An Event-driven Neuromorphic System with Biologically Plausible Temporal Dynamics

Haowen Fang, Amar Shrestha, Ziyi Zhao, Yilan Li, Qinru Qiu
Department of Electrical Engineering & Computer Science, *Syracuse University*, Syracuse, NY 13244, USA
{hfang02, amshrest, zzhao37, yli41, qiqiu}@syr.edu

***Abstract*—Driven by the expanse of Internet of Things (IoT) and Cyber-Physical Systems (CPS), there is an increasing demand to process streams of temporal data on embedded devices with limited energy and power resources. Among all potential solutions, neuromorphic computing with spiking neural networks (SNN) that mimic the behavior of brain, have recently been placed at the forefront. Encoding information into sparse and distributed spike events enables low-power implementations, and the complex spatial temporal dynamics of synapses and neurons enable SNNs to detect temporal pattern. However, most existing hardware SNN implementations use simplified neuron and synapse models ignoring synapse dynamic, which is critical for temporal pattern detection and other applications that require temporal dynamics. To adopt a more realistic synapse model in neuromorphic platform its significant computation overhead must be addressed. In this work, we propose an FPGA-based SNN with biologically realistic neuron and synapse for temporal information processing. An encoding scheme to convert continuous real-valued information into sparse spike events is presented. The event-driven implementation of synapse dynamic model and its hardware design that is optimized to exploit the sparsity are also presented. Finally, we train the SNN on various temporal pattern-learning tasks and evaluate its performance and efficiency as compared to rate-based models and artificial neural networks on different embedded platforms. Experiments show that our work can achieve 10X speed up and 196X gains in energy efficiency compared with GPU.**

***Index Terms*—FPGA, Spiking neural network, Neuromorphic computing**

I. Introduction

Spiking neural networks (SNN) are the third generation of neural networks where the neurons, which are functionally similar to biological neurons, communicate using sequences of spikes. These asynchronous and sparse spikes enable spiking neurons to process large amounts of data using a relatively small number of computations, showing significant energy effciency over deep learning systems [27] [25]. The SNNs have been shown to solve problems solvable by other classes of neural networks along with being computationally more powerful than them [20] [28] [9].

An important question on the road to maximize SNNs potential is how to represent information through spikes. Encoding information in spiking neurons in terms of its firing rate (rate coding) is the traditional coding scheme. The firing rate increases, generally non-linearly, with increasing stimulus intensity while ignoring information in the temporal structure or the exact timing of spikes in the input spike train [16]. This makes rate coding inefficient. Rate coding is also treated statistically such that it ignores irregularities in spiking patterns hence it is robust to noise. However, these irregularities could also convey essential information that could be lost during the encoding process due to its inability to capture the temporal structure of the input spike train. Ignoring the timing information in the spike train also reduces the information capacity of rate coding. Spike trains with similar spiking rate could also have distinct spike patterns, hence should be able to embed more information. Temporal codes enable spiking neurons to capture such temporal structures in the spike train [5].

The ability to detect the temporal codes is determined by the dynamics of the spiking neurons and synapses. To exploit the full potential of temporal coding, it is important for neuron and synapse models to have sufficient dynamics. Such property must be retained during hardware implementations. In this work, we introduce a biologically-inspired neuron model that treats neurons as a dynamical system characterized by at least one state variable, and a biologically-inspired synapse model whose spike response is a filter function. Although analog hardware are potentially most-suited to emulate the behavior of biological neural system, they are usually noisy and unreliable, and hard to be configured precisely. Hence neuron and synapse behavior are usually approximated using discrete digital systems.

In this work, we present a neuromorphic hardware design of the biologically plausible neurons and synapses with low computation overhead and marginal performance loss. Event-driven computation approach is devised from the mathematical property of neuron and synapse model to exploit the sparsity of the SNN activities. A configurable and bio-inspired population encoding scheme is also developed to convert continuous values into temporal spike patterns. Finally we evaluate the performance of proposed design, and compare its computation overhead and energy efficiency with rate based SNN and LSTM. Experiments show that our encoding scheme can reduce model size by 88% compared with rate coding. The system achieves 196X improvement in energy efficiency and 10.1X speed up compared with SNN GPU implementation.

II. Background and Motivation

Spiking neural network can be classified based on how information is encoded as rate-based and time-based system. In a rate-based system, the number of spikes generated by a spiking neuron in a fixed amount of time represents a numerical value. In a rate based SNN, each neuron performs integrate and fire, which is similar to those in the Artificial Neural Network such as Multi Layer Perceptron. This type of SNN has demonstrated

This work is partially supported by the National Science Foundation under Grants CNS-1822165.

state-of-art performance in various tasks [22]. However, it suffers from high spike activities [18], thus cannot fully benefit from event-driven computation. Studies have shown that temporal pattern of spike trains carries information [15] [14] [18]. Such temporal encoding can efficiently represent information using extremely sparse spikes-events. Neurons capability to classify temporal spike patterns relies on synapse dynamic, temporal information of input spikes is preserved by synapses as postsynaptic potentials (PSP). However, keeping track of synaptic dynamics introduces significant computation overhead, because the number of synapses is quadratic to the number of pre- and post-synaptic neurons. It is desirable to exploit advantages of temporal coding and event-driven computation to mitigate the overhead.

There have been various works on hardware implementation of SNNs using different design style, including Application Specific Integrated Circuits(ASICs), analog circuits, emerging devices and FPGAs etc. As an example, IBM TrueNorth [22] neurosynaptic processor is a neuromorphic ASIC chip. It has demonstrated state-of-art accuracy on multiple tasks with very low power consumption. However, the design tradeoffs also impose constraints on the functionality. For example, synapse dynamics are not considered; Only four different weights are allowed in per neuron, etc. These limitations make it difficult to implement algorithms that rely on SNNs temporal behavior on TrueNorth. SpiNNaker [11] is another well-known neuromorphic platform. SpiNNaker chip uses general purpose ARM cores as its computation engines. It aims at parallel simulation of large scale SNN for computational neuroscience. Emerging devices such as memristor have also drawn research interests recently [30] [7]. FPGAs are another popular approach for neuromorphic hardware implementation. They provide higher flexibility with short development time. Intrinsic parallelism of FPGA make it an ideal platform for SNNs. However, most FPGA based SNN employ simplified neuron model and ignore temporal dynamics of neurons and synapses so that only rate-based SNNs are handled. They have limited capability to process temporal data.

