respect to the envisioned optimized digital hardware implementation. Using the cNSAT simulator, we show that learning in digital NSAT requires fewer SynOps compared to MACs in equivalent digital hardware, suggesting that a custom hardware implementation of NSAT can be more efficient than mainstream computing technologies by a factor equal to the J/MAC to J/Synop ratio. Furthermore, to verify the viability of a digital implementation, we validated NSAT on a Field Programmable Gate Array (FPGA).

This article is organized as follows: In the section 2 we describe the neuron model and its mathematical equations. We present the NSAT architecture and software simulator (publicly available under GPLv3 license). In section 3 we show that the neuron model can simulate the Mihalas-Niebur neuron and thus demonstrate a rich repertoire of spike behaviors and neural

field models. Then, we demonstrate that the NSAT framework supports a type of gradient back-propagation in deep networks, unsupervised learning in spike-based Restricted Boltzmann Machines (RBMs), and unsupervised learning of sequences using a competitive learning.

2. MATERIALS AND METHODS

In this section we introduce the mathematical description of the NSAT framework and the details regarding its architecture and the main information processing flow. NSAT software implementation (cNSAT) details are given in the **Appendices**.

2.1. Leaky Integrate-and-Fire Neurons as Dynamical Systems

We start the discussion with the leaky integrate-and-fire neuron (LIF) model, given by the following equations

$$\tau_m \frac{\mathrm{d}}{\mathrm{d}t} V(t) = -V(t) + RI(t). \tag{1a}$$

$$\text{If } V(t) \geq \theta \text{ then } V(t) = V_r \text{ and } s = 1, \tag{1b}$$

where $V(t)$ is the neuron's membrane potential, τ_m is the membrane time constant, R is the membrane resistance and $I(t)$ is the driving current, which can be comprised of external current I_{ext} and/or synaptic one I_{syn}. When the membrane potential is greater or equal to a threshold value (θ), the neuron fires a spike and the membrane potential value at that time step is set to a reset value V_r (resting potential).

The dynamical properties of LIF neurons can be extended with synaptic dynamics or other internal currents such as calcium channels, potassium channels and other biophysical variables. For instance, the concentration of some neurotransmitter or ion, $U(t)$, can be captured by the linear dynamics:

$$\tau_U \frac{\mathrm{d}}{\mathrm{d}t} U(t) = -U(t) + \sum_k \delta(t - t_k), \tag{2}$$

where $U(t)$ is the concentration within the neuron cell reflecting for example calcium concentration, although the biological interpretation is not indispensable for the NSAT framework. The term $\sum_k \delta(t - t_k)$ indicates the pre-synaptic incoming spikes to the current post-synaptic neuron (δ is the Dirac function[1]). If we rewrite the summation term as $S(t) = \sum_k \delta(t - t_k)$ then the dynamics become the linear system:

$$\begin{bmatrix} \dot{V}(t) \\ \dot{U}(t) \end{bmatrix} = \begin{bmatrix} -\frac{1}{\tau_m} & 0 \\ 0 & -\frac{1}{\tau_U} \end{bmatrix} \cdot \begin{bmatrix} V(t) \\ U(t) \end{bmatrix} + \begin{bmatrix} \frac{RI(t)}{\tau_m} \\ \frac{S(t)}{\tau_U} \end{bmatrix}. \tag{3}$$

A generalization of such linear dynamics to N dimensions can be written in the following vector notation:

$$\frac{\mathrm{d}}{\mathrm{d}t} \mathbf{x}(t) = \mathbf{A}\mathbf{x}(t) + \mathbf{Z}(t), \tag{4}$$

[1] $\delta(t) = \infty$, if $t = 0$ otherwise $\delta(t) = 0$. In the discrete version we have $\delta(t) = 1$, if $t = 0$ otherwise $\delta(t) = 0$.

where the temporal evolution of state $\mathbf{x}(t) = (x_0(t), x_1(t), x_2(t), \cdots, x_N(t))$ is characterized by the solution of Equation (4). In the equation above, $\mathbf{A}$ is the state transition matrix and $\mathbf{Z}(t)$ the time-varying external inputs or commands to the system. Solutions to linear dynamical systems of Equation (4) are given by:

$$\mathbf{x}(t) = \exp(\mathbf{A}t)\mathbf{x}(t_0) + \int_{t_0}^{t} \exp(\mathbf{A}(t - \tau))\mathbf{Z}(\tau)d\tau. \tag{5}$$

Equation (5) can be computed numerically by using the Putzer algorithm (Putzer, 1966) for computing the matrix exponential. Numerical solutions of Equation (4) can be obtained using several numerical integration methods, such as the Forward Euler which is computationally simple, fast, less expensive than other Runge-Kutta methods which require more operations, and compatible with stochastic differential equations (Kloeden and Platen, 1991). Furthermore, even in the case where Equation (4) is stiff we can adjust the time-step such that the Forward Euler is stable. For these reasons, Forward Euler is a common choice for digital simulations of neural networks (Zenke and Gerstner, 2014; Davies et al., 2018).

2.2. NSAT Neuron and Synapse Model

In continuous form, the NSAT neuron consists of linear dynamics described in general by Equation (4) extended with firing thresholds, resets mechanisms and inputs $\mathbf{Z}(\mathbf{t})$ written in open form:

$$\frac{\mathrm{d}\mathbf{x}(t)}{\mathrm{d}t} = \mathbf{A}\mathbf{x}(t) + \underbrace{(\boldsymbol{\Xi}(t) \circ \mathbf{W}(t)) \cdot \mathbf{s}(t) + \boldsymbol{\eta}(t) + \mathbf{b}}_{Z(t)}. \tag{6a}$$

$$\text{If } \mathbf{x}(t) \geq \boldsymbol{\theta} \text{ then } \mathbf{x}(t) = \mathbf{X_r}, \tag{6b}$$

$$\text{If } x_0(t) \geq \theta_0 \text{ then } s_0(t) = \delta(t). \tag{6c}$$

The state components $\mathbf{x} = (x_0, ..., x_k)$ describe the dynamics of a neural compartment or variable such as membrane potential, internal currents, synaptic currents, adaptive thresholds and other biophysical variables, although a biological interpretation is not essential. $\mathbf{A}$ is the state-transition square matrix that describes the dynamics of each state component and their couplings. $\boldsymbol{\Xi}$ is a random variable drawn from a Bernoulli distribution and introduces multiplicative stochasticity to the NSAT, which is an important feature for learning (Hinton et al., 2012; Wan et al., 2013) inspired by synaptic failures (Vogelstein et al., 2002; Neftci et al., 2016). Probabilistic synapses support Poisson-like variability in the of spiking neural networks and provide a mechanism for performing highly robust probabilistic inference under noisy and ambiguous conditions (Moreno-Bote, 2014). Furthermore, stochasticity at the synaptic level accounts for optimizing the energetic efficiency of neurons (Levy and Baxter, 2002).

$\mathbf{W}$ is the synaptic strength matrix and defines the connectivity and the strength of each connectivity between neurons. $\mathbf{s}$ is a vector that takes values in $\{0, 1\}$ and registers whether the neuron

has spiked. The $\circ$ symbol defines an element-wise multiplication (or Hadamard product). $\boldsymbol{\eta}$ is the additive normal noise with zero mean and programmable variance, allowing for shifting the equilibrium of Equation (8) and decorrelating repetitive spiking patterns (Tuckwell and Jost, 2010). And finally, $\mathbf{b}$ is a constant value that is added to each state component acting as a constant input (i.e., current injection from a neuroscience point of view or bias from a machine learning point of view). When a component x_i crosses its threshold value (θ_i) then it is subject to reset to some predefined value X_{r_i}. Additionally if the zero state of a neuron ($x_0(t)$) crosses its threshold value then that neuron fires a spike as shown in Equation (6), with ($s_0(t) = \delta(t)$) and a new setting of X_{r_0}. After the neuron has spiked, the membrane potential is clamped during a programmable refractory period, during which it is not permitted to fire.

2.2.1. Event-driven Synaptic Plasticity and the NSAT Plasticity Model

Spike-Timing Dependent Plasticity (STDP) is a popular learning rule used throughout computational neuroscience models, thanks to empirical evidence and its simplicity. It is a form of Hebbian learning that modifies the synaptic strengths of connected pre- and post-synaptic neurons based on their spikes firing history in the following way (Bi and Poo, 1998; Sjöström et al., 2008): if a post-synaptic neuron generates action potential within a time interval after the pre-synaptic neuron has fired multiple spikes then the synaptic strength between these two neurons potentiates (causal update, long-term potentiation-LTP). On the other hand if the post-synaptic neuron fires multiple spikes before the pre-synaptic neuron generates action potentials within that time-interval then the synapse depotentiates (acausal update, long-term depression-LTD).

Like Hebb's rule, STDP is an unsupervised rule that depends on pre-synaptic and post-synaptic factors (here spike times), and so STDP alone is impractical for learning with reward or error signals extrinsic to the STDP neuron pairs. On the other hand, three factor rules solve this problem by adding a factor indicative of reward, error, gradients provided extrinsically or through other neural states (Clopath et al., 2010; Urbanczik and Senn, 2014). Several theoretical work underline that gradient descent on spike train distances or classification loss indeed take the form of such three factor rules (Pfister et al., 2006; Urbanczik and Senn, 2014; Neftci et al., 2017b; Zenke and Ganguli, 2017). These results also indicate that optimal gradient descent learning rules involve continuous-time dynamics. However, because continuous-time updates are prohibitive in digital hardware, one must resort to event-based learning such as STDP.

To implement multiple learning scenarios in a fully event-based fashion with minimal memory overhead, NSAT follows a modulated, index-based STDP rule. Index-based architectures are memory-efficient with realistic and practical sparse connectivities, but are challenging to implement in neuromorphic hardware because synaptic memory is typically localized at the pre-synaptic neurons, and so causal updates require reverse look-up tables or reverse search for the forward table at every spike-event. Although reverse lookups are not an issue in crossbar memories (they are compatible with the data structure associated with the crossbar), they can incur a significant memory overhead for non-dense connectivities and are not considered here.

Recent implementations of STDP in the Spinnaker hardware use dedicated synaptic plasticity cores (Galluppi et al., 2015) for implementing STDP. While this approach gives additional flexibility in the STDP learning rule, it relies on relatively large SDRAMs and more communication for its realization. Furthermore, it has the disadvantage of segregating synaptic memory from neural states, which as argued above, may contain important information for learning.

To mitigate these problems, NSAT uses a forward table-based, pre-synaptic event-triggered, nearest-neighbor STDP rule (Pedroni et al., 2016) coupled with the neuron dynamics. This method implements both causal and acausal weight updates using only forward lookup access of the synaptic connectivity table. A single timer variable for each neuron is sufficient to implement this rule, permitting implementation that requires only $\mathcal{O}(N)$ memory, where N is the number of neurons. The basic nearest-neighbor STDP (Sjöström et al., 2008) is recovered in the case of refractory periods greater than the STDP time window, and otherwise it closely approximates exact STDP cumulative weight updates.

This method is related to the deferred event-driven (DED) rule used in Spinnaker (Jin et al., 2010), which does not allow the pre-synaptic spike to trigger the STDP until a predetermined time limit is reached. The time that a pre-synaptic spike occurred is recorded as a time-stamp and is used in the future once the missing information from the future spikes has been made available (post-synaptic neurons have fired action potentials). Such STDP schemes are called "pre-sensitive", as STDP takes place only when a pre-synaptic neuron fires an action potential.

Similar to DED, the three-factor NSAT STDP learning rule implements a pre-sensitive scheme: The NSAT framework learning rule keeps track of the spike times using a time counter per neuron. When a pre-synaptic neuron fires then all the corresponding acausal STDP updates are triggered and the post-synaptic weights are updated based on a linear or exponential approximation STDP kernel (see Figure A2 in **Appendix A**). If the counter of the pre-synaptic neuron expires, then only the causal STDP update takes place. As long as the counter has not expired and post-synaptic neurons fire within the STDP time-window then the acausal updates are computed. If, now, a new spike from the pre-synaptic neuron is emitted then causal updates are computed.

To enable learning using extrinsic and intrinsic modulation, the NSAT three-factor learning rule is modulated by the neural state components, which enables modulation based on continuous dynamics using both local and global information. Since NSAT connectivity allows extrinsic inputs to drive its state components, modulation can be driven by extrinsic rewards, error or the dynamics of entire neural population. The mathematical formulation of the NSAT three-factor STDP

learning rule is given by:

$$\epsilon_{ij}(t) = x_m^j(t)\Big(K(t-t_i) + K(t_j - t)\Big), \tag{7a}$$

$$\frac{\mathrm{d}}{\mathrm{d}t}\mathbf{w}_{ij}(t) = \epsilon_{ij}(t)\delta_j(t), \tag{7b}$$

where $\epsilon_{ij}(t)$ is the eligibility of the synapse between the i-th pre-synaptic neuron and the j-th post-synaptic neuron update, $K(\cdot)$ is the STDP learning window, t_i is the last time that the i-th pre-synaptic neuron fired a spike, and t_j is the last time that the post-synaptic neuron fired a spike. The kernel (or learning window) $K(t-t_i)$ refers to the causal (positive) STDP update and the term $K(t_j - t)$ refers to the acausal (negative) STDP update. $x_m^j(t)$ is the m-th state component of the post-synaptic neuron that dynamically modulates the amplitude of the STDP kernel.

