S2N2: A FPGA Accelerator for Streaming Spiking Neural Networks

Alireza Khodamoradi
alirezak@eng.ucsd.edu
University of California, San Diego

Kristof Denolf
kristof@xilinx.com
Xilinx

Ryan Kastner
kastner@eng.ucsd.edu
University of California, San Diego

ABSTRACT

Spiking Neural Networks (SNNs) are the next generation of Artificial Neural Networks (ANNs) that utilize an event-based representation to perform more efficient computation. Most SNN implementations have a systolic array-based architecture and, by assuming high sparsity in spikes, significantly reduce computing in their designs. This work shows this assumption does not hold for applications with signals of large temporal dimension. We develop a streaming SNN (S2N2) architecture that can support fixed-per-layer axonal and synaptic delays for its network. Our architecture is built upon FINN and thus efficiently utilizes FPGA resources. We show how radio frequency processing matches our S2N2 computational model. By not performing tick-batching, a stream of RF samples can efficiently be processed by S2N2, improving the memory utilization by more than three orders of magnitude.

CCS CONCEPTS

• **Hardware → Hardware accelerators**; *High-speed input / output*; *Reconfigurable logic applications*; **Emerging architectures**.

KEYWORDS

Spiking Neural Networks, Streaming, FINN, RF

ACM Reference Format:
Alireza Khodamoradi, Kristof Denolf, and Ryan Kastner. 2021. S2N2: A FPGA Accelerator for Streaming Spiking Neural Networks. In *Proceedings of the 2021 ACM/SIGDA International Symposium on Field Programmable Gate Arrays (FPGA '21), February 28-March 2, 2021, Virtual Event, USA.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3431920.3439283

1 INTRODUCTION

Artificial Neural Networks have shown remarkable success in large-scale image and video recognition [39, 41], speech recognition [5, 13], radio signal classification [33], and many other application domains [15, 20]. Spiking Neural Networks (SNNs) use an event-based model that better mimics biological neurons [12, 25] with the goal of providing high prediction accuracy while using minimal energy [42]. Recent advancements in SNN architecture design and training methods show promise in matching the accuracy of non-spiking ANNs [2, 3, 10, 40] and the potential to out-perform a similar-sized non-spiking ANN [8]. However, much work remains until we fully uncover the potentials of SNNs [42].

A conventional neuron model assumes every input requires calculation and performs N operations, e.g., multiplying and accumulating N input values with N weights (and an optional bias - see Equation 1). A typical convolutional layer in a modern feedforward neural network includes many neurons with an equal number of inputs (fan-in). This architecture creates patterns suitable for massively parallel implementations. Frameworks such as FINN [4], fpgaConvNet [48], and Eyeriss [6] provide efficient implementations of this architecture on FPGAs.

Conversely, event-based neural networks reduce wasted computation by only processing received events. For example when an event-based neuron with fan-in=N receives $M < N$ events, calculating the input only requires M operations (Equation 2). This assumes a certain amount of sparsity and requires dynamic handling of events. This sparsity creates a run-time dependency based on the input data and induces unpredictable and potentially irregular memory accesses. Therefore exploiting parallelism in SNN is more challenging than CNNs, DNNs, and other more traditional neural networks.

Previous works such as IBM TrueNorth [2], Intel Loihi [9], SpiNNaker [35], and BlueHive [28] have shown that processing SNN events can be efficiently implemented in custom hardware for both training and inference. Neurogrid [3] uses a mixed analog-digital approach for simulating large-scale spiking models and Minitaur [31] and SpinalFlow [30] describe inference accelerators for SNNs. Event processing is either done by encoding and storing events in a buffer to be processed in a systolic fashion (tick-batching) [2, 9, 30, 31] or a spike-routing mechanism is used to prevent deadlocks [3, 28, 35].