Clock-driven computation and event-driven computation are two main paradigms of SNN simulation. In clock-driven simulation, neuron and synapse states are updated every tick. Significant computation overheads are introduced due to the constant update [3], which makes it less favorable for energy-constrained applications. In contrast, event-driven computation is more biologically plausible, it only updates SNN states when a spike is received or issued, thus it has the potential to exploit the sparse activity of SNN to avoid redundant computation. Without considering the synapse dynamic, event-driven computation can be implemented in a straightforward way. Because synapse is stateless, i.e. synapse does not have PSP, an incoming spike causes an instantaneous change on membrane potential. Hence membrane potential is simply an accumulation of weighted input spikes [22] [24]. However, such systems is not capable of simulating complex temporal behaviors of SNN, as the temporal dynamic is ignored.

III. Realistic SNN Model

Without generality, in this work, we adopt the most widely used Leaky Integrate and Fire(LIF) neuron. The neuron has N input synapses as shown in figure 1. Each synapse has a postsynaptic potential (PSP), which is defined by the Spike Response Model as [12]:

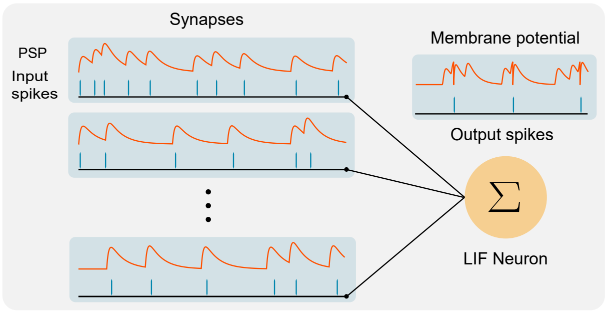


Fig. 1: Spiking neuron model

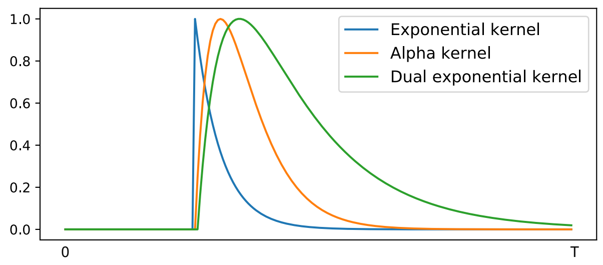


Fig. 2: Synapse kernel

$$PSP(t) = \sum_{t_i}^{t_i<t} K(t - t_i)w \tag{1}$$

where t_i is the arrival time of i_{th} spike, $K(t)$ is the postsynaptic potential(PSP) kernel, which acts as a filter. The $PSP(t)$ is the impulse response of the filter scaled by the weight factor w. The neurons potential is the summation of all weighted input PSP plus a reset term [12]:

$$V(t) = \sum_{i}^{N} w_i \sum_{t_j^i}^{t_j^i<t} K(t - t_j^i) - V_{th} \sum_{t_s^j<t} e^{-\frac{t-t_s^j}{\tau_m}} \tag{2}$$

where w_i is the weight associated with each input synapse, t_j^i is the arrival time of j_{th} spike at i_{th} input synapse, V_{th} is firing threshold. When the potential exceeds the threshold, neuron generates a spike, and the potential is reset. $t_s^j < t$ is the time when the neuron generates a spike. The last term in equation 2 can be interpreted as a negative impulse $V_{reset}(t)$ applied to the neuron to reset potential. Equation 1 and 2 intuitively explain the neurons capability to recognize temporal pattern. At time t, PSP and membrane potential are determined by all previous inputs as if the past information is stored in the PSP. Such filter function in synapse response is critical for biological neuron to learn and recognize temporal patterns [15] [14]. However, this important behavior is usually ignored by many SNN hardware implementations.

The shape of PSP is determined by the kernel $K(t)$. There are three commonly used synapse kernels: exponential kernel, alpha kernel and dual exponential kernel. They are defined as the following: [26]:

$$\text{Exponential kernel:} K(t) = e^{-\frac{t}{\tau_s}} \tag{3}$$

$$\text{Alpha kernel:}K(t) = V_0 \cdot \frac{t}{\tau_s} \cdot e^{-\frac{t}{\tau_s}} \tag{4}$$

$$\text{Dual exponential kernel:}K(t) = V_0(e^{-\frac{t}{\tau_m}} - e^{\frac{-t}{\tau_s}}) \tag{5}$$

where V_0 is a normalization factor to set the maximum synapse response to 1, τ_m and τ_s ($\tau_m > \tau_s$) are membrane and synapse time constants respectively. The exponential kernel is simplified representation of the synapse response. It introduces less computation overhead than alpha and dual exponential kernel, widely used by rate-based SNN models. It can be proved that if all synapses have the same time constant, then the realistic neuron with post synaptic potential behaves the same as a simple integrate and fire neuron for rate-coded input with uniform distributed stochastic behavior [13]. Alpha kernel and dual exponential kernel are more biologically realistic models [26], which are used by temporal pattern classification algorithms such as tempotron [15]. To support various demands, without loss of generality, we approach the work to support the above three types of synapse models. The training of the weight parameters of a temportron neuron is discussed in [15].

IV. Spike-driven Computation

As shown in figure 2, unlike the simplified SNN models, in a biological realistic SNN, synapse is stateful and PSP evolves continuously over time. Equation 2 requires the evaluation of $PSP(t)$, $V(t)$ and $V_{reset}(t)$ based on all previous spike activities. It is impractical to record $PSP(t)$ and $V(t)$ at every time step and store the entire spike history. In addition, the computation involves multiplication and exponential function, which are expensive for FPGA implementation. However, we can utilize the mathematical property of the kernel to derive an event-driven implementation to exploit SNNs sparsity.