A remark with respect to learning rule (7) is that rate-based learning schemes can be implemented on NSAT, as NSAT neurons are compatible with firing rate neurons. To achieve this, NSAT neurons configured as integrators to read-out estimates of the firing rate, and the synaptic plasticity rule can be configured as a membrane voltage-modulated learning rule. Another example of rate-based learning is the BCM learning rule (Bienenstock et al., 1982), from which one can derive a modulated STDP rule compatible with NSAT assuming stochastic firing of the pre- and post-synaptic neurons (Izhikevich and Desai, 2003). The latter can be realized using additive or multiplicative noise.

2.3. Difference Equations of NSAT (Quantized) Framework

The NSAT software simulator consists of discrete-time versions of the above equations, based on fixed point arithmetics without any multiplications. The continuous-time dynamics of NSAT described by Equations (6) and (7) are rewritten here in a discrete (quantized) form:

$$\begin{aligned}\mathbf{x}[t+1] = \mathbf{x}[t] + \mathbf{A}\diamondsuit\mathbf{x}[t] + \\ + (\boldsymbol{\Xi}[t] \circ \mathbf{W}[t]) \cdot \mathbf{s}[t] \\ + \boldsymbol{\eta}[t] + \mathbf{b}.\end{aligned} \tag{8a}$$

$$\text{If } \mathbf{x}[t+1] \geq \boldsymbol{\theta} \text{ then } \mathbf{x}[t+1] \leftarrow \mathbf{X_r}. \tag{8b}$$

$$\text{If } x_0[t+1] \geq \theta_0 \text{ then } s_0[t+1] \leftarrow 1, \tag{8c}$$

where the entries of matrices $\mathbf{A}$ and $\mathbf{b}$ are integer constants, and $\boldsymbol{\eta}$ is the variance of the additive noise. More details regarding the parameters are provided in **Appendix A** and in the **Supplementary Information**.

The binary operator $\mathcal{D}(\cdot,\cdot) : GF(2^n) \rightarrow GF(2^n)$ (or $\diamondsuit$), where $n = 4$ or $n = 5$, is defined as

$$\mathcal{D}(a,x) = a\diamondsuit x = \begin{cases} sign(-a \diamond x) & \text{if } d(a,x) \neq 0 \text{ and } a = 0 \\ a & \text{otherwise,} \end{cases} \tag{9}$$

(described also by Algorithm 2 in **Appendix A**) plays the role of a multiplication implemented with bit shift operations. In particular, it ensures that all state components leak toward the resting state in the absence of external input (current).

The binary operator $d(\cdot,\cdot) : GF(2^n) \rightarrow GF(2^n)$ (or $\diamond$), where $n = 4$ or $n = 5$, defines a custom bit shift. It performs a multiplication by power of two using only bitwise operations, and it is defined as

$$d(a,x) = a \diamond x = \begin{cases} x << a & \text{if } a \geq 0 \\ sign(x)(|x| >> -a) & \text{otherwise} \end{cases} \tag{10}$$

(see also Algorithm 3 in **Appendix A**).

The reason for using $\diamond$ rather than left and right bit shifting is because integers stored using a two's complement representation have the property that right shifting by a values such that $x > -2^{a'}, \forall a' < a$ is -1, whereas 0 is expected in the case of a multiplication by 2^{-a}. The $\diamond$ operator corrects this problem by modifying the bit shift operation such that $-2^{a'} \diamond a = 0, \forall a' < a$. In addition, such multiplications by powers of 2 have the advantage that the parameters are stored on a logarithmic scale, such that fewer bits are required to store parameters. For example, $-(-3\diamondsuit x_0)$ is the NSAT equivalent of $-2^{-3}x_0[t]$. A logarithmic scale for the parameters is suitable since solutions to the equations consist of sums of exponentials of these parameters (5).

The learning rule given by Equation (7)is also discretized:

$$\epsilon_{ij}[t] = x_j^m[t] \diamond \Big(K[t-t_i] + K[t_j - t]\Big), \tag{11a}$$

$$w_{ij}[t+1] = \mathrm{Clip}(w_{ij}[t] + \underbrace{\epsilon_{ij}[t]s_j[t]}_{\Delta w_{ij}}). \tag{11b}$$

Where $\mathrm{Clip}(x) = \max\{w_{\min}, \min\{x, w_{\max}\}\}$ clips its first argument to within the range $[w_{\min}, w_{\max}]$ dictated by the fixed point representation of the synaptic weights at every time step.

In addition, the weight updates can be randomized using a discretized version of randomized rounding (Muller and Indiveri, 2015), which interprets the r least significant bits of Δw as a probability, as follows:

$$\Delta w_{ij}^r = (\Delta w \gg r) + \begin{cases} 1 & \text{if random}(0,1) < p \\ 0 & \text{otherwise.} \end{cases}, \tag{12}$$

where p is the number formed by the r least significant bits of Δw_{ij}.

Figure 1A shows an example of the NSAT learning rule (Equation 11). In this example, each neuron consisted of 4 components. The first and second component correspond to classical leaky integrate and fire dynamics with current-based synapses. The third state component driven externally to modulate the STDP update. As a result most weights updates concentrate around high modulation states.

2.4. The NSAT Architecture

Figures 1C,D, **2** illustrate the NSAT architecture consisting of multiple interconnected cores (or threads each simulating one NSAT core). Only addresses of a neuron's spike are transmitted in inter- and intra-thread communication. At every simulation time step (or tick) each thread runs independently, executing NSAT dynamics in two stages. In the first stage each thread integrates

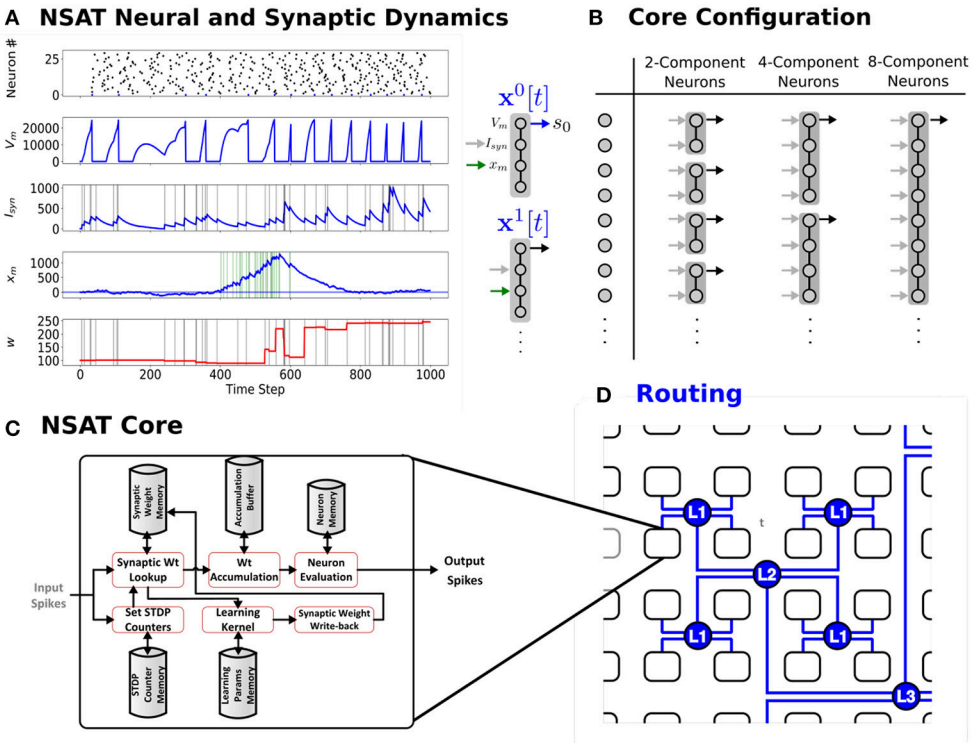

FIGURE 1 | The Neural and Synaptic Array Transceiver (NSAT). **(A)** Sample run externally modulated STDP, showing raster plot of 25 neurons and detailed temporal dynamics of the four components of the first neuron's (neuron number 0) state. The first component represents the membrane potential (V_m), the second component represents the synaptic state (I_{syn}), while the third is the plasticity modulation (x_m). The latter state is driven externally by a spike train that is active between time steps 400 and 600, and white noise of constant amplitude. **(B)** NSAT Neurons consist of compartments that can be coupled to trading off the number of neurons vs. neuron complexity (number of compartments). **(C–D)** NSAT information flow and envisioned layout of the NSAT cores using Hierarchical Address-Event Routing (HiAER) (Park et al., 2017) for scalable and expandable neural event communication with reconfigurable long-range synaptic connectivity.

the neural dynamics of its neurons based on Equation (8) without accumulating the synaptic inputs on the neuron state. At that stage all the threads are synchronized (thread barrier) and then they detect new spike-events and transmit them accordingly to their destinations. All the detected inter- and intra-core spike events at time t are made available to the next time step $(t + 1)$. After the distribution of all the spikes (intra- and inter-core) all the threads are synchronized once again before proceed to the next stage.

In the second stage, the detected spike events (including the external ones) are accumulated onto the neural states components x_i according to $(\boldsymbol{\Xi}[t] \circ \mathbf{W}[t]) \cdot \mathbf{s}[t]$. Synaptic weights are multiplied with a predefined constant (implemented as a bit shift operation) to trade off precision and range limitations imposed by fixed point integer arithmetic. Our previous work (Neftci et al., 2017a) and Drop Connect (Wan et al., 2013) showed that a probability of $\frac{1}{2}$ work best as multiplication constant. Consequently, we use a blank-out factor as close as possible to $\frac{1}{2}$ throughout our simulations. The blank-out factor does not directly affect the required weight precision. When the learning is enabled, Equation (11)is computed. First, threads compute the causal and then the acausal part of the STDP learning curve. After learning, the STDP counters of neurons that have spiked are set to their new values (either the last time that a neuron spiked, or a neuron clock starts ticking until expiration). The final steps in the second stage perform update of modulator dynamics (x_i^m) and reset of the neuron state components that spiked. The modulator state component (x_m) adjusts the amplitude of the STDP function as described in Equation (11). Algorithm 1 provides in pseudo-code the flow of the NSAT operations. Furthermore, in the **Appendix A** we provide more details regarding the data structures, simulation details and in the **Supplementary Material** the parameters for all of our results presented in the next section.

2.5. NSAT Hardware Architecture

A synchronous digital architecture with the same functionality as the cNSAT was written in Verilog and its functionality was validated by emulating the same on FPGA. This section provides an overview of the architecture and provides an idea on the potential power savings that result from optimized NSAT data-structure and functions.

Figure 3A shows the top level organization of the NSAT architecture. We refer to this as a single NSAT tile. Note that such tiled architecture has been proposed earlier in the context of neuromorphic hardware with multi-tile communication enabled

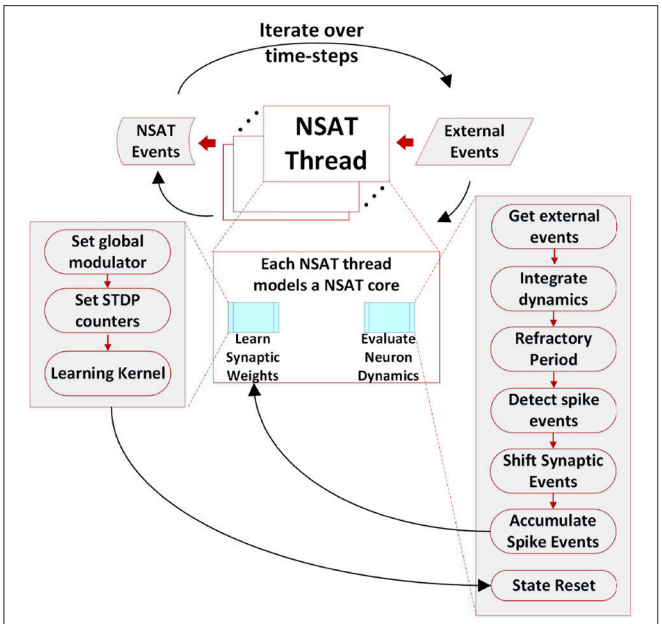

FIGURE 2 | The NSAT Architecture. Multiple NSAT threads receive external events and generate event responses in two stages. Each thread consists of two major instruction sets (cyan boxes), one for evaluating neuron dynamics and the other for implementing an event-based STDP learning rule. The instructions are shown in the two large gray boxes. Arrows indicate the flow of the information within an NSAT thread during simulation.