In this work, we introduce a streaming accelerator for spiking neural networks, S2N2. Our design efficiently supports both axonal and synaptic delays for feedforward networks with interlayer connections. We show that because of the spikes' binary nature, a binary tensor can be used for addressing the input events of a layer. We describe the condition when addressing events with a binary tensor, and no tick-batching (streaming) can provide a better memory utilization compared to encoding events and tick-batching. We show that this condition depends on the input's sparsity (more detail in Section 3.3) and holds, for example, applications, in particular for Radio Frequency (RF) applications.

We use the FINN framework [47] as our baseline and extend it with new functions for supporting our event-based processing of SNNs. Our proposed changes can maintain the high throughput of FINN and provide an efficient streaming implementation for SNNs by benefiting from FINN's optimized utilization footprint.

We also propose novel example applications for SNNs in the RF domain that can benefit from S2N2's streaming architecture.

FPGA '21, February 28-March 2, 2021, Virtual Event, USA

ACM ISBN 978-1-4503-8218-2/21/02…$15.00
https://doi.org/10.1145/3431920.3439283

By looking at RF samples as events in In-phase and Quadrature (I/Q) plane, RF samples can be turned into highly sparse events as input to a SNN. RF inputs available in RF datasets [32, 34] have a large temporal dimension, and a SNN designed for classifying these inputs can efficiently be implemented in S2N2.

In addition, our design is tested by using some of the published applications for SNNs in the image classification domain [18, 40]. In order to adopt these previously published spiking networks to S2N2, we propose new architectural changes in these networks and show that modified networks can maintain their accuracy after re-training with new hyperparameters.

Our contributions can be summarized as following:

- We introduce a new streaming architecture, S2N2, for accelerating SNNs on FPGA platforms.
- We describe how to reduce the memory utilization for inputs with a large temporal dimension.
- We propose novel applications for SNNs in the RF domain that can benefit from our streaming architecture.
- We release our code as open-sourced to enhance accessibility and aid in future comparisons of our work [1].

The remainder of the paper is organized as follows. In Section 2, we introduce SNNs in more depth and review the previous work on SNN FPGA implementations. S2N2 is described in detail in Section 3 and we demonstrate its advantages and implementation results for RF applications in Section 4. Additionally, Section 5 applies the S2N2 architecture to previously published SNNs for image classifications. We conclude our work in section 6.

2 SPIKING NEURAL NETWORK

Spiking neural networks are the third generation of ANNs developed to process information more similar to biological neural networks [25, 42]. In these networks, neurons propagate information by using spikes. The information is coded into the rate and time-of-arrival of the spikes.

Figure 1 shows an example comparison between a frame-based input and an event-based input with rate-coded spikes. In general, input to each layer in a non-spiking neural network is a tensor of values (a multi-channel matrix). In contrast, in a SNN, inputs to each layer are events that have temporal and spatial positions. The temporal dimension of the input in SNNs consists of several "*ticks*". A tick is the minimum unit of time in a SNN that a neuron evaluates its input, updates its potential, and, depending on its model parameters, may generate a spike in its output.

For a more clear comparison, we look at the operations required for evaluating inputs in non-spiking and spiking neurons. Input to a non-spiking neuron is calculated as follows:

$$I = \sum_{i=0}^{N} w_i x_i \tag{1}$$

Here, x_i are the input values and w_i are their associated weights and bias is not shown.

While input to a spiking neuron at tick=t is calculated as following:

$$I_t = \sum_{i \in S_t} w_i \tag{2}$$

Here, S_t is the set of inputs to the neuron that have a spike at tick=t, and w_i are the weights associated with those inputs.

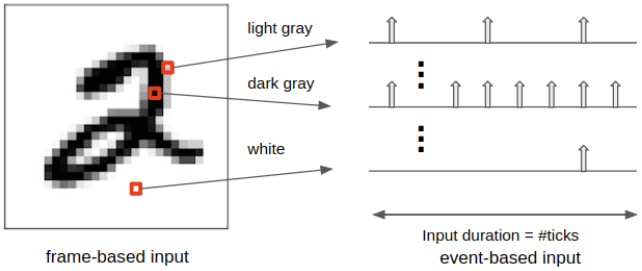


Figure 1: A frame-based input (on the left) is a matrix of numbers. An event-based input (on the right) includes trains of spikes. In this example, the number of trains is equal to the number of pixels in the frame. The duration of the spike trains is equal to the number of ticks. At each tick, up to one spike can exist in each train.