Consider the PSP of an exponential synapse:

$$PSP_{exp}(t) = V_0 \sum \exp(-\frac{t - t_i}{\tau_s})w \tag{6}$$

Suppose at time t, the value of $PSP_{exp}(t)$ is known, Δt unit time later, and assume that no input spike during this period, i.e. at time $t' = t + \Delta t$, the PSP can be computed as:

$$\begin{aligned} PSP_{exp}(t') &= V_0 \sum e^{-\frac{t+\Delta t - t_i}{\tau_s}} w \\ &= V_0(\sum_{t_i < t'} e^{-\frac{t-t_i}{\tau_s}} \cdot e^{-\frac{\Delta t}{\tau_s}})w \\ &= PSP_{exp}(t) e^{\frac{-\Delta t}{\tau_s}} \end{aligned} \tag{7}$$

If at time $t_j = t + \Delta t$, there is an input spike, an instantaneous charge is applied on PSP:

$$PSP_{exp}(t_j) = PSP_{exp}(t) e^{\frac{-\Delta t}{\tau_s}} + V_0 \cdot w \tag{8}$$

Similarly, the reset voltage $V_{reset} = V_{th} \sum_{t_s^j < t} e^{-\frac{t-t_s^j}{\tau_m}}$ can also be computed in an event-driven manner:

$$V_{reset}(t + \Delta t) = V_{reset}(t) e^{\frac{-\Delta t}{\tau_m}} \tag{9}$$

Equation 7, 8 and 9 indicate that $V(t)$ can be computed using an incremental approach. By tracking the elapsed time (i.e. Δt), $V(t)$ only needs to be updated when an event triggers the computation, i.e. the computation is only necessary when a spike event is received or issued. In addition $e^{\frac{-\Delta t}{\tau_s}}$ vanishes over time, and will be effectively 0 after certain period of time. Therefore, the values of a few different $e^{\frac{-\Delta t}{\tau_s}}$ can be pre-computed and stored in a look-up table to speed up the computation and reduce the overhead.

Algorithm 1: Spike-driven updating for LIF neuron with exponential synapse

```
Set of input spike: $Q_{spike} = \emptyset$
Set of destination synapse: $Q_{syn}$
$\Delta t$ of each synapse $\Delta t$: $D_t$
Postsynaptic potential: $PSP$
Reset history: $D_r[:] = 0$
Reset potential $V_{reset} = 0$
if $Q_{spike} \neq \emptyset$ then
    foreach synapse i do
        if $i$ in $Q_{syn}$ then
            // Update spike response
            $PSP[i] = PSP[i] \cdot e^{\frac{-D_t[i]}{\tau_s}} + w \cdot V_0$
            $V+ = PSP[i]$
            $D_t[i] = 0$
        else
            $D_t[i]+ = 1$
            $V+ = PSP[i] \cdot e^{\frac{-D_t[i]}{\tau_s}}$
    if $V > V_{th}$ then                // Generate spike
        $D_r = 0$
        $V_{reset} = V_{reset} \cdot e^{\frac{-Delta_{reset}}{\tau_m}} + V_{th}$
        $V = V - V_{reset}$
    else
        $D_r+ = 1$
else
    $D_t+ = 1$
    $D_r+ = 1$
```

The spike-driven implementation for LIF neuron with exponential synapse is given in Algorithm 1. Since PSP is defined to decay, $V(t)$ is only possible to reach peak value when there is an input spike, therefore checking if $V(t)$ exceeds V_{th} is only necessary when spike buffer is not empty. Studies have shown that the SNNs spiking rate is less than 10% [6]. Computation overhead is drastically reduced in such sparse environments utilizing the event-driven approach.

Following a similar principle, the PSP of dual exponential synapse can be computed as:

$$PSP(t + \Delta t) = V_0(PSP_{fall}(t) e^{\frac{-\Delta t}{\tau_m}} - PSP_{rise}(t) e^{\frac{-\Delta t}{\tau_s}}) \tag{10}$$

where $PSP_{fall}(t)$ and $PSP_{rise}(t)$ are:

$$PSP_{fall}(t) = V_0 \sum e^{-\frac{t-t_i}{\tau_m}} w \tag{11}$$

and

$$PSP_{rise}(t) = V_0 \sum e^{-\frac{t-t_i}{\tau_s}} w \tag{12}$$

The dual exponential synapse can be interpreted as the combination of two exponential synapse, therefore the updating algorithm is similar to the exponential synapse. However, as

Algorithm 2: Spike-driven updating for LIF neuron with alpha synapse

```
if $Q_{spike} \neq \emptyset$ then
    $timer[t_{delay}] = 1$
    foreach synapse i do
        if $i$ in $Q_{syn}$ then
            // Update spike response
            $PSP_{exp}[i] = PSP_{exp}[i] \cdot e^{\frac{-D_t[i]}{\tau_s}} + w \cdot V_0$
            $PSP_{\alpha}[i] = PSP_{\alpha}[i] e^{\frac{-D_t[i]}{\tau_s}} + PSP_{exp}[i]$
            $D_t[i] = 0$
        else
            $D_t[i] += 1$
        if $timer[0] == 0$ then
            $V = V + PSP_{exp}[i] \cdot e^{\frac{-D_t[i]}{\tau_s}} + PSP_{\alpha}[i] e^{-\frac{\Delta t}{\tau_s}}$
    if $V > V_{th}$ then                    // Generate spike
        // Update reset history
        $D_r = 0$
        $V_{reset} = V_{reset} \cdot e^{\frac{-D_r}{\tau_m}} + V_{th}$
        $V = V - V_{reset}$
    else
        $D_r += 1$
    $timer << 1$
else
    $D_t += 1$
    $D_r += 1$
    $timer << 1$
    $timer[t_{delay}] = 0$
```

figure 2 shows, the spike response of dual exponential filter is non-instantaneous, there is a delay $t_{delay} = \frac{\tau_m \tau_s}{\tau_m - \tau_s} \log(\frac{\tau_m}{\tau_s})$ between the arrival of spike and the time when PSP reaches peak value. The evaluation of PSP is delayed by t_{delay}, such delayed evaluation can be implemented with a timer.