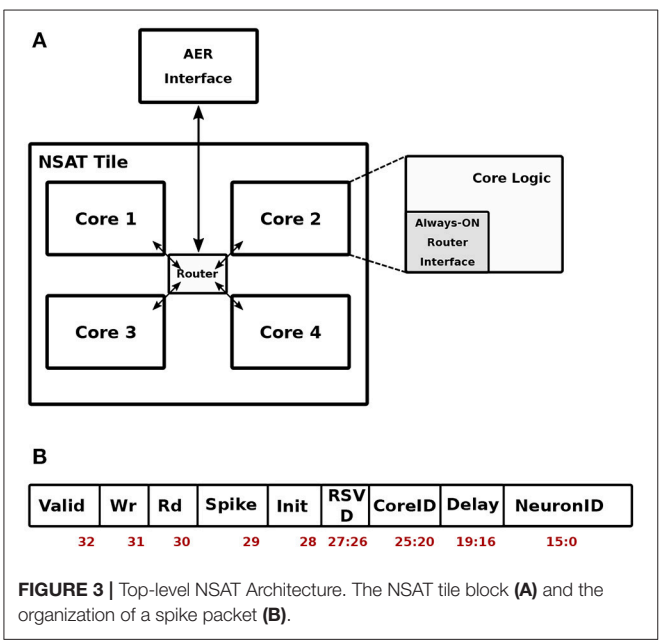

FIGURE 3 | Top-level NSAT Architecture. The NSAT tile block **(A)** and the organization of a spike packet **(B)**.

through a hierarchical AER communication fabric (Park et al., 2017). The contribution of this work therefore focused on specifics of the digital implementation inside each tile. Each tile is hierarchically organized into four NSAT cores, which communicate via a packet-switched router conforming to the Address Event Representation (AER) protocol (Lazzaro et al., 1993). The AER packets are routed from/to the primary AER interface at each tile to each core following a wormhole routing strategy implemented in the router. The digital implementation of the router is inspired from Vangal et al. (2007), which is adopted to work on single-flit packets. These packets have the format as shown in **Figure 3B**.

As shown in **Figure 3B**, the packet is functionally diverse. It can act as (i) a memory write packet—to initialize the weight memory and the configuration register inside each core. The Neuron and Delay fields then carry the address and the payload for writing to the memory location, (ii) a memory read packet—to read from a memory location in a given core. The payload is then the memory address. In response to a memory read packet, the core responds by sending out the data being read from the memory location. (iii) A spike packet, representing a spike from another core or tile, carries the destination core and neuron

Algorithm 1 Algorithmic (software) NSAT implementation (see text for more details).

Require: Synaptic Weights, Parameters, Learning Parameters
Ensure: Spike events, States, Synaptic Weights
for all p in {Threads} **do**
for $t \leftarrow 1 \ldots t_{\text{final}}$ **do**
for all i in {Neurons} **do**
spike_list $\leftarrow$ external_events
$\mathbf{x}^i[t] \leftarrow \mathbf{x}^i[t] + \mathbf{A} \circledcirc \mathbf{x}^i[t] + \boldsymbol{\eta}[t] + \mathbf{b}$
if ref_period$^i > 0$ **then**
ref_period$^i \leftarrow$ ref_period$^i - 1$
$\mathbf{x}^i[t] \leftarrow \mathbf{X}^i_{\text{reset}}$
end if
if Spike is Enabled **then**
if $\mathbf{x}^i[t] \geq \boldsymbol{\theta}^i$ **then**
spike_list $\rightarrow$ id $\leftarrow i$
spike_list $\rightarrow$ ts $\leftarrow t$
$s^i[t] = 1$
end if
else if Adaptive θ is Enabled **then**
if $x^i_0[t] \geq x^i_1[t]$ **then**
spike_list $\rightarrow$ id $\leftarrow i$
spike_list $\rightarrow$ ts $\leftarrow t$
$s^i[t] = 1$
end if
end if
$\mathbf{x}^i[t] \leftarrow \mathbf{x}^i[t](\boldsymbol{\Xi}[t] \circ \mathbf{W}[t]) \cdot \mathbf{s}^i[t]$
$\mathbf{W}[t] \leftarrow \mathbf{W}[t] \circ G^i$
if Learning is Enabled **then**
Compute Equations (11) and (12)
end if
if $s^i[t] == 1$ and Reset is Enabled **then**
$\mathbf{x}^i[t] \leftarrow \mathbf{X}^i_{\text{reset}}$
ref_period$^i \leftarrow \mathbf{X}^i_{\text{ref}}$
end if
end for
end for
end for

addresses along with an axonal delay information. This conforms to the general AER definition where an event is tagged with a destination address. Each core is logically divided into an always-ON router interface and the core logic which is active only at the arrival of an input spike or at the beginning of each time-stamp. If no spike is present in a given time-stamp, only the neuron dynamics are evaluated and the core logic is thereafter put in a low-power retention mode.

2.5.1. NSAT Core Architecture

Figure 4 shows the detailed breakdown of each core. The Always-ON (AON) router interface consists of two channels—packetizer and de-packetizer corresponding to the outgoing and incoming streams of packets. The AON module also generates a gated clock for the rest of the NSAT core. In absence of input activity, the core clock is gated to prevent dynamic power dissipation. Each NSAT core contains logic and memory to map 512 8-component state neurons which can also be reconfigured as 4096 1-component state neuron. Following are the primary components of the NSAT core.

The synaptic weight memory is physically the largest of the NSAT core modules, with 128 KB of synaptic weight storage. Input to this module are spikes and output are its fanout weights. In addition to the synaptic storage, decoder logic selects the appropriate memory array for weight retrieval. In order to store weights for sparse fanouts, we have implemented a compressed storage scheme for the weights. The weight memory is divided into a pointer array which stores the pointers to the weight data array, while the weight data array stores the actual weights. The compression scheme we used is Run Length Encoding (RLE) which skips zero weights corresponding to missing connections. The pointer memory is also useful for storing pointers for weight parameters that are shared across multiple neurons (e.g., in convolutional filters). A decompression engine following the weight data array decompresses the weights before sending them out for accumulation to destination neurons. The organization is illustrated in **Figure 4B**.

The fanout weights obtained are accumulated on the destination neurons in the weight accumulation module. While, one memory array is used to store and update the partial sums from weight accumulation in the current time-stamp, another array is used to feed the accumulated weights from the previous time-stamp to the neuron evaluation unit. This approach decouples weight look-up and accumulation from neuron evaluation. The total memory size is proportional to the number of neurons, number of state components/neuron and the number of bits per component. **Figure 4C** illustrates the organization of the weight accumulation module.

The neuron evaluation block evaluates the neuron dynamics in the NSAT core. It receives the accumulated weights from the weight accumulation block. The previous state components are stored in the neuron state memory in the neuron evaluation block. The neuron configuration parameters are also locally stored in the neuron evaluation block. The NSAT neural dynamics is implemented as a 4-stage pipeline where all the state components of a neuron are evaluated in parallel and all the neurons are evaluated in a time-multiplexed fashion. The output from the block are neuronal responses, i.e., spikes. **Figure 4D** shows the block diagram of the neuron evaluation unit.

The learning engine implements multiple learning algorithms as available in cNSAT (**Figure 4E**). This includes weight update corresponding to conventional STDP as well as the state dependent weight update. For STDP weight update, both the causal and acausal pipelines were used, using a forward table-based pre-synaptic event-triggered STDP (as described in section 2.2.1), accordingly to which the state dependent weight updates use only the acausal pipeline. A dedicated latch-based memory module implements the STDP counters. The latch-based design instead of a register file/SRAM based approach allows multiple counters to be updated at the same time corresponding to multiple spikes. In addition to the input spikes (from internal and external to the core), in order to perform state-dependent weight update, the learning engine reads (from the neuron evaluation unit) the state of its fanout neurons. The learning configuration parameters are stored in a small memory in the learning engine.

The axon module holds the routing information to route spikes into the same core or to other cores (**Figure 4F**). The routing information is stored per each neuron that is mapped to a given core. A flag in the routing table indicates whether a spike generated in the core is routed back or routed outside. If it is expected to be routed back, it is inserted into the queue at the input of the synaptic weight memory module. It then performs weight look-up and accumulation in a manner similar to other spikes coming from other cores. For spikes destined for other cores, the output from the axon module is routed to the packetizer unit in the AON router interface block.

Stochastic learning (stochastic synapses and randomized rounding) have proven to be extremely effective in the NSAT framework (see supervised and unsupervised in section 3). To support randomness we implement a robust Linear-feedback Shift Register (LFSR)-based pseudo-random number generator (Tkacik, 2002). There are four individual uniform random number generators, which are combined to generate a normal random sequence which can also be used as a random noise on the neuron membrane potential (Neftci et al., 2017b).

The control unit oversees the overall control of operations in the NSAT core. It is implemented as a state-machine which is responsible for triggering smaller state-machines in each individual block in the design. The sequence of operations in each core follow the behavior as shown in **Figure 4G**. The beginning of a time step is indicated by the `start_tstep` signal, which is a global signal that is broadcasted from the main control center (PC in this case), while the 4 cores in the NSAT tile act as slave accelerator cores. Each core indicates the completion of its neuron evaluation and learning periods by sending out the `done_tstep` signal to the control center. Once `done_tstep` is received from all cores, the control center waits for any global time-step constraint (e.g., simulated time-constant of 1ms) to elapse before sending out the `start_tstep` for the next time-step. This simple approach allows us to achieve multi-core synchronization which can be easily scaled to multiple NSAT tiles.

In addition, the control unit also stores spikes which are attributed with non-zero delays. An array with a size

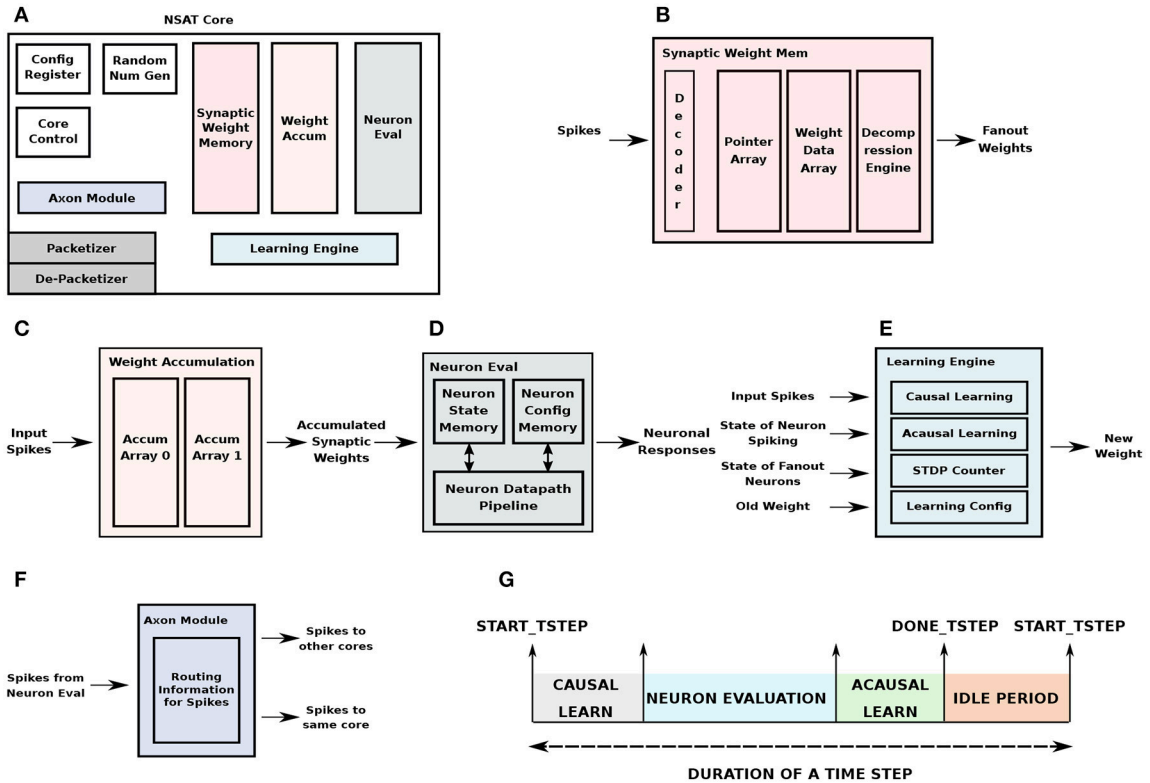

FIGURE 4 | NSAT Hardware Core Architecture. **(A)** The architectural organization of an NSAT hardware core and the different modules constitute the core. **(B)** Synaptic weight memory module, **(C)** synaptic weights accumulation module, **(D)** neuron evaluation module, **(E)** learning engine, and **(F)** neuron axon module, **(G)** time-step duration indicating the sequence of operations in each core.

corresponding to the total neuron space × max future delay time-steps in the control unit stores spikes to be retrieved and used at a future time-step. Configuration parameters corresponding to overall mapping of neurons and learning strategy (STDP or state-based) are stored in these global configuration registers.

2.5.2. Validation of the Architecture

FPGA was used as a means of validating the NSAT architecture and pre-Si demonstration of NSAT software (see **Figure 5**). It is also intended to capture performance statistics of various parts of the NSAT pipeline (e.g., NSAT dynamics, learning module) which would otherwise be extremely slow to capture using RTL simulations. We intend to leverage these statistics for designing better power-management scheme for the NSAT ASIC. The NSAT mapping on the FPGA consumes all types of resources including logic, DSP and memory. DSP utilization is low ($< 1\%$) since NSAT does not use any large multiplier, but only accumulators and shifters. Memory utilization is high (80 BRAMs), primarily due to large synaptic weight memory, pointer memory, neuron state table. One quarter of the logic resources is used to map a single NSAT tile. We expect logic resources and routing (interconnect resources) to be the limiter to mapping multiple tiles. The design was synthesized for a clock frequency of 200 Mhz, which was found enough for a real-time demonstration of spiking neural networks (SNN) based inference workload. At this target frequency, all timing paths were satisfied. However the most critical path was found in the logic for neuron dynamics for a 8-state neuron scenario. Since the goal of the FPGA mapping was only emulation, no power measurement was done.