With sparsity in input spikes, Equation 2 requires fewer and simpler accumulation operations compared to the fixed number of MAC operations required in Equation 1. However, Equation 1 is more suitable for applying techniques such as loop-unrolling for exploiting parallelism. In addition, Equation 2 requires memory to store S to keep track of input spikes. Later in this work, we provide solutions to efficiently implement Equation 2 on custom hardware.

In a conventional ANN, an activation function of a neuron defines the output of that neuron given an input. In SNNs, neuron's output and evaluation of neuron potential are governed by *neuron's model*.

Neuron models used in SNNs are biologically plausible models that are computationally more powerful units compared to activation functions used in non-spiking networks [12]. These models are capable of extracting the temporal information embedded in their input and perform more complex tasks [25].

Although more complex mathematical models such as Izhikevich [16] and Hodgkin–Huxley [14] can accurately model a biological neuron's behavior, current training methods for SNNs are not geared to train these complex models [42]. For now, simpler models such as Integrate and Fire (IF) and Leaky Integrate and Fire (LIF) are more prevalent in current SNN applications. In this work, we use a LIF model with one internal parameter.

2.1 LIF Model

Leaky Integrate and Fire (LIF) model is a neuron model widely used in SNN applications [18, 21, 36, 40, 50]. LIF model memorizes its past inputs by adding every input to its membrane potential and uses a leak (decay) parameter to forget them. This leak parameter is reflecting the diffusion of ions that occurs through the membrane when some equilibrium is not reached in the cell:

[1] github.com/arkhodamoradi/s2n2

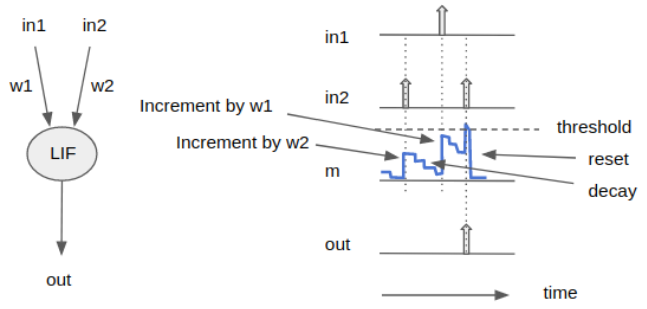


Figure 2: LIF neuron with two inputs. An incoming spike increases the membrane voltage by the weight associated with its connection. A decay parameter decreases the membrane potential, and if this voltage passes a threshold, it resets to a preset value, and the neuron generates a spike at its output.

$$m_t = (1 - \text{out}_{t-1}) * d * m_{t-1} + I_t \quad , \quad 0 < d < 1 \tag{3}$$

$$\text{out}_t = \begin{cases} 1, & \text{if} \quad m_t > T \\ 0, & \text{ow} \end{cases} \tag{4}$$

Here, $d \in (0, 1)$ is the decaying leak parameter, T is the threshold, m_t is the membrane voltage at tick= t, and I_t is the input from Equation 2. The term $(1 - \text{out}_{t-1})$ in Equation 3 is the reset mechanism that sets the membrane voltage to zero if neuron fires a spike in its output. This mechanism is illustrated in Figure 2.

Generally, training LIF neurons is done by treating the threshold (T) and decay (d) as non-trainable hyperparameters.

2.2 Propagation Delays in Neuron

As shown in Figure 3, a biological neuron has different components. Simply, nerve impulses are received by dendrites and processed by the nucleus. Impulses generated by the neuron travel through the axon and are distributed through synapses.

This process has two propagation delays: 1) *Axonal delay* that is the time required for an action potential to travel from soma to synapses through the axon. 2) *Synaptic delay* that is the time interval required for a neurotransmitter to be released from a presynaptic membrane distribute across the synaptic cleft and received by the post-synaptic membrane.