The evaluation of alpha synapse requires a different algorithm. Suppose at time t, $PSP_{\alpha}(t)$ is known, at $t' = t + \Delta t$, $PSP_{\alpha}(t')$ can be computed as:

$$\begin{aligned} PSP_{\alpha}(t') &= V_0 \sum \frac{t + \Delta t - t_i}{\tau_s} e^{-\frac{t+\Delta t - t_i}{\tau_s}} w \\ &= V_0 \cdot e^{-\frac{\Delta t}{\tau_s}} \sum_{t_i < t'} \left(\frac{t - t_i}{\tau_s} e^{-\frac{t - t_i}{\tau_s}} + \frac{\Delta t}{\tau_s} e^{-\frac{k}{\tau_s}} \right) w \\ &= e^{\frac{-\Delta t}{\tau_s}} PSP_{\alpha}(t) + e^{\frac{-\Delta t}{\tau_s}} \frac{\Delta t}{\tau_s} PSP_{exp}(t) \end{aligned} \tag{13}$$

Alpha synapse PSP reaches maximum $t_{delay} = \tau_s$ later than the arrival if spike, thus similar as dual exponential synapse, the evaluation of alpha synapse PSP is also delayed. The implementation is presented in algorithm 2.

V. Encoding Scheme

Rate coding has been a widely used encoding scheme in SNNs. It represent the value of each input dimension using the activity of an individual neuron that fires at a particular rate. It assumes that the spike count C in a time window T represents a numerical value. There are several drawbacks of this approach: 1) Encoding using the stochastic behavior of an individual neuron is too noisy due to large variance; 2) It introduces high spiking activity, which will deprive the energy efficiency of SNN ; 3) It suffers from limited precision because the value represented by rate coding is quantized using a bin size of $1/T$, where T is the window size. A larger T will improve the data precision but increase the computational latency as well.

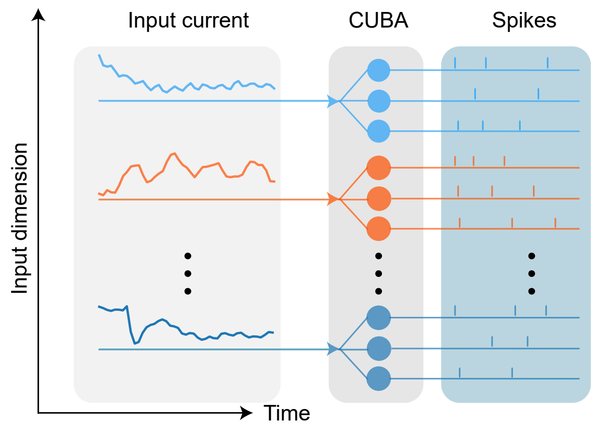


Fig. 3: Encoding method

A. Temporal Population Encoding

To address aforementioned issues, we use an encoding scheme that combines population coding and temporal coding. Population coding represents information by the activity of a group of neurons. Recent research shows that this coding method widely exists in sensory systems [1] and it represents information more accurately [4]. Research also observed that spike-timing carries information in neural systems [5] [21], which leads to temporal coding.

We utilize a population of Current-based Integrate and Fire(CUBA) neurons as encoder. A CUBA neuron is defined as a hybrid system [3]:

$$\begin{aligned} \frac{dV}{dt} &= -\frac{1}{\tau} V + I(t) \\ V \leftarrow 0 & \quad\quad \text{when V exceeds } V_{th} \end{aligned} \tag{14}$$

where $I(t)$ is the time-varying input signal, τ is the membrane time constant, $V(t)$ is the neuron membrane potential. Neuron accumulates the input and updates the potential continuously overtime. The reset event is triggered when the potential exceeds threshold. Unlike the delay coding proposed by [18], in which input value is sampled at time t and translated into specific spike delays, CUBA accumulates all of the input through a low pass filter. In order to differentiate subtle differences and cover wide range of the input $I(t)$, the delay coding requires a high resolution in the delay value, which can either be achieved by increasing the clock frequency or extend the window size. The resolution of the delay is set by the worst case scenario of the aforementioned local input features. CUBA employs the entire spike train to represent the entire input sequence, hence will not be constrained by those local features.

For an input that has D dimensions, each input dimension is connected to E encoder neurons, which form a sub-population.

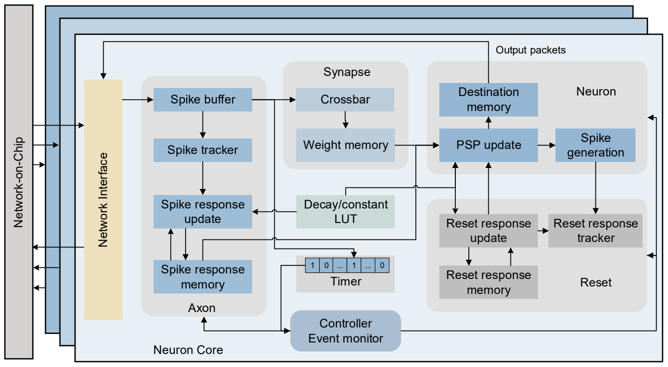


Fig. 4: Neuron core design

D sub-populations are grouped into one population. Each neuron in a sub-population has a different time constant τ, so that they respond differently to the input. In such a way, continuous value is encoded into time-varying patterns of spike trains. Unlike image data where all dimensions have identical range, temporal streams are collected from various sensors, therefore the range, scale, precision may vary. Existing rate-coding or delay-coding are required to provide data precision for the worst case input, while CUBA coding can fine tune its timing constants and sub-population size to provide just enough data resolution for each channel of input [29].