3. RESULTS

In this section we demonstrate the NSAT capabilities by performing five different tasks using the cNSAT simulator. First, we show that NSAT supports a wide variety of neural responses, such as tonic, bursting and phasing spiking. The second task is a simulation of Amari's neural fields (Amari, 1977) in three different applications: stationary "bump" solutions, target selection and target tracking. Then, we illustrate three learning tasks in supervised and unsupervised settings.

3.1. NSAT Neuron Dynamics Support a Wide Variety of Neural Responses

The Mihalas-Niebur neuron (MNN) model (Mihalas and Niebur, 2009) is a linear leaky integrate-and-fire neuron that is able to capture a wide spectrum of neural responses, such as tonic spiking, bursts of spikes, type I and type II spike responses. Here, we show that the NSAT neuron model can implement the MNN model and thus simulate a similar spectrum of neural responses.

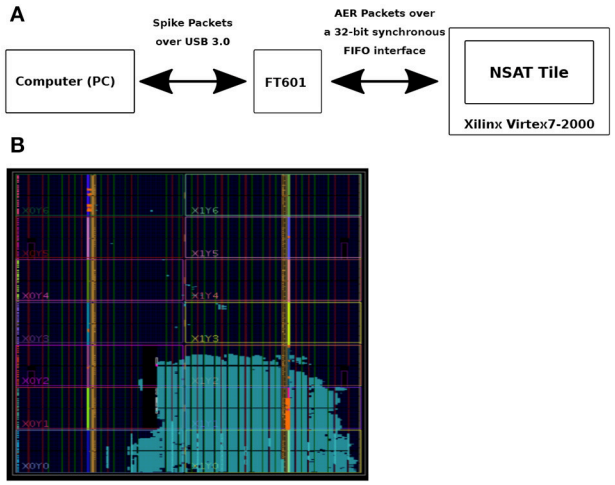

FIGURE 5 | FPGA-based emulation of NSAT. **(A)** Flow of information from a computer to the Xilinx FPGA board. **(B)** NSAT tile mapped to a Virtex-7 2000T device.

The MNN model consists of $N + 2$ equations, where the first two equations describe the membrane potential and an adaptive threshold, respectively. The remaining N equations define internal currents of the neuron. The subthreshold dynamics of MNN neuron are given by,

$$\frac{\mathrm{d}V(t)}{\mathrm{d}t} = \frac{1}{\mathrm{C_m}}\Big(I_e - g(V(t) - E_L) + \sum_{j=1}^{N} I_j(t)\Big), \tag{13a}$$

$$\frac{\mathrm{d}\Theta(t)}{\mathrm{d}t} = a(V(t) - E_L) - b(\Theta(t) - \Theta_\infty), \tag{13b}$$

$$\frac{\mathrm{d}I_j(t)}{\mathrm{d}t} = -k_j I_j(t), \qquad j = 1, \ldots, N, \tag{13c}$$

where $V(t)$ is the membrane potential of the neuron, $\Theta(t)$ is the instantaneous threshold, $I_j(t)$ is the j-th internal current of the neuron. $\mathrm{C_m}$ is the membrane capacitance, I_e is the external current applied on the neuron, g is a conductance constant, E_L is a reversal potential. a and b are some constants, Θ_∞ is the reversal threshold and k_j is the conductance constant of the j-th internal current. The MNN neuron generates spikes when $V(t) \geq \Theta(t)$ and updates neural state as follows:

$$I_j(t) \leftarrow R_j \times I_j(t) + P_j, \tag{14a}$$

$$V(t) \leftarrow V_r, \tag{14b}$$

$$\Theta(t) \leftarrow max\{\Theta_r, \Theta(t)\}, \tag{14c}$$

where R_j and P_j are freely chosen constants, V_r and Θ_r are the reset values for the membrane potential and the adaptive threshold, respectively.

We implement the MNN model using the NSAT framework and following the configuration provided in Mihalas and Niebur (2009). Therefore, we assume $N = 2$ (the number of the internal currents), which has been demonstrated to be sufficient for a wide variety of dynamics (Mihalas and Niebur, 2009).

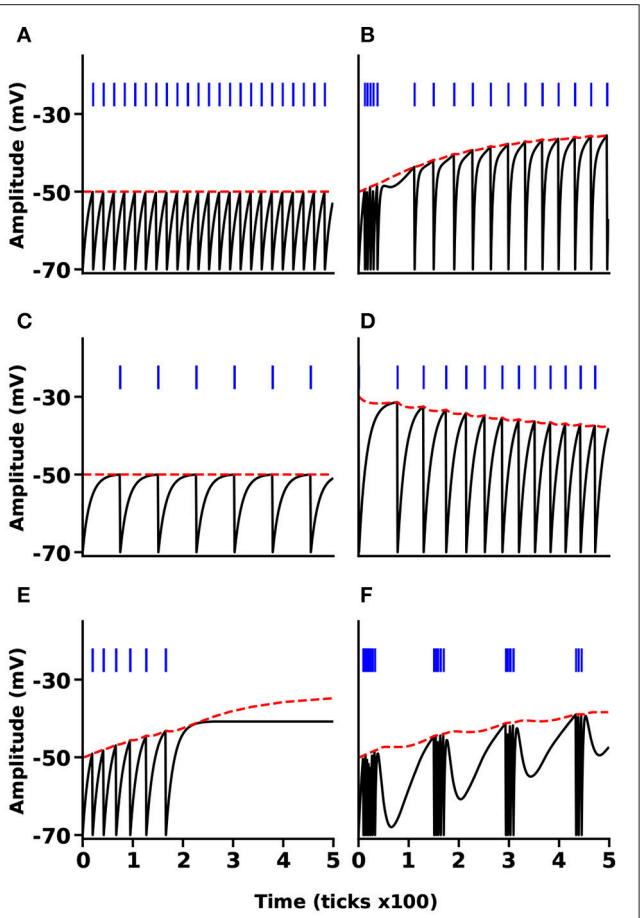

FIGURE 6 | NSAT Mihalas-Niebur Simulation. Results from an NSAT simulation of MNN model. **(A)** Tonic spiking, **(B)** mixed mode, **(C)** class I neuron, **(D)** class II neuron, **(E)** phasic spiking, and **(F)** tonic burst. Black and red lines indicate the membrane potential (state $x_0(t)$) and the adaptive threshold (state $x_1(t)$), respectively. Blue vertical line segments represent spike events.

We simulated the MNN in six different cases, tonic spiking, phasic spiking, mixed mode, class I and II, and tonic bursting. These six neural responses are important because (i) they are the most frequently used neural responses in the field of computational neuroscience and (ii) in Mihalas and Niebur (2009) all the 20 different neural behaviors reduce to three different classes in terms of implementation. Our results produce very similar responses compared to the original MNN ones. **Figure 6** illustrates the results of all these six simulations. The black lines show the membrane potential of the neuron ($V(t)$), the red dashed lines indicate the adaptive threshold ($\Theta(t)$), and the vertical blue line segments show the spike trains for each simulation. Some of the simpler neural responses and behaviors provided by Mihalas and Niebur can be achieved by NSAT in a simpler way. For instance, a linear integrator can be implemented in the NSAT by just solving the equation $x_0[t+1] = x_0[t] + (x_0[t] + \sum_{j=1}^{n} w_{ij}s_j[t])$, where the sum reflects the synaptic input to the neuron.

3.2. Amari's Neural Fields

Neural fields are integro-differential equations usually modeling spatiotemporal dynamics of a cortical sheet, firstly introduced by Shun'ichi Amari in 1977 in his seminal paper (Amari, 1977). Neural fields have a rich repertoire of dynamics (Bressloff, 2011) (waves, breathers, stationary solutions, winner-take-all) and are thus key components of neural computational models. The original Amari's neural field equation is given by:

$$\tau \frac{\partial u(r,t)}{\partial t} = -u(r,t) + I_{\text{ext}} + h + \int_{\Omega} w(|r-r'|) f(u(r',t)) dr', \quad (15)$$

where $u(r,t)$ is the average neural activity at position r and at time t of a neural population, τ is a time constant, Ω denotes a compact subset of $\mathbb{R}^q$, where $q \in \mathbb{N}_{\geq 1}$, $w(|r-r'|)$ is a connectivity function that defines the connectivity strength between neurons at positions r and r'. I_{ext} is an external input that is applied on the neural field (subcortical inputs for instance) and h is the resting potential of the neural population. The function $f(r)$ is the activation or transfer function of the system and in Amari's case is a Heaviside function:

$$f(r) = \begin{cases} 1, & \text{if } x > 0 \\ 0, & \text{otherwise.} \end{cases} \quad (16)$$

The kernel function in this case is a Difference of Gaussians (DoG, see Figure A4a in **Supplementary Information**),

$$w(r) = K_e \exp\left(-\frac{r^2}{2\sigma_e^2}\right) - K_i \exp\left(-\frac{r^2}{2\sigma_i^2}\right), \quad (17)$$

where K_e, K_i, and σ_e, σ_i are the excitatory and inhibitory amplitudes and variances, respectively.

Here, we show the implementation of neural fields in the NSAT framework. First, we observe that the dynamics of each i unit in Equation (A1) is a leaky integrate-and-fire neuron if we consider $f(r)$ as a pre-synaptic spike-event indicator function. Taking into account that the transfer function $f(r)$ is a Heaviside, we can then model every unit i as a leaky integrate-and-fire neuron. This implies that the first state component of neuron i reflects the i-th neural field unit, and the rest of the neuron's state components remain idle. This methodology has been previously used to implement spiking neural fields (Vazquez et al., 2011; de Vangel et al., 2015).

We quantize the kernel function $w(r)$ using a uniform quantizer $Q(r) = \Delta \cdot \lfloor \frac{r}{\Delta} + 0.5 \rfloor$ (see **Figure 7A**). Neural resetting is disabled to match the neural fields behavior (described by Equation 15 and Equation A1): Neurons fire when they reach the firing threshold, but their states do not reset after spiking.

We test the NSAT neural fields implementation on three different tasks. The first model expresses a sustained activity or stationary "bump" solution (Amari, 1977). We simulate 100 internal neurons, all-to-all connected and 100 external neurons (no dynamics) connected with internal neurons in a one-to-one relation. The external neurons transmit spikes generated by a Poisson distribution with maximum firing rate 35 Hz for neurons indexed from $i = 40$ to $i = 60$ and 10 Hz for the rest of the neurons. The total duration of the input signal injection is 400 simulation ticks and the total simulation time is 2500 ticks. Thus, the input is similar to the Gaussian function used in the continuous case (see Figure A4 in **Appendix A**). **Figure 7A** shows the quantized kernel w_{ij} (blue line) and **Figure 7B** indicates the spatial solution of the neural field implementation at the equilibrium point. The solution obtained with the NSAT neural field implementation in **Figure 7B** is similar to the one in Figure A4a in **Appendix A** (red line, Amari's neural field "bump" solution).

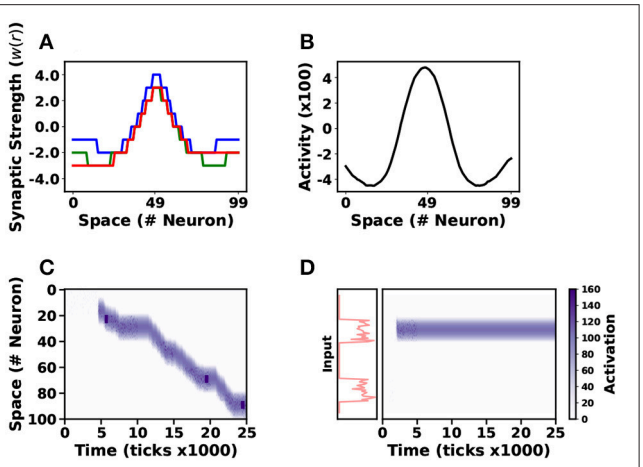

FIGURE 7 | Neural Field Implementation. Three different neural field models were simulated in NSAT. Three different lateral connectivity kernels ($w(r)$) are illustrated in **(A)**. Blue, green and red lines correspond to stationary solution, action-selection and tracking neural field models, respectively. The first neural field model generates a stationary (or "bump") solution **(B)**, a tracking neural field model follows a moving target on the vertical axis (y) as **(C)** illustrates. Finally the action-selection model selects one out of two input stimuli **(D)**, red lines indicate the firing rate of the two stimuli. In **(C,D)**, the purple colormap indicates the neural activity (white-no activity, purple-high activity).

The second task involves target tracking. Asymmetric neural fields have been used for solving target tracking (Cerda and Girau, 2013). We use the same number of neurons (internal and external) and the same simulation time as above, and modify the kernel to an asymmetric one as **Figure 7A** indicates (red line). The stimulus consists of Poisson-distributed spike trains that are displaced along the y-axis every 500 ticks. **Figure 7C** illustrates the NSAT neural field to track the moving target. In this case, a small fraction of neurons receive Poisson-distributed spike trains at a firing rate of 50 Hz, while the rest of the neurons do not receive any input.