Supporting these propagation delays in implementation can increase the complexity of the design. Hence, only a few previous works support these delays (more detail in the next section).

2.3 Custom SNN Implementations

Analog [22, 23, 44], digital [2, 7, 9, 28, 30, 31, 35, 45], and mixed-analog-digital [3, 29] accelerators for SNNs have been described in the literature.

Analog realizations [22, 23, 44] are based on memristive technology [43] and have to deal with latency, density, and variability issues related to this technology [1]. In an other work [29], in addition to a memristive-based analog module, a digital module is used to route events and update receptive neurons. Neurogrid [3] does not use memristive technology in its analog module and increases

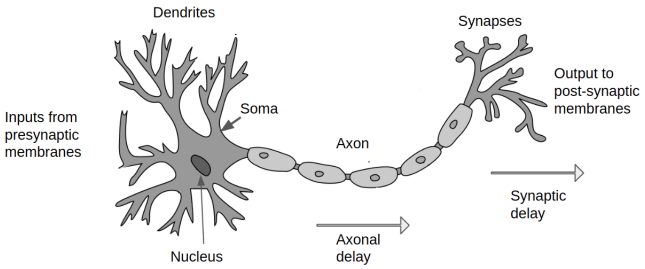


Figure 3: Illustration of a biological neuron. Dendrites receive inputs from presynaptic membranes to soma. The nucleus reacts to the received signals and may produce an action potential, which then has to go through the axon and distribute to post-synaptic membranes through Synapses.

parallelism by using a digital router for its events. In this work, we introduce a digital implementation for SNNs, and therefore we do not compare our work with analog realizations.

Large scale custom chip implementations such as Intel Loihi [9] with 4,096 on-chip cores and 1,024 neural units per core, SpiNNaker supercomputer [35] with 57,600 chips and 1,036,800 processors each capable of simulating 1,000 neurons, and IBM TrueNorth [2] with 4,096 cores and supporting one million neurons are designed with synaptic delay support. These implementations are designed to support a mesh of neurons with no particular topology. This is done by using advanced routers and schedulers. For example, Loihi uses six bits for the synaptic delay and two independent physical routing networks for core-to-core multicast. And events in SpiNNaker are coded to AER [26] packets (including timestamp, position, polarity, and debugging bits) and are source coded, meaning that the destination of each neuron has to be stored for routing the packets. TrueNorth has its own packet coding scheme, including the address of the core, axon index, tick number, and debugging flags. It buffers the events and uses a scheduler for processing events at specified ticks for supporting the synaptic delay.

Previous FPGA implementations of SNNs took a similar approach. BlueHive [28] is a 4-FPGA system and supports 64k Izhikevich [16] neurons per FPGA. BlueHive uses a routing system for events and 16 FIFOs for queuing events for 16 different synaptic delays with 1 millisecond granularity. Minitaur [31] encodes its events into five bytes, four bytes for timestamp and one byte for layer index. It supports a fixed axonal delay by buffering its events. In some other implementations routing and queuing is done without supporting synaptic or axonal delays [7, 11, 30, 45]. Because of queuing, parallelism in these works is done when an event is processed. Each event has a number of destinations, and upon processing an event, all of its destinations (membrane potentials) are incremented by their associated weights in parallel. Routers and schedulers are used to prevent deadlocks and data hazards while processing events from different queues with the same destinations. A comparison is provided at Table 1.

In the next section, we argue that by considering the network topology, for a feedforward network with interlayer connections, fixed-per-layer axonal and synaptic delays can be supported without extra FIFOs, schedulers, and separate routing networks.

Table 1: A comparison between S2N2 and previous works.