B. Model Size Reduction

Figure 3 shows an example of the encoding scheme. Each input dimension is sent to multiple neurons as a time-varying current. Neurons convert continuous values into spike patterns. Consider a time series of length T and dimension D, and a one layer classifier whose output size is K. Using the rate-coding scheme, since it is not capable of handling temporal pattern but only spatial pattern, the input sequence has to be buffered and flattened into a vector with size $T \times D$. The value of each entry in the vector will be rate coded into a spike train. Hence the classifier requires $T \times D$ input neurons. $T \times D \times K$ weights are required if the classifier is a fully connected network. In the temporal-coding scheme, buffering is no longer necessary, time-varying input is converted to P spike trains, where P is the size of the population, and the number of weights for input layer is reduced from $T \times D \times K$ to $P \times K$. For long sequences, $T \times D >> P$. In addition to fewer weight parameters and simpler network, the CUBA coding scheme will also reduce spike activity compared to rate-coding, which will be discussed in section VII-C1.

VI. Hardware Architecture

We adopted a Network-on-Chip(NoC) architecture proposed in [10] for scalability. Each neuron core is attached to a router. Spikes generated by neuron core are delivered to destination through on-chip network. To increase resource utilization, neuron cores are time-multiplexed and a neuron core implements multiple spiking neurons.

The block diagram of neuron design is shown in figure 4. A neuron core has four sub-units and a controller. The neuron core is time-multiplexed as N LIF neurons with M input synapses. We use a $M \times N$ crossbar to support massive connectivity, where M and N are input size and output size of crossbar respectively. Each crossbar input is called an axon, each crossbar output connects to a LIF neuron and each switch is a synapse. An axon is shared by all neurons. To enable network topology to be flexible, the connectivity of crossbar is configurable, so that the crossbar can provide one-to-many and dedicated one-to-one connection.

The LIF neuron model requires to store the spike history and PSP of each synapse, thus a neuron core requires $M \times N$ memory entries. Updating the PSP and spike history of each synapse also introduces significant computation overhead. The memory and computation overhead can be optimized. PSP can be regarded as a spike response $R(t) = \sum_{t_i}^{t_i<t} K(t - t_i)$ times a weight w. Axon is shared by multiple neurons through synapses, the synapses that belong to different neurons but have same source axon have identical spike histories and spike responses. Thus, tracking spike history and PSP of each synapse is not necessary. Therefore in a core with M axons shared by N neurons, only M different spike responses and spike histories have to be tracked. An independent axon module tracks spike history and spike response. It consists of a DSP block, an adder and two memories. The DSP block performs fixed-point multiplication to update spike response, adder updates spike history. A 16-bit width memory of M entries stores spike responses, A 8-bit memory is used to track the time elapsed since last spike. In common settings, for example, when $\tau_s = 5$, e^{-255/τ_s} is effectively 0, there is no need to spend extra memory to store spike history. If the spike buffer is empty, no computation needs to be done and the spike history of axon is increased by 1. If an axon receives a spike, the spike history is set to 0, and the spike triggers a spike response update.

The charge of alpha synapse and dual exponential synapse is non-instantaneous, evaluation should be delayed. A 16-bit shift register is used as task scheduler to assign the computation task. In exponential synapse, this shift register is bypassed because the charge is instantaneous, evaluation is performed immediately. In alpha synapse or dual synapse mode, upon receiving a spike, k_{th} bit of shift register is set to 1, where $k = t_{delay}$, otherwise k_{th} bit is set to 0. The register is shifted by one bit every time. Controller checks 0_{th} bit of register, computation is performed when 0_{th} bit is 1.

Synapse module stores the connectivity between axons and neurons. A 16-bit width memory stores the weights of synapses, 12 bits for fractional bits, 3 bits for integer bits and 1 for sign. A 16-bit width LUT stores the pre-computed decay factor for spike response computation.

Neuron module consists of a DSP block, an accumulator, a configuration memory and a destination memory. Input spike triggers the neuron to compute the membrane potential; first it reads the spike response from the axon memory, and then the weight multiplies with spike response to obtain the PSP of the first synapse, and accumulates in a dedicated potential accumulator. This procedure is repeated for every synapse to get the membrane potential. Then, the membrane potential is compared with threshold stored in configuration memory. Neuron generates a spike if the potential exceeds the threshold. The spike also triggers the reset module to generate a negative impulse applied to the neuron potential to push it to 0. The reset module has similar structure as axon module, it has a

DPS block to compute reset response, and a memory to track reset history. Reset response history is also set to 0 when generating a spike. Spike also triggers the neuron to forward a spike packet to the router and the on-chip network delivers that spike packet to the destination core specified by the packet information. If no spike is generated, reset response history is increased by 1, no addition computation is required.

VII. Experiments

The proposed design is implemented on Cyclone V FPGA clocked at 75 MHz. To validate and evaluate the design, we use MNIST, Australian Sign Language [17] as benchmarks. The training of the synaptic weight is off-chip and done by the Tempotron algorithm proposed in [15]. We also compared the computation overhead and energy efficiency for different neural networks and platforms.

A. Spatial Pattern Classification

The first experiment is to validate the correctness of the implementation and we test the design for spatial pattern classification with a rate coded input. The number of spikes in a time window represents a value. As we mentioned in Section III, for uniformly distributed stochastic rate coding, the neuron with PSP acts similar as a simple integrate and fire neuron. This experiment is also to demonstrate the flexibility of the model and design.

We use MNIST dataset as benchmark. It includes grayscale images of 10 classes of handwritten digits. The image size is 2828. Each pixel is represented by a spike train, resulting 784 spike trains. The spike rate of each spike train is proportional to corresponding pixels value. We trained a MLP(784x600x10) and then convert it to SNN and use the methods proposed by [8] to find the optimal threshold.