Finally, we implemented neural fields' models of action-selection and attention (Vitay and Rougier, 2005). In this case we use the same architecture as in the previous task. The difference is that now we have changed the kernel function and the input. The modified kernel function has weaker excitatory component as **Figure 7A** shows (green line). The input consists of spike trains drawn from a Poisson distribution with two localized high firing rates regions (50 Hz, neurons indexed from $i = 20$ to $i = 40$ and from $i = 70$ to $i = 90$) for 500 simulation ticks (all the

other internal units receive no input). **Figure 7D** shows activity when we apply the input stimulus for 500 simulation ticks. The neural field selects almost immediately one of the two stimuli and remains there during the entire simulation (even after the stimuli removal).

We have shown how NSAT can simulate neural fields (Amari, 1977; Bressloff, 2011), a sort of firing rate models. NSAT can thus contribute a generic framework for neuromorphic implementations of neural fields (Sandamirskaya, 2013) and potentially enhance them with learning features, as described in the following results.

3.3. Supervised Event-Based Learning

Deep neural networks, and especially their convolutional and recurrent counterparts constitute the state-of-the-art of a wide variety of applications, and therefore a natural candidate for implementation in NSAT. The workhorse of deep learning, the gradient descent Back Propagation (BP) rule, commonly relies on high-precision computations and the availability of symmetric weights for the backward pass. As a result, its direct implementation on a neuromorphic substrate is challenging and thus not directly compatible with NSAT. Recent work demonstrated an event-driven Random Back Propagation (eRBP) rule that uses a random error-modulated synaptic plasticity for learning deep representations. eRBP builds on the recent advances in approximate forms of the gradient BP rule (Lee et al., 2014; Baldi et al., 2016; Lillicrap et al., 2016) for event-based deep learning that is compatible with neuromorphic substrates, and achieves nearly identical classification accuracies compared to artificial neural network simulations on GPUs (Neftci et al., 2017b).

We use a two-layer network in NSAT for eRBP equipped with stochastic synapses, and applied to learning classification in the MNIST dataset. The network consists of two feed-forward layers (**Figure 8**) with N_d "data" neurons, N_h hidden neurons and N_p prediction (output) neurons. The class prediction neuron and label inputs project to the error neurons with opposing sign weights. The feedback from the error population is fed back directly to the hidden layers' neurons through random connections. The network is composed of three types of neurons: hidden, prediction, and error neurons.

The dynamics of a hidden neuron follow integrate-and-fire neuron dynamics:

$$\tau_{syn}\frac{\mathrm{d}V^h}{\mathrm{d}t} + V^h = \sum_k \xi(t) w_k s_k(t) \tag{18a}$$

$$\tau_m\frac{\mathrm{d}m^h}{\mathrm{d}t} + m^h = \sum_k g_k^E(s_k^{E+}(t) - s_k^{E-}(t)) \tag{18b}$$

$$\text{if } V^h(t) > V_T \text{ then } V_i^h \leftarrow 0 \text{ during refractory period } \tau_{refr}.$$

where $s_k(t)$ are the spike trains produced by the previous layer neurons, and ξ is a stochastic Bernouilli process with probability $(1-p)$ (indices k are omitted for clarity). Each neuron is equipped with a plasticity modulation compartment m^h following similar subthreshold dynamics as the membrane potential. The term $s^E(t)$ is the spike train of the error-coding neurons and g_k^E is a fixed random vector drawn independently for each hidden neuron. The modulation compartment is not directly coupled to the membrane potential V^h, but indirectly through the learning dynamics. For every hidden neuron, $\sum_k g_k^E = 0$, ensuring that the spontaneous firing rate of the error-coding neurons does not bias the learning. The synaptic weight dynamics follow an error-modulated and membrane-gated rule:

$$\frac{\mathrm{d}}{\mathrm{d}t}w_j^h \propto m^h \Theta(V^h) s_j(t). \tag{19}$$

where Θ is a boxcar function with boundaries b_{min} and b_{max} and the proportionality factor is the learning rate. Weight values were clipped to the range $[-128, 127]$ (8 bits). To mitigate the adverse effect of low-precision weights in gradient descent learning, we used randomized rounding where the first $r = 6$ bits of Δw were interpreted as probability. Prediction neurons and associated synaptic weight updates follow the same dynamics as the hidden neurons except for the modulation, where one-to-one connections with the error neurons are formed (rather than random connections).

Error is encoded using two neurons, one encoding positive error $E+$, the other encoding negative error $E-$. The positive error neuron dynamics are:

$$\frac{\mathrm{d}}{\mathrm{d}t}V_i^{E+} = w^{L+}(s_i^P(t) - s_i^L(t)) \tag{20}$$
$$\text{if } V^{E+} > V_T^E \text{ then } V^{E+} \leftarrow V^{E+} - V_T^E,$$

where $s_i^P(t)$ and $s_i^L(t)$ are spike trains from the prediction neurons and labels. The membrane potential is lower bounded to 0 to prevent negative activity to accumulate across trials. Each error neuron has one negative counterpart neuron. Negative error neurons follow the exact same dynamics but with $w^{L-} = -w^{L+}$. The firing rate of the error-coding neurons is proportional to a linear rectification of the inputs. For simplicity, the label spike train is regular with firing rate equal to τ_{refr}^{-1}. When the prediction neurons classify correctly, $(s_i^P(t) - s_i^L(t)) \cong 0$, such that the error neurons remain silent. Input spike trains were generated as Poisson spike trains with rate proportional to the intesenity of the pixel. Label spikes were regular, i.e., spikes were spaced regularly with inter-spike interval equal to the refractory period. All states were stored in 16 bit fixed point precision (ranging from $-32,768$ to $32,767$), except for synaptic weights which were stored with 8 bit precision (ranging from -128 to 127). To prevent the network from learning (spurious) transitions between digits, the synaptic weights did not update in the first 400 ticks of each digit presentation (1, 500 ticks).

We trained fully connected feed-forward networks MNIST hand-written digits, separated in three groups, training, validation, and testing (50,000, 10,000, 10,000 samples respectively). During a training epoch, each of the training digits were presented in sequence during 150ms (**Figure 8**). Although experiments here focused on a single layer network, random back-propagation can be extended to networks with several layers, including convolutional and pooling layers (Baldi et al., 2016).

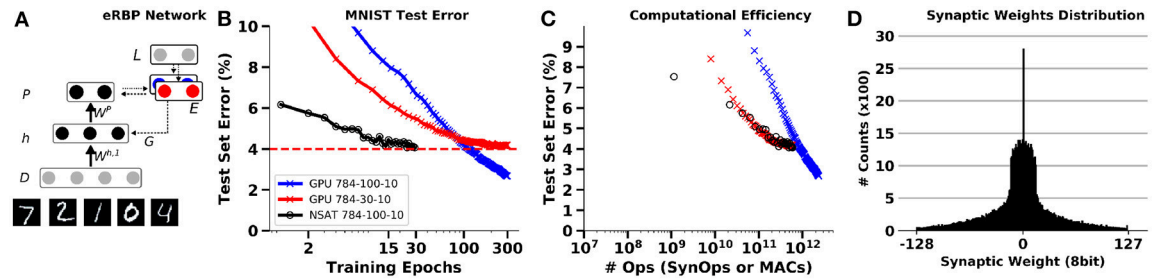

FIGURE 8 | Training an MNIST network with event-driven Random Back-propagation compared to GPU simulations. **(A)** Network architecture. **(B)** MNIST Classification error on the test set using a fully connected 784-100-10 network on NSAT (8 bit fixed-point weights, 16 bits state components) and on GPU (TensorFlow, floating-point 32 bits). **(C)** Energy efficiency of learning in the NSAT (lower left is best). The number of operations necessary to reach a given accuracy is lower or equal in the spiking neural network (NSAT-SynOps) compared to the artificial neural network (GPU-MACs) for classification errors at or above 4%. **(D)** Histogram of synaptic weights of the NSAT network after training. One epoch equals a full presentation of the training set.

Simulations of eRBP on NSAT in a quantized 784-100-10 network demonstrate results consistent with previous findings (Neftci et al., 2017a). To highlight the potential benefits of the NSAT, we compare the number of synaptic operations (SynOp) necessary to reach a given classification accuracy (**Figure 8C**) to the number of multiply operations in standard artificial neural networks. Although larger networks trained with eRBP were reported to achieve error rates as low as 2.02% (Neftci et al., 2017b), this NSAT network with 100 hidden units converges to around 4% error. As two meaningful comparisons, we used one artificial neural network with the same number of hidden units (100) and one with 30 hidden units. The latter network was chosen to achieve similar peak accuracies as the simulated NSAT network. The artificial neural network was trained using mini-batches (the size of each mini-batch was 30 images) and exact gradient back-propagation using TensorFlow (GPU backend). As previously reported, up moderate classification accuracies (here 4%), the NSAT requires an equal or fewer number of SynOps compared to MACs to reach a given accuracy for both networks. We note here that only multiply operations in the matrix multiplications were taken into account in the artificial network. Other operations such as additions, non-linearities were ignored, which would further favor NSAT in this comparison. Finally, **Figure 8D** illustrates the distribution of synaptic weights at the end of learning. It is apparent that synaptic weights concentrate mostly around 0 with a variance of 30. This means that 5 bits precision is sufficient to represent the final synaptic weights. With randomized rounding enabled, lower synaptic precision during learning converges to similar results as with 8 bits of precision, but requires more time to do so (see Figure A5 in **Appendix A**).

These results suggests that a standard computer (e.g., GPU) remains the architecture of choice if classification accuracy on a stationary dataset is the target, regardless of energy efficiency. However, the smaller or equal number of operations, compounded with the fact that a SynOp requires many fold less energy (Merolla et al., 2014) makes a very strong argument for NSAT in terms of energy efficiency for low to moderate accuracies. Therefore, if real-time learning is necessary, or if the streaming data is non-stationary, our results suggest that NSAT can outperform standard architectures in terms of energy efficiency *at least* by a factor equal to the achieved J/MAC to J/SynOp ratio. Furthermore, the NSAT implementation of the event-driven Random Backprogation can serve as the building block for neuromorphic deep neural network architectures in the future.

3.3.1. Real-Time Learning With Event-Driven Random Back-Propagation

The simplicity of the eRBP algorithm and the efficiency of cNSAT render it suitable for on-line real-time learning. To this end we implemented eRBP on cNSAT and interfaced it with a Davis camera (Brandli et al., 2014). A 28x28 pixel, center crop of the Davis camera provides spike events as input to the cNSAT while the user feeds the labels during learning through a keypad. To interleave learning and inference, weight updates were only allowed when a label was presented. This mechanism was implemented within the network through an additional "label-on" neuron, which when active gates the positive and negative error neurons. We trained the network using the MNIST data by alternatively presenting three different MNIST digits. The network was able to learn the MNIST classes on real-time after less than 5 presentations and the results are shown in the **Video S1**.

3.4. Unsupervised Representation Learning

Synaptic Sampling Machines (S2M) are a class of neural network models that use synaptic stochasticity as a means to Monte Carlo sampling in Boltzmann machines (Neftci et al., 2016). Learning is achieved through event-driven Contrastive Divergence (eCD), a modulated STDP rule and event-based equivalent of the original Contrastive Divergence rule (Hinton, 2002). Previous work has shown that, when pre-synaptic and post-synaptic neurons firing follow Poisson statistics, eCD is equivalent to CD (Neftci et al., 2014). Unsupervised learning in RBMs and S2Ms are useful for learning representations of unlabeled data, and perform approximate probabilistic inference (Hinton, 2002; Neftci et al., 2014, 2016).

The NSAT synaptic plasticity dynamics are compatible with eCD under the condition that the refractory period is larger

than the STDP learning window in order to maintain weight symmetry. Here, we demonstrate on-line unsupervised learning implementing S2Ms in NSAT using the network architecture depicted in **Figure 9A**. First, we use two types of input neurons, excitatory and inhibitory (red and blue nodes in **Figure 9A**, respectively). These are the external units that provide the inputs to the event-based Restricted Boltzmann Machine (eRBM) visible units and their synaptic strengths are constant during learning. The visible units (see **Figure 9A**) are all-to-all connected with the hidden ones. Two modulatory units, one excitatory and one inhibitory, are connected with the visible and hidden units (black and gray nodes in **Figure 9A**). The two modulatory units are active in an alternating way, providing an implementation for the positive (only the excitatory unit is on) and the negative (only the inhibitory unit is active) phases of the Contrastive Divergence rule.

In the S2M, weight updates are carried out even if a spike is dropped at the synapse. This speeds up learning without adversely affecting the entire learning process because spikes dropped at the synapses are valid samples in the sense of the sampling process. During the data phase, the visible units were driven with constant currents equal to the logit of the pixel intensity (bounded to the range $[10^{-5}, 0.98]$ in order to avoid infinitely large currents), plus a white noise process of low amplitude σ to simulate sensor noise.