Architecture	Technology	Purpose	Supported Topology	Supported Propagation Delay	Required Complexity for supporting delay
Loihi [9]	custom chip	training and simulation	general mesh	synaptic	two separate physical routers
SpiNNaker [35]	custom chip	simulation	general mesh	synaptic	AER packets+router
TrueNorth [2]	custom chip	simulation	general mesh	synaptic	per-chip scheduler
BlueHive [28]	FPGA	simulation	general mesh	synaptic	16 FIFOs with 1ms granularity
Minitaur [31]	FPGA	accelerator	general mesh	fixed axonal	tick-batching and sorting
SpinalFlow [30]	FPGA	accelerator	feedforward	none	tick-batching (without supporting delays)
[45]	FPGA	accelerator	small and dense	none	N/A
[7]	FPGA	accelerator	feedforward	none	tick-batching (without supporting delays)
[11]	FPGA	accelerator	feedforward	none	tick-batching (without supporting delays)
[17]	FPGA	accelerator	feedforward	none	tick-batching (without supporting delays)
S2N2	FPGA	accelerator	feedforward	synaptic+axonal	streaming

3 STREAMING SPIKING NEURAL NETWORKS (S2N2)

To explain the streaming architecture of S2N2, we first look into the coding scheme used for storing events in input buffers. And explain the condition when a binary tensor can utilize less memory. We then explain how feedforward SNNs with interlayer connections can support fixed-per-layer synaptic and axonal delays without requiring schedulers and separate routing systems.

3.1 Input Buffer - Memory Utilization

As shown in Figure 1, a spiking input has a temporal duration with a total number of ticks (time units). In tick-batching, all the events for the entire duration of input are buffered and processed in a systolic implementation [30].

Let's look at the input events in a layer of a feedforward network. Assuming S being the total number of inputs to the layer, and T the total duration of the input, to encode events, we need $\log_2 S$ bits for addressing the position and $\log_2 T$ bits for addressing the tick number of each event. Assuming sparsity in the incoming events, the layer can receive up to $p * ST$ events when $p = 1 - sparsity\_ratio$ and $p \in (0, 1)$. Therefore we need a buffer of size:

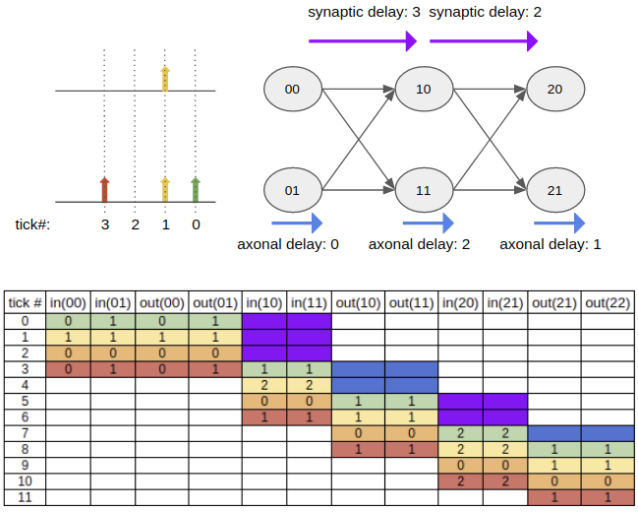
$$\text{buffer size in bits} = pST \log_2 ST \tag{5}$$

On the other hand, we can use a binary tensor to address the input events, ones for when there is an event, and zeros otherwise. In this case, we need ST-bits to store addresses in a binary tensor. Buffering encoded events requires less memory compared to a binary tensor if:

$$p \log_2 ST < 1 \tag{6}$$

This can be a tight condition on input's sparsity. E.g., for a layer with an input tensor of size $64 \times 16 \times 16$ with a total duration of 16 ticks, only for $p < \frac{1}{18} = 5.5\%$ or 94.5% sparsity for input, buffering encoded events uses less memory compared to a binary tensor of size 2^{18} bits. In this example, as soon as the input's sparsity drops below 94.5%, Equation 6 is not satisfied, and the binary tensor requires less memory. In Sections 4 and 5, we show that Equation 6 can not be satisfied for our applications.