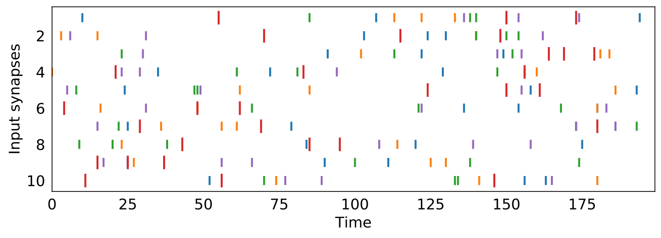
Our work is compared with other FPGA-based spiking neural network implementation as shown in table I. [24] is an event driven neuromorphic hardware implementing IF neuron, the synapse dynamic is ignored. [23] is a step-based FPGA SNN implementation of non-leaky integrate and fire neuron, the computation is performed every time step. Among those works, ours achieves the highest accuracy.

TABLE I: Hardware Comparison

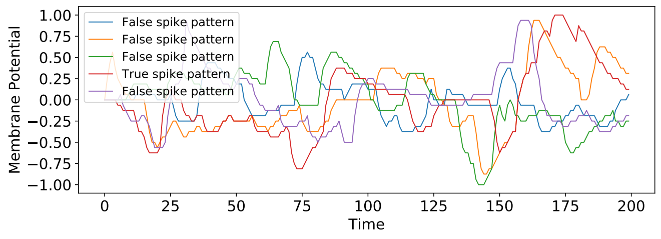
Platform	Update Method	Network Structure	Precision	Acc.
Minitaur [24]	Event driven	784x500x500x10	16 (5.11)	94.2
Hesham [23]	Step based	784x600x10	16 (3.13)	96.8
Tao [19]	Step based	784x512x256x10	16(-)	96.8
This work	Event driven	784x600x10	16 (4.12)	97.7

B. Temporal Spike Pattern Classification

The second experiment is to test whether our design with realistic synapse dynamic is capable of memorizing and detecting desired temporal spike patterns. Such detection is not able to be performed by rate-based SNN models. In the experiment, we train a neural network to memorize 5 different temporal patterns that are generated randomly. Each pattern contains 10 independent spike trains whose length is 200 time steps.



(a) Random spike patterns



(b) Resulting membrane potential

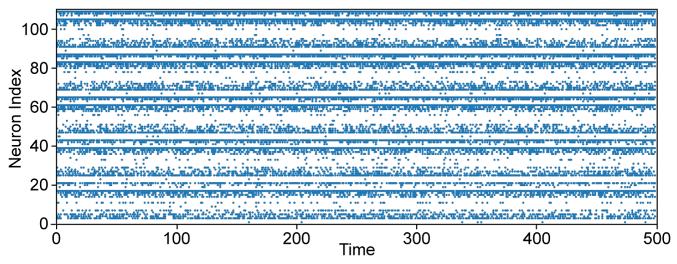
Fig. 5: Spike patterns and neuron potential

Each spike train has same spike rate, which is 5%, however the precise spike timing is different. Although these patterns are different, from the perspective of rate-based input, these patterns are not distinguishable. A fully connected layer is constructed, with 10 input neurons and 5 output neurons. The 5 output neurons are PSP neurons with dual exponential kernel. Their weights are trained using Tempotron algorithm.

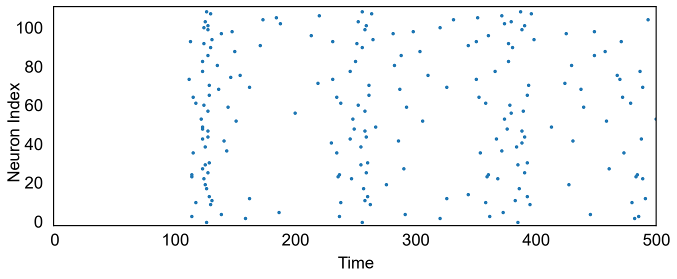
Figure 5a shows the 5 different input spike patterns. The x-axis gives the time. Each row shows the spike activity at a specific time. The five different colors (i.e. red, green, blue, orange and purple) represents the spikes belong to 5 different patterns. When we present different spiking patterns to the network, as shown by an example in 5b, the neurons membrane potential are constantly changing, and eventually, the membrane of the neuron corresponding to the correct class will surge and exceed the threshold. In this experiment, we got 100% detection of those patterns.

C. Performance Evaluation on Sign Language

To evaluate effectiveness of the proposed encoding scheme, the event-driven computation and synaptic dynamics, we apply our work to classify a temporal dataset, Australian Sign Language. It is a multivariate time-series dataset, which includes 95 classes of sign language actions. Data is collected from sensors by tracking hand movements using a data glove. The data has 22 attributes including X, Y, Z positions, roll, pitch and yaw etc. Tempotron [15] algorithm is used to train the synaptic weights. We randomly pick 10 classes from the data set, and convert the time-series into temporal spike patterns using method in section V-A. We also trained a rate-based SNN with same number of neurons and a LSTM with size 10 as the reference system for comparison. In the following experiments, we set $\tau_m = 10$, $\tau_s = 2.5$ for dual exponential kernel. Time window is set to 500 for both rate-based SNN and timing-based SNN. To temporal code the time series into spike patterns, we use a sub-population of size 5 to encode each dimension of the input, resulting in 110 distinct spike



(a) Rate-based input.



(b) Timing-based input

Fig. 6: Input comparison

TABLE II: Coding Efficiency

Model	**Input Dimension**	**Spike Count**	**Firing Rate**	**Window**
Rate SNN	990	91390.7	18.46%	500
Temporal SNN	110	711.3	1.30%	500

trains in total. For rate-based SNN, the sign language data is flattened into a 1-D array of size 990. Each entry in this array is represented by a spike train.