We run the eRBM with eCD using single precision arithmetic (eRBMhp-integers of 16 and eRBM-8 bits), as well as a classical RBM with batch size 32 samples as a reference, on the bars and stripes data set (MacKay, 2003) (see **Figure 9B**). We trained eRBM and eRBMhp using 32 samples per epoch and for 50 epochs. The RBM was trained using 1 batch of 32 samples and 3,000 epochs (until it reaches a similar error as the eRBM and eRBMhp did). At every epoch of the eRBM and the eRBMhp learning we run a test on all 32 different samples and measure the classification error (how many missed classifications), whereas testing is undertaken every 50 epochs in the RBM. **Figure 9C** shows the test set error against the training epochs. The eRBM (black discs) and eRBMhp (green triangles) approach the performance of the classical RBM (blue crosses) faster. **Figure 9D** shows the test set error against the number of operations required for each of the implementations to reach the minimum acceptable error (orange solid line). Similarly to the supervised learning case, eRBM (black discs) and eRBMhp (green triangles) perform less or the same number of operations (synaptic operations) with the classical RBM (MACs). The three Figures A6a–c in **Appendix A** illustrate the synaptic weights (receptive fields) of hidden units for the RBM, eRBMhp, and eRBM, respectively. For all three implementations we used 100 hidden units and 18 visible ones. It is apparent that the receptive fields are qualitatively similar among the three different implementations (for illustration purposes we show only 64 out of 100 receptive fields).

The similarity of this S2M implementation with previous ones and the RBM suggest that NSAT is capable of unsupervised learning for representation learning and approximate probabilistic inference at SynOp–MAC parity. This NSAT implementation of S2Ms requires symmetric (shared) connections and is thus limited to single core implementation. This requirement can be overcome with random contrastive Hebbian learning, as described in Detorakis et al. (2018). There we show that a systems of continuous non-linear differential equations compatible with NSAT neural dynamics is capable of representation learning similarly to restricted Boltzmann machines, while improving the speed of convergence at maintaining high generative and discriminative accuracy on standard tasks.

3.5. Unsupervised Learning in Spike Trains

So far, the results have mostly focused on static data encoded in the firing rates of the neurons. The NSAT learning rule is capable of learning to recognize patterns of spikes. Here we demonstrate a recently proposed post-synaptic membrane potential dependent plasticity rule (Sheik et al., 2016) for spike train learning. Unlike STDP, where synaptic weights updates are computed based on spike timing of both pre- and post-synaptic neurons, this rule triggers a weight update only on pre-synaptic spiking activity.

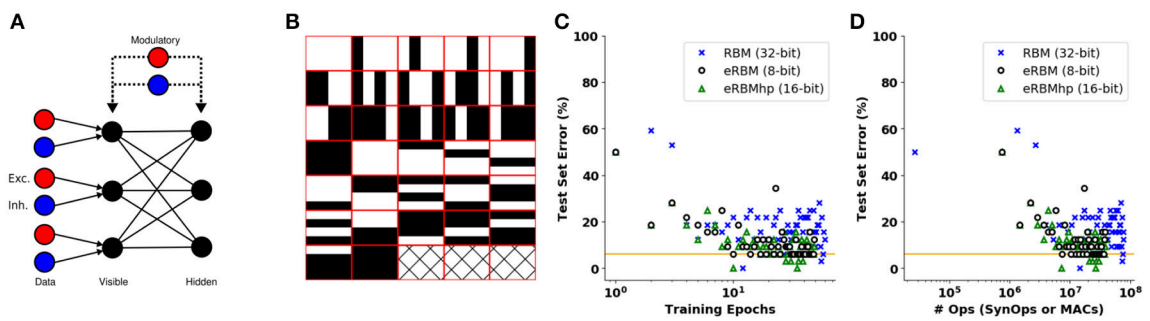

FIGURE 9 | Event-based Restricted Boltzmann Machine (eRBM). **(A)** Event-based RBM neural network architecture. One excitatory and one inhibitory unit project on the visible units providing the input to the eRBM. Two modulatory units project to all the visible and hidden units. **(B)** Bars and stripes data set. **(C)** Training error over 300 epochs for the eRBMhp (single precision, green triangles), eRBM (low precision, discs) and the classic RBM (blue crosses). The orange solid line indicates the minimum acceptable error for all three different algorithms. **(D)** Number of operations vs. test error for the eRBMhp (single precision, green triangles, number of synaptic operations), eRBM (low precision, discs, number of synaptic operations) and the classic RBM (blue crosses, MACs). The orange dashed line indicates the minimum acceptable error for all three different implementations.

The neuron and synapse dynamics are governed by the following equations.

$$\tau_m \frac{\mathrm{d}V}{\mathrm{d}t} = -V + \sum_{j=1}^{N} w_j s_j(t), \tag{21a}$$

$$\tau_{\mathrm{Ca}} \frac{\mathrm{d}\mathrm{Ca}_i}{\mathrm{d}t} = -\mathrm{Ca} + w^{\gamma} s(t), \tag{21b}$$

where V is the membrane potential and Ca is the calcium concentration, w_j is synaptic weight, γ the constant increment of the calcium concentration, and $s_j(t)$, $s(t)$ are the pre-synaptic and post-synaptic spike trains. The synaptic weight update dynamics are given by the equations below:

$$\Theta_m = \delta(V(t) > V_{lth})\eta_+ - \delta(V(t) < V_{lth})\eta_- \tag{22a}$$

$$\mathrm{mod} = \Theta - \eta_h(\bar{\mathrm{Ca}} - \mathrm{Ca}), \tag{22b}$$

$$\Delta w_j = \mathrm{mod}\, s_j(t), \tag{22c}$$

V_{lth} is the membrane threshold that determines LTP or LTD, $\eta_+ = 8$ and $\eta_- = -2$ the corresponding magnitudes of LTP and LTD, $\bar{\mathrm{Ca}}$ is a constant denoting the steady-state calcium concentration and η_h magnitude of homeostasis.

These equations can be efficiently translated to the NSAT using four (4) components per neuron state. Hence component x_0 is the membrane potential V_{mem}, x_1 the calcium concentration Ca_i, x_2 the LTP/LTD state based on thresholded membrane component x_0 (Θ), and x_3 represents the weight modulation (value by which a weight will be updated). The first two state components follow exponential decay dynamics. State components x_2 and x_3 are used to compute the effective weight updates based on the current value of membrane potential and calcium concentration. This is done by exploiting the fact that, at any given point in time, the weight update magnitude (if any) is given purely by the post synaptic states and is the same for every incoming spike within one time step.

We demonstrate these dynamics in learning to identify a hidden spike pattern embedded in noisy spike train. We use 100 input neurons projecting to 5 neurons. A randomly generated fixed spike pattern is repeatedly presented, interspersed with random spike patterns of the same firing rate as **Figure 10A** top raster plot indicates. The initial weights were randomly initialized from a uniform distribution. Over time the synaptic weights converge such that the post-synaptic neurons (indexed 1–5, black line segments in the bottom raster plot of **Figure 10A**) selectively spike only on presentation of the spike pattern as **Figure 10A** bottom raster plot illustrates. The temporal evolution of four components dynamics of the first neuron's state are given in **Figure 10C–E**.

This particular learning rule is a type of unsupervised temporal learning suitable for hardware implementation as demonstrated in Sheik et al. (2016). When coupled with a winner-take all mechanism, it can account for the self organization of spatio-temporal receptive fields.

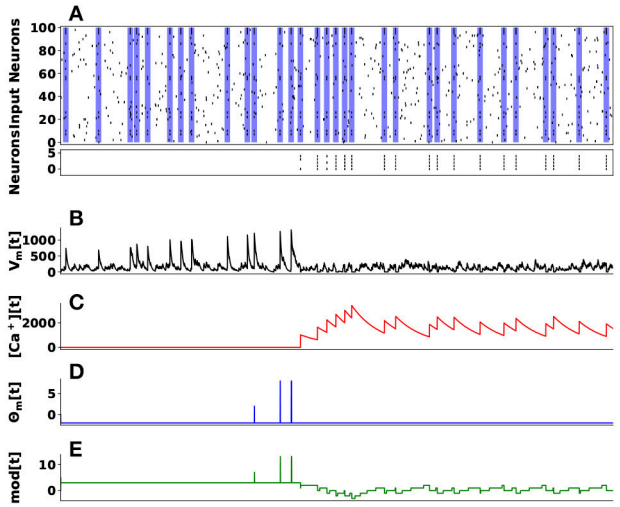

FIGURE 10 | Unsupervised Learning of Spike Patterns. **(A)** On top is a raster plot of 100 input spike trains (pre-synaptic neurons) projecting to 5 post-synaptic neurons. The main goal here is the post-synaptic neurons to learn a hidden spike pattern indicating by the blue vertical bars. The bottom raster plot shows 5 post-synaptic neurons firing when they have learned the hidden spatiotemporal spike pattern. **(B)** Indicates the membrane potential ($V_m[t]$) of the first post-synaptic neuron, **(C)** its calcium concentration ($[Ca^+][t]$), **(D)** LTP/LTD state based on thresholded membrane potential, and **(E)** its weight modulation over time.

4. DISCUSSION

We introduced a neuromorphic computing platform and framework, called Neural and Synaptic Array Transceiver (NSAT), that is able to provide flexible and dynamic learning (on-line) suited for efficient digital implementation. The NSAT takes advantage of tractable linear neural model dynamics and three-factor rules for flexibility in processing and learning. As with existing neuromorphic systems based on Address Event Representation, only (digital) spike events are communicated across cores and between cores using event-based routing. For reasons related to efficiency in projected digital hardware implementations, the proposed framework operates using fixed-point representation. In addition, all the multiplications are implemented as bit shift operations, i.e., multiplications by powers of two. These operations are many-fold more power-efficient compared to floating-point multiply accumulates implemented in digital CMOS (Horowitz, 2014). Taken together, the multiplier-less design, fixed-width representation and event-driven communication enable an energy-efficient and scalable system.

In this work, we demonstrated the capabilities of NSAT by showing first that neuron models with rich behavior such as the Mihalas-Niebur neuron (Mihalas and Niebur, 2009) can be implemented, with comparable spiking dynamics (Mihalas and Niebur, 2009). Next, we demonstrated the simulation of neural field models (Amari, 1977; Bressloff, 2011; Coombes, 2005). We demonstrated three core neural field behaviors, (i) a

stationary "bump" solution (winner-take-all, working memory), (ii) an action-selection process where the neural field chooses between two input signals, and (iii) a target tracking task, where the neural field tracks a moving target. These neural field behaviors form the backbone of many computational models using neural field, such as movement (Erlhagen and Schöner, 2002), pattern generation (Schoner and Kelso, 1988), soft state machines (Neftci et al., 2013) and navigation (Milde et al., 2017).

NSAT is capable of on-line, event-based learning in supervised and unsupervised settings. Embedded learning algorithms are necessary components for real-time learning, which can confer adaptability in uncontrolled environments and more fine-grained context awareness in behaving cognitive agents. This makes NSAT suitable for environments where data are not available *a priori* but are instead streamed in real-time to the device, i.e., using event-based sensors (Liu and Delbruck, 2010).

The implementation of machine learning algorithms in NSAT is a significant achievement, as most machine learning algorithms rely on network-wide information and batch learning. In contrast to machine learning algorithms implemented in standard computers, the NSAT learning is based on information that is locally available at the neuron, i.e., (i) neurons only read weights on their own synapses, (ii) they communicate through all-or-none events (spikes), and (iii) their elementary operations are limited to highly efficient multi-compartment integrate-and-fire. Such implementations can be more scalable compared to their Von Neumann counterparts since the access to system-wide information funnels through the von Neumann bottleneck, which dictates the fundamental limits of the computing substrate.

Learning in NSAT is achieved using three-factor, spike-driven learning rules, i.e., where the third factor modulates the plasticity, in addition to a programmable STDP-like learning rule (Pedroni et al., 2016). In the NSAT, the third, modulating factor is one of the state components of the neuron. The use of a neural state component as a third factor is justified by the fact that gradient descent learning rules in spiking neurons often mirror the dynamics of the neurons and synapses (Pfister et al., 2006; Zenke and Ganguli, 2017), while being addressable by other neurons in the network for error-driven or reward-driven learning. Three factor rules are thus highly flexible and can support multiple different learning rules on a single neuron, thereby enabling the NSAT neuron model to be programmed for supervised, unsupervised and reinforcement learning. Building on previous work, we demonstrated three specific algorithms: (i) event-based deep learning and event-based Random Back-propagation algorithm for supervised settings, (ii) a Contrastive-Divergence algorithm used to train a Restricted Boltzmann Machine implemented on NSAT for unsupervised settings, and (iii) a Voltage-based learning rule for spike-based sequence learning.

The NSAT computes with limited numerical precision in its states (16 bits in this work) and in its weights (8 bits in this work). Often, artificial neural networks require higher precision parameters to average out noise and ambiguities in real-world data (e.g., stochastic gradient descent) (Courbariaux et al., 2014; Stromatias et al., 2015), and introduce challenges at all levels of implementation (Azghadi et al., 2014; Indiveri and Liu, 2015). The NSAT framework mitigates the effect of low precision using a discretized version of randomized rounding (Muller and Indiveri, 2015), where a programmable number of bits are interpreted as update probability. The randomized rounding has been demonstrated in the event-based random back propagation algorithm (Neftci et al., 2017b), a model that is sensitive to weight precision. Moreover, the randomized rounding has a significant effect on the learning rate. We find that the networks perform well even when the synaptic weights are bounded to 256 levels (8 bits precision).