3.2 Fixed-Per-Layer Propagation Delays



tick #	in(00)	in(01)	out(00)	out(01)	in(10)	in(11)	out(10)	out(11)	in(20)	in(21)	out(21)	out(22)
0	0	1	0	1								
1	1	1	1	1								
2	0	0	0	0								
3	0	1	0	1	1	1						
4					2	2						
5					0	0	1	1				
6					1	1	1	1				
7							0	0	2	2		
8							1	1	2	2	1	1
9									0	0	1	1
10									2	2	0	0
11											1	1

Figure 4: Top: A simple 3-layer network with fixed-per-layer axonal and synaptic delays. Inputs to each layer are binary vectors, and "1"s are for spikes. Input to a neuron is the sum of all inputs, and all weights are equal to one. Bottom: the flow of the input through the network. E.g., network input at tick=1 (color-coded) is received by both neurons in the first layer. With no axonal delay, they each produce one spike at their outputs at the same tick. The second layer receives this input (same color code) with a synaptic delay (3 ticks). At tick=4, both inputs to each neuron in the second layer have spikes. Hence their inputs are equal to 2.

As mentioned before, synaptic delays are realized in a limited number of previous work. Custom chips [2, 9, 35] queue their events

and use complex routing and scheduling systems to process events at the correct tick with an appropriate delay. In FPGA implementations, multiple FIFOs are used to support synaptic delays with large granularity (1 millisecond) [28]. These implementations support different topologies of spiking networks. And [31] supports feedforward networks with fixed axonal delays by buffering and sorting its encoded events.

Feedforward SNNs with interlayer connections have a specific topology that can be exploited for supporting fixed-per-layer synaptic and axonal delays with a reduced implementation cost. As shown in [49], temporal coding is still possible with fixed propagation delays. Figure 4 shows a simple 3-layer network with fixed-per-layer axonal and synaptic delays, meaning that all neurons in one layer have the same axonal delay and the same synaptic delay. For the sake of simplicity, neurons in this network spike if they receive an input larger than zero, and all weights are equal to one. Input to each neuron is the sum of all inputs.

The bottom part of Figure 4 shows how spikes spread through the network under axonal and synaptic delay conditions. Input to each layer is a binary vector, and spikes are represented by "1"s. Weights are equal to one, and input to a neuron is the sum of weights for connections with a spike. E.g. at tick=4, both inputs to neuron 10 have spikes and $\text{in}(10) = 2$. Previous works with propagation delay support [2, 9, 28, 31, 35] support this with different complexities (see Table 1).

tick #	in(00)	in(01)	out(00)	out(01)	in(10)	in(11)	out(10)	out(11)	in(20)	in(21)	out(21)	out(22)
0	0	1	0	1	1	1	1	1	2	2	1	1
1	1	1	1	1	2	2	1	1	2	2	1	1
2	0	0	0	0	0	0	0	0	0	0	0	0
3	0	1	0	1	1	1	1	1	2	2	1	1
4												
5												
6												
7												
8												
9												
10												
11												

Figure 5: With fixed-per-layer propagation delays in the example network shown in Figure 4, we can process inputs and outputs of all layers assuming no delay and push all the delays to the end. Then an accumulated delay (shown in purple and blue) can be added to the network output.

However, because of the network topology, we can process all the layers, assuming no propagation delay, and push all the delays to the end. Then a total delay equal to all accumulated delays can be applied to the network's output as shown in Figure 5.

This practice can be applied to any structured feedforward network with only interlayer connections. In this case, we can support both synaptic and axonal delays without schedulers, extra FIFOs, and sorting mechanisms used in previous works.

3.3 Architecture

The streaming architecture of S2N2 is designed based on the FINN framework [47]. In the following, we first describe FINN's approach to implementing non-spiking and conventional neural networks. We then describe our design to support the LIF model in FINN.

FINN framework: The original FINN paper [47] introduced a framework for building fast and flexible FPGA accelerators using a flexible heterogeneous streaming architecture. Exploiting a set of optimizations, FINN enables efficient mapping of binarized neural networks to hardware and supports fully connected, convolutional, and pooling layers. The second version of FINN described in [4], provides support for non-binary networks.