1) Encoding Efficiency: Figure 6 shows the spike patterns generated using rate-based and timing-based input encoding. Each point represents a spike event. Rate-based input has 990 spike trains, only first 110 spike trains of rate-based input are shown in the figure for clarity. The latter is not only significantly sparser, but also has lower dimension. Table II shows the statistics of rate-based SNN and timing based SNN for 100 input samples. The number of spikes generated using timing-based input encoding is only 7.04% of that of rate-based encoding, therefore its on-chip traffic as well as power consumption is significantly reduced.

2) Computation Overhead: Temporal encoding scheme reduces the dimension of a temporal input and also increases the sparsity of the representation. This makes event-driven computation even more effective. Table III shows the computation complexity in the number of different types of operations required to process a sample during inference using temporal SNN as compared to that of the LSTM and the rate-based SNN. Benefiting from the event-driven approach, computation is only necessary when there is input or output spikes, therefore the sparsity leads to significant reduction of computation overhead. In addition, the temporal SNN model size is reduced by 88 % compated with rate SNN. Temporal SNN also reduces number of addition and multiplication by 96.17%. The LUT here refers to look-up-table access, it is needed to read out the leakage value in equation 7. Percentage accuracy is also given in the table. LSTM has the lowest accuracy because, given the same number of units as the SNN, the small network size limits its capability to classify temporal patterns. The timing-based SNN achieves better performance, due to the sufficient temporal information preserved by the input synapses [15]. Rate-based SNN achieves highest performance, while at the cost of additional computation overhead. Please note that the computation will further be reduced in hardware implementation due to the shared spike response.

TABLE III: Computation Overhead

Model	**Param. number**	**Add.**	**Mult.**	**LUT**	**Exp.**	**Acc.**
LSTM	1380	51930	57700	-	10080	84.73
Rate SNN	12880	801900	801900	-	49500	95.66
Temporal SNN	1440	30719	30580	30580	-	94.79

3) Energy Efficiency: To evaluate the energy efficiency across different platforms, the two SNNs are implemented on Nengo spiking neural network simulator [2] and the LSTM is implemented on Keras. The networks are evaluated on Core 7700K CPU, nVidia GTX 1060, ARM A57 processor and TX1 GPU. In addition, the two SNN are tested as the FPGA based neuromorphic hardware. The time, power consumption and energy dissipation are presented in table IV for the inference of 3000 samples.

As can be seen from the table, SNNs are not suitable for Von Neumann systems. The CPUs and GPUs cannot take advantage of the sparsity in the SNNs and its event driven nature. SNNs are even less energy effective compared with LSTM. Rate SNN and temporal SNN consumes 33X and 27.5X times ennergy than LSTM. It also has a higher latency because SNNs have to process a window of 500 ticks. We do observe a reduction in the energy per sample when comparing timing-based SNN to rate-based SNN on CPU and GPU. This is due to reduced input dimension.

The advantage of temporal SNN is clearly visible when implemented on the dedicated hardware. Since its input dimension is significantly reduced, the time to process each sample and the consumption of programmable logic resource as well as on-chip memory are reduced. In addition, the sparsity in timing-based input can fully exploit the benefit of event-driven computation. Finally, the sparsity in the spiking activities significantly reduces the on-chip communication overhead, hence contributes to the energy reduction. Temporal SNN is 8.43X faster than rate SNN on FPGA platform. When compared with GPU and CPU, the advantage is more significant. Temporal SNN on FPGA achieves 10.15X speed up than GPU and 196X improvement in energy efficiency.

VIII. Conclusions

In this work, we proposed a biological realistic neuromorphic system. The proposed hardware with realistic synapse exhibits complex temporal dynamic, which enables it to process temporal data. To fully exploit the sparsity of SNN, it runs in an event-driven manner. A population encoding scheme is also introduced to convert continuous value into discrete spike

TABLE IV: Energy Efficiency

Platform	Model	Time (s)	Avg. power(W)	Time(ms) per sample	Energy(J) per sample
i7 7700K	LSTM	2.79	28.95	1.12	0.032
	Rate SNN	182.56	17.48	60.86	1.06
	Temporal SNN	143.52	18.43	47.83	0.88
GTX 1060	Rate SNN	30.63	16.43	10.21	0.17
	Temporal SNN	24.04	17.90	8.01	0.14
ARM A57	Rate SNN	686.39	4.72	228.79	1.08
	Temporal SNN	630.90	4.63	210.30	0.97
TX1 GPU	Rate SNN	141.89	5.63	47.30	0.27
	Temporal SNN	125.09	5.39	41.70	0.22
FPGA	Rate SNN	19.9646	1.029	6.65	0.0065
	Temporal SNN	2.364	0.926	0.79	0.00072

trains. The proposed design is implemented on Cyclone V FPGA. To evaluate its performance, multiple experiments on different platforms are tested. The proposed hardware shows significant energy efficiency. Since proposed design supports standard Integrate and Fire Neuron, its application is not only limited to temporal spike pattern classification, it also capable of implementing rate-based SNNs and simulation for biologically plausible SNNs. We need to point out that a significant challenge of proposed design is the lack of effective SNN training method to classify temporal spike patterns. Existing algorithms such as [15], [14] suffer from scalability, cannot be extended to multiple layers like RNN/LSTM, which benefit from deep structures. There is an urgency to explore more efficient training rules in the future to fully utilize the advantage of SNNs.

REFERENCES

[1] B. B. Averbeck, P. E. Latham, and A. Pouget. Neural correlations, population coding and computation. *Nature reviews neuroscience*, 7(5):358, 2006.

[2] T. Bekolay, J. Bergstra, E. Hunsberger, T. DeWolf, T. C. Stewart, D. Rasmussen, X. Choo, A. Voelker, and C. Eliasmith. Nengo: a python tool for building large-scale functional brain models. *Frontiers in neuroinformatics*, 7:48.

[3] R. Brette, M. Rudolph, T. Carnevale, M. Hines, D. Beeman, J. M. Bower, M. Diesmann, A. Morrison, P. H. Goodman, F. C. Harris, et al. Simulation of networks of spiking neurons: a review of tools and strategies. *Journal of computational neuroscience*, 23(3):349–398, 2007.