Under the selected specification, large-scale hardware implementation of NSAT is well within reach of current memory technology and can guide (and benefit from) the development of emerging memory technologies (Mostafa et al., 2015; Querlioz et al., 2015; Eryilmaz et al., 2016; Naous et al., 2016). While it is not possible provide direct energy comparisons at this stage, our results consistently highlight a SynOp to MAC parity in learning tasks, i.e., the number of operations required to reach a given task proficiency. The significance of this parity is that the SynOp requires manyfold less energy in reported large-scale neuromorphic implementations (Merolla et al., 2014) compared to equivalent algorithms implemented on standard computers and GPUs. Thus, learning in NSAT is potentially more power efficient compared to standard architectures by a factor at least equal to the ratio J/MAC to J/SynOp, while achieving comparable accuracies.

4.1. Relation to State-of-the-Art and Other Research

Several research groups investigated brain-inspired computing as an alternative to non-von Neumann computing and as a tool for understanding the mechanisms by which the brain computes.

IBM's TrueNorth delivered impressive machine learning implementations in terms of power (Esser et al., 2016). TrueNorth's domain of application is limited to off-line learning, partly to be able to meet targeted design specifications, and partly due to the lack of suitable learning algorithms.

On-chip spike-driven, bistable learning rules were successfully demonstrated in mixed signal neuromorphic hardware. Also, significant effort has gone into learning in digital systems (Venkataramani et al., 2014): Earlier prototypes of IBM's TrueNorth (Seo et al., 2011) also demonstrated the feasibility of low-power embedded learning using STDP, and evolutionary algorithms were recently applied to FPGA-based spiking neural networks (Dean et al., 2014). Stanford's Neurogrid team was among the first to demonstrate STDP learning in mixed-signal neuromorphic hardware (Arthur and Boahen, 2006). Other related neuromorphic projects on learning with neuromorphic hardware are the SpiNNaker (Furber et al., 2014) and BrainScales (Schemmel et al., 2010), as part of the Human Brain Project. SpiNNaker is a parallel multi-core computer architecture composed of half million ARM968 processors (each core is capable of simulating 1,000 neurons) providing a massive implementation of spiking neural networks. The BrainScales project and their subsequent developments

are based on an analog neuromorphic chip, with its main functional blocks consisting of time-accelerated leaky integrate-and-fire neurons (Aamir et al., 2016). There, the proposed learning rule is a hybrid implementation using an on-chip SIMD processor programmable with a range of modulated STDP rules (Friedmann et al., 2017). Both projects are targeted to accelerating simulations of biological neural networks, using technologies that favor speed over compactness and/or power. In contrast, the NSAT design favors compactness and power over speed, and targets application-oriented, flexible, ultra low-power neural and synaptic dynamics for real-time learning tasks.

From the design perspective, the NSAT framework is closest to the TrueNorth ecosystem, but adds on-line learning capabilities and inference dynamics that are compatible with some existing event-based learning dynamics. For instance, NSAT allows for programmable weights and stochastic synapses, a combination that has been shown to be extremely successful in both unsupervised and supervised learning settings.

We believe that the algorithmic-driven design of the NSAT framework, combined with the provided open-source implementation will engage the research community to further investigate brain-inspired, event-based machine learning algorithms.

4.2. NSAT Software Developments

The software stack is a critical component to interface between the majority of potential end-users and the NSAT framework. Our software development efforts are targeted to providing a general purpose framework for Computational Neuroscience and Machine Learning applications that combines the power of machine learning frameworks [such as TensorFlow (Abadi et al., 2015), Neon[2]] and neuromorphic hardware network description [e.g., pyNCS (Stefanini et al., 2014), TrueNorth Corelet (Amir et al., 2013)]. To this end, we are expanding the software for automatic network generation in deep neural network-like scenarios (e.g., automatic differentiation). Such a software stack will enable end users to simulate neural networks (artificial, spiking or compartmental and even firing rate models) without knowledge of NSAT's technical details.

In this article, we briefly introduced PyNSAT (see **Appendix A**), an interface for our NSAT software implementation that can serve as an Application Programming Interface (API) for the NSAT framework. PyNSAT offers a rapid way to program and use the NSAT framework through the Python environment, thereby leveraging the wide capabilities of Python's application ecosystem. Current developments of pyNSAT are targeting proof-of-concept approaches for network synthesis in machine learning applications, in-line with existing machine learning libraries such as Keras (Tensorflow) or Neon.

[2]https://neon.nervanasys.com/index.html/

AUTHOR CONTRIBUTIONS

GD: wrote the NSAT software; GD, SS, and EN: conceived the experiment; GD, SS, and EN: conducted the experiment; GD, SS, and EN: analyzed the results; CA and SP: designed and developed the FPGA implementation. All authors wrote and reviewed the manuscript.

ACKNOWLEDGEMENTS

This work was partly supported by the Intel Corporation and by the National Science Foundation under grant 1640081, and the Korean Institute of Science and Technology. We also thank NVIDIA corporation for providing the GPU used in this work.

SUPPLEMENTARY MATERIAL

The Supplementary Material for this article can be found online at: https://www.frontiersin.org/articles/10.3389/fnins.2018.00583/full#supplementary-material

REFERENCES

Aamir, S. A., Müller, P., Hartel, A., Schemmel, J., and Meier, K. (2016). "A highly tunable 65-nm cmos lif neuron for a large scale neuromorphic system," in *European Solid-State Circuits Conference, ESSCIRC Conference 2016: 42nd* (Lausanne: IEEE), 71–74.

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., et al. (2015). *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software. Available online at: tensorflow.org

Amari, S. (1977). Dynamics of pattern formation in lateral-inhibition type neural fields. *Biol. Cyber.* 27, 77–87. doi: 10.1007/BF00337259

Amir, A., Datta, P., Risk, W. P., Cassidy, A. S., Kusnitz, J. A., Esser, S. K., et al. (2013). "Cognitive computing programming paradigm: a corelet language for composing networks of neurosynaptic cores," in *The 2013 International Joint Conference on Neural Networks (IJCNN)* (Dallas, TX: IEEE), 1–10.

Arthur, J., and Boahen, K. (2006). "Learning in silicon: Timing is everything," in *Advances in Neural Information Processing Systems 18*, eds Y. Weiss, B. Schölkopf, and J. Platt (Cambridge, MA: MIT Press), 75–82.

Azghadi, M. R., Moradi, S., Fasnacht, D. B., Ozdas, M. S., and Indiveri, G. (2015). Programmable spike-timing-dependent plasticity learning circuits in neuromorphic vlsi architectures. *J. Emerg. Technol. Comput. Syst.* 12, 17:1–17:18. doi: 10.1145/2658998

Azghadi, R., Iannella, N., Al-Sarawi, S., Indiveri, G., and Abbott, D. (2014). Spike-based synaptic plasticity in silicon: design, implementation, application, and challenges. *Proc. IEEE* 102, 717–737. doi: 10.1109/JPROC.2014.2314454

Baldi, P., Sadowski, P., and Lu, Z. (2016). Learning in the machine: random backpropagation and the learning channel. *arXiv preprint arXiv:1612.02734*.

Benjamin, B. V., Gao, P., McQuinn, E., Choudhary, S., Chandrasekaran, A. R., Bussat, J., et al. (2014). Neurogrid: a mixed-analog-digital multichip system for large-scale neural simulations. *Proc. IEEE* 102, 699–716. doi: 10.1109/JPROC.2014.2313565

Bi, G.-Q., and Poo, M.-M. (1998). Synaptic modifications in cultured hippocampal neurons: dependence on spike timing, synaptic strength, and postsynaptic cell type. *J. Neurosci.* 18, 10464–10472. doi: 10.1523/JNEUROSCI.18-24-10464.1998

Bienenstock, E., Cooper, L., and Munro, P. (1982). Theory for the development of neuron selectivity: orientation specificity and binocular interaction in visual cortex. *J. Neurosci.* 2, 32–48. doi: 10.1523/JNEUROSCI.02-01-00032.1982

Box, G. E., and Muller, M. E. (1958). A note on the generation of random normal deviates. *Anna. Math. Stat.* 29, 610–611. doi: 10.1214/aoms/1177706645

Brandli, C., Berner, R., Yang, M., Liu, S.-C., and Delbruck, T. (2014). A 240× 180 130 db 3 μs latency global shutter spatiotemporal vision sensor. *IEEE J. Solid State Circ.* 49, 2333–2341. doi: 10.1109/JSSC.2014.2342715

Bressloff, P. C. (2011). Spatiotemporal dynamics of continuum neural fields. *J. Phys. A Math. Theor.* 45:033001. doi: 10.1088/1751-8113/45/3/033001

Cao, Y., Chen, Y., and Khosla, D. (2015). Spiking deep convolutional neural networks for energy-efficient object recognition. *Int. J. Comput. Vis.* 113, 54–66. doi: 10.1007/s11263-014-0788-3

Cerda, M., and Girau, B. (2013). Asymmetry in neural fields: a spatiotemporal encoding mechanism. *Biol. Cyber.* 107, 161–178. doi: 10.1007/s00422-012-0544-0

Clopath, C., Büsing, L., Vasilaki, E., and Gerstner, W. (2010). Connectivity reflects coding: a model of voltage-based stdp with homeostasis. *Nat. Neurosci.* 13, 344–352. doi: 10.1038/nn.2479

Coombes, S. (2005). Waves, bumps, and patterns in neural field theories. *Biol. Cyber.* 93, 91–108. doi: 10.1007/s00422-005-0574-y

Courbariaux, M., Bengio, Y., and David, J.-P. (2014). Low precision arithmetic for deep learning. *arXiv preprint arXiv:1412.7024*.

Davies, M., Srinivasa, N., Lin, T. H., Chinya, G., Cao, Y., Choday, S. H., et al. (2018). Loihi: a neuromorphic manycore processor with on-chip learning. *IEEE Micro* 38, 82–99. doi: 10.1109/MM.2018.112130359

de Vangel, B. C., Torres-Huitzil, C., and Girau, B. (2015). "Stochastic and asynchronous spiking dynamic neural fields," in *2015 International Joint Conference on Neural Networks (IJCNN)* (IEEE), 1–8.

Dean, M. E., Schuman, C. D., and Birdwell, J. D. (2014). "Dynamic adaptive neural network array," in *International Conference on Unconventional Computation and Natural Computation* (Cham: Springer), 129–141.

Detorakis, G., Bartley, T., and Neftci, E. (2018). Contrastive hebbian learning with random feedback weights. *arXiv preprint arXiv:1806.07406*.

Eliasmith, C., and Anderson, C. (2004). *Neural Engineering: Computation, Representation, and Dynamics in Neurobiological Systems*. MIT Press.

Eliasmith, C., Stewart, T., Choo, X., Bekolay, T., DeWolf, T., Tang, Y., et al. (2012). A large-scale model of the functioning brain. *Science* 338, 1202–1205. doi: 10.1126/science.1225266

Erlhagen, W., and Schöner, G. (2002). Dynamic field theory of movement preparation. *Psychol. Rev.* 109, 545–572. doi: 10.1037/0033-295X.109.3.545

Eryilmaz, S. B., Joshi, S., Neftci, E., Wan, W., Cauwenberghs, G., and Wong, H.-S. P. (2016). "Neuromorphic architectures with electronic synapses," in *International Symposium on Quality Electronic Design (ISQED)* (Santa Clara, CA).

Esser, S. K., Merolla, P. A., Arthur, J. V., Cassidy, A. S., Appuswamy, R., Andreopoulos, A., et al. (2016). Convolutional networks for fast, energy-efficient neuromorphic computing. *Proc. Natl. Acad. Sci. U.S.A.* 113, 11441–11446. doi: 10.1073/pnas.1604850113

Florian, R. (2007). Reinforcement learning through modulation of spike-timing-dependent synaptic plasticity. *Neural Comput.* 19, 1468–1502. doi: 10.1162/neco.2007.19.6.1468

Friedmann, S., Schemmel, J., Grübl, A., Hartel, A., Hock, M., and Meier, K. (2017). Demonstrating hybrid learning in a flexible neuromorphic hardware system. *IEEE Trans. Biomed. Circ. Sys.* 11, 128–142. doi: 10.1109/TBCAS.2016.2579164

Furber, S. B., Galluppi, F., Temple, S., Plana, L., et al. (2014). The spinnaker project. *Proc. IEEE* 102, 652–665. doi: 10.1109/JPROC.2014.2304638

Galluppi, F., Lagorce, X., Stromatias, E., Pfeiffer, M., Plana, L. A., Furber, S. B., et al. (2015). A framework for plasticity implementation on the spinnaker neural architecture. *Front. Neurosci.* 8:429. doi: 10.3389/fnins.2014.00429

Graupner, M., and Brunel, N. (2012). Calcium-based plasticity model explains sensitivity of synaptic changes to spike pattern, rate, and dendritic location. *Proc. Natl. Acad. Sci. U.S.A.* 109, 3991–3996. doi: 10.1073/pnas.1109359109

Hinton, G. E. (2002). Training products of experts by minimizing contrastive divergence. *Neural Comput.* 14, 1771–1800. doi: 10.1162/089976602760128018

Hinton, G. E., Srivastava, N., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. R. (2012). Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*.

Horowitz, M. (2014). "1.1 computing's energy problem (and what we can do about it)," in *2014 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)* (San Francisco, CA: IEEE), 10–14.