[4] D. A. Butts and M. S. Goldman. Tuning curves, neuronal variability, and sensory coding. *PLoS biology*, 4(4):e92, 2006.

[5] D. A. Butts, C. Weng, J. Jin, C.-I. Yeh, N. A. Lesica, J.-M. Alonso, and G. B. Stanley. Temporal precision in the neural code and the timescales of natural vision. *Nature*, 449(7158):92, 2007.

[6] J. A. Cardin, M. Carlén, K. Meletis, U. Knoblich, F. Zhang, K. Deisseroth, L.-H. Tsai, and C. I. Moore. Driving fast-spiking cells induces gamma rhythm and controls sensory responses. *Nature*, 459(7247):663–667, 2009.

[7] P. Chi, S. Li, C. Xu, T. Zhang, J. Zhao, Y. Liu, Y. Wang, and Y. Xie. Prime: A novel processing-in-memory architecture for neural network computation in reram-based main memory. In *ACM SIGARCH Computer Architecture News*, volume 44, pages 27–39. IEEE Press, 2016.

[8] P. U. Diehl, D. Neil, J. Binas, M. Cook, S.-C. Liu, and M. Pfeiffer. Fast-classifying, high-accuracy spiking deep networks through weight and threshold balancing. In *2015 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE, 2015.

[9] C. Ding, S. Liao, Y. Wang, Z. Li, N. Liu, Y. Zhuo, C. Wang, X. Qian, Y. Bai, G. Yuan, et al. C ir cnn: accelerating and compressing deep neural networks using block-circulant weight matrices. In *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 395–408. ACM, 2017.

[10] H. Fang, A. Shrestha, D. Ma, and Q. Qiu. Scalable noc-based neuromorphic hardware learning and inference. In *2018 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE, 2018.

[11] S. B. Furber, D. R. Lester, L. A. Plana, J. D. Garside, E. Painkras, S. Temple, and A. D. Brown. Overview of the spinnaker system architecture. *IEEE Transactions on Computers*, 62(12):2454–2467, 2013.

[12] W. Gerstner and W. M. Kistler. *Spiking neuron models: Single neurons, populations, plasticity*. Cambridge university press, 2002.

[13] W. Gerstner, W. M. Kistler, R. Naud, and L. Paninski. *Neuronal dynamics: From single neurons to networks and models of cognition*. Cambridge University Press, 2014.

[14] R. Gütig. Spiking neurons can discover predictive features by aggregate-label learning. *Science*, 351(6277):aab4113, 2016.

[15] R. Gütig and H. Sompolinsky. The tempotron: a neuron that learns spike timing–based decisions. *Nature neuroscience*, 9(3):420, 2006.

[16] T. M. Jessell, E. R. Kandel, J. H. Schwartz, and T. Jessel. *Principles of neural science*. Elsevier, 1991.

[17] M. W. Kadous et al. *Temporal classification: Extending the classification paradigm to multivariate time series*. University of New South Wales, 2002.

[18] T. Liu, Z. Liu, F. Lin, Y. Jin, G. Quan, and W. Wen. Mt-spike: A multi-layer time-based spiking neuromorphic architecture with temporal error backpropagation. In *Proceedings of the 36th International Conference on Computer-Aided Design*, pages 450–457. IEEE Press, 2017.

[19] T. Luo, X. Wang, C. Qu, M. K. F. Lee, W. T. Tang, W.-F. Wong, and R. S. M. Goh. An fpga-based hardware emulator for neuromorphic chip with rram. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2018.

[20] W. Maass. Networks of spiking neurons: the third generation of neural network models. *Neural networks*, 10(9):1659–1671, 1997.

[21] R.-M. Memmesheimer, R. Rubin, B. P. Ölveczky, and H. Sompolinsky. Learning precisely timed spikes. *Neuron*, 82(4):925–938, 2014.

[22] P. A. Merolla, J. V. Arthur, R. Alvarez-Icaza, A. S. Cassidy, J. Sawada, F. Akopyan, B. L. Jackson, N. Imam, C. Guo, Y. Nakamura, et al. A million spiking-neuron integrated circuit with a scalable communication network and interface. *Science*, 345(6197):668–673, 2014.

[23] H. Mostafa, B. U. Pedroni, S. Sheik, and G. Cauwenberghs. Fast classification using sparsely active spiking networks. In *2017 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1–4. IEEE, 2017.

[24] D. Neil and S.-C. Liu. Minitaur, an event-driven fpga-based spiking network accelerator. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 22(12):2621–2628, 2014.

[25] A. Ren, Z. Li, C. Ding, Q. Qiu, Y. Wang, J. Li, X. Qian, and B. Yuan. Sc-dcnn: Highly-scalable deep convolutional neural network using stochastic computing. *ACM SIGOPS Operating Systems Review*, 51(2):405–418, 2017.

[26] D. Sterratt, B. Graham, A. Gillies, and D. Willshaw. *Principles of computational modelling in neuroscience*. Cambridge University Press, 2011.

[27] S. Wang, Z. Li, C. Ding, B. Yuan, Q. Qiu, Y. Wang, and Y. Liang. C-lstm: Enabling efficient lstm using structured compression techniques on fpgas. In *Proceedings of the 2018 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pages 11–20. ACM, 2018.

[28] Y. Wang, C. Ding, Z. Li, G. Yuan, S. Liao, X. Ma, B. Yuan, X. Qian, J. Tang, Q. Qiu, et al. Towards ultra-high performance and energy efficiency of deep learning systems: an algorithm-hardware co-optimization framework. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[29] S. Yarrow and P. Seriès. The influence of population size, noise strength and behavioral task on best-encoded stimulus for neurons with unimodal or monotonic tuning curves. *Frontiers in computational neuroscience*, 9:18, 2015.

[30] Y. Zhang, X. Wang, and E. G. Friedman. Memristor-based circuit design for multilayer neural networks. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 65(2):677–686, 2018.