Indiveri, G., and Liu, S.-C. (2015). Memory and information processing in neuromorphic systems. *Proc. IEEE* 103, 1379–1397. doi: 10.1109/JPROC.2015.2444094

Izhikevich, E. M., and Desai, N. S. (2003). Relating stdp to bcm. *Neural Comput.* 15, 1511–1523. doi: 10.1162/089976603321891783

Jin, X., Rast, A., Galluppi, F., Davies, S., and Furber, S. (2010). "Implementing spike-timing-dependent plasticity on spinnaker neuromorphic hardware," in *The 2010 International Joint Conference on Neural Networks (IJCNN)* (Barcelona: IEEE), 1–8.

Kloeden, P., and Platen, E. (1991). Numerical methods for stochastic differential equations. *Stoch. Hydrol. Hydraul.* 5, 172–172. doi: 10.1007/BF01543058

Lagorce, X., and Benosman, R. (2015). Stick: spike time interval computational kernel, a framework for general purpose computation using neurons, precise timing, delays, and synchrony. *Neural Comput.* 27, 2261–2317. doi: 10.1162/NECO_a_00783

Lazzaro, J., Wawrzynek, J., Mahowald, M., Sivilotti, M., and Gillespie, D. (1993). Silicon auditory processors as computer peripherals. *IEEE Trans. Neural Netw.* 4, 523–528. doi: 10.1109/72.217193

Lee, D.-H., Zhang, S., Biard, A., and Bengio, Y. (2014). Target propagation. *arXiv preprint arXiv:1412.7525*.

Levy, W. B., and Baxter, R. A. (2002). Energy-efficient neuronal computation via quantal synaptic failures. *J. Neurosci.* 22, 4746–4755. doi: 10.1523/JNEUROSCI.22-11-04746.2002

Lillicrap, T. P., Cownden, D., Tweed, D. B., and Akerman, C. J. (2016). Random synaptic feedback weights support error backpropagation for deep learning. *Nat. Communic.* 7:13276. doi: 10.1038/ncomms13276

Liu, S.-C., and Delbruck, T. (2010). Neuromorphic sensory systems. *Curr. Opin. Neurobiol.* 20, 288–295. doi: 10.1016/j.conb.2010.03.007

MacKay, D. J. (2003). *Information Theory, Inference, and Learning Algorithms, Vol. 7*. Cambridge: Cambridge University Pres.

Markram, H., Gerstner, W., and Sjöström, P. (2012). Spike-timing-dependent plasticity: a comprehensive overview. *Front. Synapt. Neurosci.* 4:8. doi: 10.3389/978-2-88919-043-0

Mead, C. (1989). *Analog VLSI and Neural Systems*. Reading, MA: Addison-Wesley.

Merolla, P. A., Arthur, J. V., Alvarez-Icaza, R., Cassidy, A. S., Sawada, J., Akopyan, F., et al. (2014). A million spiking-neuron integrated circuit with a scalable communication network and interface. *Science* 345, 668–673. doi: 10.1126/science.1254642

Mihalas, S., and Niebur, E. (2009). A generalized linear integrate-and-fire neural model produces diverse spiking behavior. *Neural Comput.* 21, 704–718. doi: 10.1162/neco.2008.12-07-680

Milde, M. B., Blum, H., Dietmüller, A., Sumislawska, D., Conradt, J., Indiveri, G., et al. (2017). Obstacle avoidance and target acquisition for robot navigation using a mixed signal analog/digital neuromorphic processing system. *Front. Neurorobot.* 11:28. doi: 10.3389/fnbot.2017.00028

Moreno-Bote, R. (2014). Poisson-like spiking in circuits with probabilistic synapses. *PLoS Comput. Biol.* 10:e1003522. doi: 10.1371/journal.pcbi.1003522

Mostafa, H. (2017). Supervised learning based on temporal coding in spiking neural networks. *IEEE Trans. Neural Netw. Learn. Syst.* doi: 10.1109/TNNLS.2017.2726060

Mostafa, H., Khiat, A., Serb, A., Mayr, C. G., Indiveri, G., and Prodromakis, T. (2015). Implementation of a spike-based perceptron learning rule using tio2- x memristors. *Front. Neurosci.* 9:357. doi: 10.3389/fnins.2015.00357

Muller, L. K., and Indiveri, G. (2015). Rounding methods for neural networks with low resolution synaptic weights. *arXiv preprint arXiv:1504.05767*.

Naous, R., Al-Shedivat, M., and Salama, K. N. (2016). Stochasticity modeling in memristors. *IEEE Trans. Nanotechnol.* 15, 15–28. doi: 10.1109/TNANO.2015.2493960

Neftci, E., Augustine, C., Paul, S., and Detorakis, G. (2017a). "Event-driven random back-propagation: enabling neuromorphic deep learning machines," in *2017 IEEE International Symposium on Circuits and Systems* (Baltimore, MD).

Neftci, E., Binas, J., Rutishauser, U., Chicca, E., Indiveri, G., and Douglas, R. J. (2013). Synthesizing cognition in neuromorphic electronic systems. *Proc. Natl. Acad. Sci. U.S.A.* 110, E3468–E3476. doi: 10.1073/pnas.1212083110

Neftci, E., Das, S., Pedroni, B., Kreutz-Delgado, K., and Cauwenberghs, G. (2014). Event-driven contrastive divergence for spiking neuromorphic systems. *Front. Neurosci.* 7:272. doi: 10.3389/fnins.2013.00272

Neftci, E. O. (2018). Data and power efficient intelligence with neuromorphic learning machines. *iScience.* 5, 52–68. doi: 10.1016/j.isci.2018.06.010

Neftci, E. O., Augustine, C., Paul, S., and Detorakis, G. (2017b). Event-driven random back-propagation: Enabling neuromorphic deep learning machines. *Front. Neurosci.* 11:324. doi: 10.3389/fnins.2017.00324

Neftci, E. O., Pedroni, B. U., Joshi, S., Al-Shedivat, M., and Cauwenberghs, G. (2016). Stochastic synapses enable efficient brain-inspired learning machines. *Front. Neurosci.* 10:241. doi: 10.3389/fnins.2016.00241

O'Neill, M. E. (2014). *Pcg: A Family of Simple Fast Space-Efficient Statistically Good Algorithms for random number generation.* Technical Report HMC-CS-2014-0905, Harvey Mudd College, Claremont, CA.

Park, J., Yu, T., Joshi, S., Maier, C., and Cauwenberghs, G. (2017). Hierarchical address event routing for reconfigurable large-scale neuromorphic systems. *IEEE Trans. Neural Netw. Learn. Sys.* 28, 2408–2422. doi: 10.1109/TNNLS.2016.2572164

Pedroni, B. U., Sheik, S., Joshi, S., Detorakis, G., Paul, S., Augustine, C., et al. (2016). "Forward table-based presynaptic event-triggered spike-timing-dependent plasticity," *2016 IEEE Biomedical Circuits and Systems Conference (BioCAS)* (Shanghai), 580–583. doi: 10.1109/BioCAS.2016.7833861

Pfeil, T., Potjans, T. C., Schrader, S., Potjans, W., Schemmel, J., Diesmann, M., et al. (2012). Is a 4-bit synaptic weight resolution enough? - constraints on enabling spike-timing dependent plasticity in neuromorphic hardware. *Front. Neurosci.* 6:90. doi: 10.3389/fnins.2012.00090

Pfister, J.-P., Toyoizumi, T., Barber, D., and Gerstner, W. (2006). Optimal spike-timing-dependent plasticity for precise action potential firing in supervised learning. *Neural Comput.* 18, 1318–1348. doi: 10.1162/neco.2006.18.6.1318

Putzer, E. J. (1966). Avoiding the jordan canonical form in the discussion of linear systems with constant coefficients. *Am. Math. Month.* 73, 2–7. doi: 10.1080/00029890.1966.11970714

Qiao, N., Mostafa, H., Corradi, F., Osswald, M., Stefanini, F., Sumislawska, D., et al. (2015). A reconfigurable on-line learning spiking neuromorphic processor comprising 256 neurons and 128k synapses. *Front. Neurosci.* 9:141. doi: 10.3389/fnins.2015.00141

Querlioz, D., Bichler, O., Vincent, A. F., and Gamrat, C. (2015). Bioinspired programming of memory devices for implementing an inference engine. *Proc. IEEE* 103, 1398–1416. doi: 10.1109/JPROC.2015.2437616

Sandamirskaya, Y. (2013). Dynamic neural fields as a step toward cognitive neuromorphic architectures. *Front. Neurosci.* 7:276. doi: 10.3389/fnins.2013.00276

Schemmel, J., Brüderle, D., Grübl, A., Hock, M., Meier, K., and Millner, S. (2010). "A wafer-scale neuromorphic hardware system for large-scale neural modeling," in *International Symposium on Circuits and Systems, ISCAS 2010* (IEEE), 1947–1950 .

Schoner, G., and Kelso, J. (1988). Dynamic pattern generation in behavioral and neural systems. *Science* 239, 1513–1520. doi: 10.1126/science.3281253

Seo, J.-S., Brezzo, B., Liu, Y., Parker, B. D., Esser, S. K., Montoye, R. K., et al. (2011). "A 45nm cmos neuromorphic chip with a scalable architecture for learning in networks of spiking neurons," in *2011 IEEE Custom Integrated Circuits Conference (CICC)* (IEEE), 1–4 .

Sheik, S., Paul, S., Augustine, C., Kothapalli, C., and Cauwenberghs, G. (2016). "Membrane-dependent neuromorphic learning rule for unsupervised spike pattern detection," in *BioMedical Circuits and Systems, (BioCAS), 2016* (IEEE).

Shouval, H. Z., Bear, M. F., and Cooper, L. N. (2002). A unified model of NMDA receptor-dependent bidirectional synaptic plasticity. *Proc. Natl. Acad. Sci. U.S.A.* 99, 10831–10836. doi: 10.1073/pnas.152343099

Sjöström, P. J., Rancz, E. A., Roth, A., and Häusser, M. (2008). Dendritic excitability and synaptic plasticity. *Physiol. Rev.* 88, 769–840. doi: 10.1152/physrev.00016.2007

Stefanini, F., Neftci, E., Sheik, S., and Indiveri, G. (2014). Pyncs: a kernel for high-level configuration and definition of neuromorphic electronic systems. *Front. Neuroinform.* 8:73. doi: 10.3389/fninf.2014.00073

Stromatias, E., Neil, D., Pfeiffer, M., Galluppi, F., Furber, S. B., and Liu, S.-C. (2015). Robustness of spiking deep belief networks to noise and reduced bit precision of neuro-inspired hardware platforms. *Front. Neurosci.* 9:222. doi: 10.3389/fnins.2015.00222

Tkacik, T. E. (2002). "A hardware random number generator," in *International Workshop on Cryptographic Hardware and Embedded Systems* (San Francisco, CA:Springer), 450–453.

Tuckwell, H. C., and Jost, J. (2010). Weak noise in neurons may powerfully inhibit the generation of repetitive spiking but not its propagation. *PLoS Comput. Biol.* 6:e1000794. doi: 10.1371/journal.pcbi.1000794

Urbanczik, R., and Senn, W. (2014). Learning by the dendritic prediction of somatic spiking. *Neuron* 81, 521–528. doi: 10.1016/j.neuron.2013.11.030

Vangal, S., Singh, A., Howard, J., Dighe, S., Borkar, N., and Alvandpour, A. (2007). "A 5.1 ghz 0.34 mm 2 router for network-on-chip applications," in *2007 IEEE Symposium on VLSI Circuits* (IEEE), 42–43.

Vazquez, R. A., Girau, B., and Quinton, J.-C. (2011). "Visual attention using spiking neural maps," in *The 2011 International Joint Conference on Neural Networks (IJCNN)* (IEEE), 2164–2171.

Venkataramani, S., Ranjan, A., Roy, K., and Raghunathan, A. (2014). "Axnn: energy-efficient neuromorphic systems using approximate computing," in *Proceedings of the 2014 International Symposium on Low Power lectronics and esign* (ACM), 27–32.

Vitay, J., and Rougier, N. (2005). "Using neural dynamics to switch attention," in *Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005. IJCNN'05*, Vol. 5, (IEEE), 2891–2896.

Vogelstein, R. J., Tenore, F., Philipp, R., Adlerstein, M. S., Goldberg, D. H., and Cauwenberghs, G. (2002). "Spike timing-dependent plasticity in the address domain," in *Advances in Neural Information Processing Systems* (Vancouver, BC), 1147–1154.

Wan, L., Zeiler, M., Zhang, S., Cun, Y. L., and Fergus, R. (2013). "Regularization of neural networks using dropconnect," in *Proceedings of the 30th International Conference on Machine Learning (ICML-13)* (Atlanta, GA), 1058–1066.

Zenke, F., and Ganguli, S. (2017). Superspike: supervised learning in multi-layer spiking neural networks. *arXiv preprint arXiv:1705.11146.*

Zenke, F., and Gerstner, W. (2014). Limits to high-speed simulations of spiking neural networks using general-purpose computers. *Front. Neuroinformat.* 8:76. doi: 10.3389/fninf.2014.00076

Conflict of Interest Statement: CA and SP were employed by company Intel Corporation.

The remaining authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.