In the FINN architecture, a Sliding Window Unit (SWU) prepares the input by applying interleaving and implementing the image-to-column (im2col) algorithm. The output stream of a SWU feeds a Matrix Vector Threshold Unit (MVTU), which is the computational core for FINN's accelerator designs. This core is used in the implementations of both fully connected and convolution layers.

As shown in Figure 6, a MVTU has several Processing Elements (PE) that can generate output channels in parallel. Each PE has a number of SIMD lanes. If P_{FINN} be the number of PEs and S_{FINN} be the number of SIMD lanes, A P_{FINN}-high, S_{FINN}-wide tile matrix is processed at a time, inputs are mapped to different SIMD lanes and outputs are calculated in parallel by PEs. To accommodate this process, weights are also loaded from memory in tiles, and each PE takes a sub-tile of the weights to process its output.

All PE units have access to the input buffer inside the MVTU. The width of this buffer in bits is equal to the number of SIMD lanes multiplied by the activation bit width. For simplicity, only one row of this buffer is shown in Figure 6. The total number of rows in this buffer is equal to the ratio of *(kernel width × kernel height × #input channels)/#SIMD lanes*. Which makes the input buffer size equal to *(kernel width × kernel height × #input channels)* for 1-bit activation.

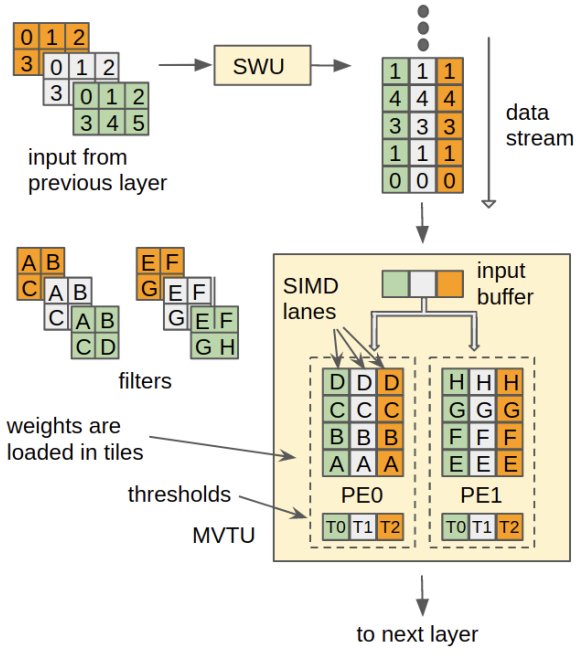


Figure 6: FINN [47] architecture. SWU interleaves the input by applying the image-to-column algorithm and feeds MVTU. Each PE inside MVTU processes one output channel and has a number of SIMD lanes that read from input channels and multiply the input by kernel weights in parallel.

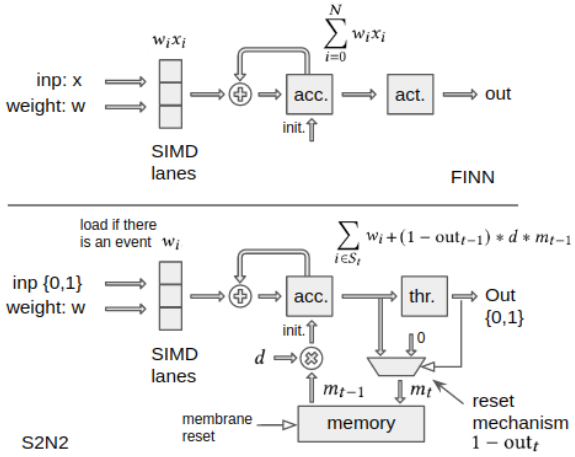


Figure 7: Upper: PE implementation in FINN [47]. Lower: PE architecture to support SNN in FINN.

The process flow of one PE is shown in (Figure 7.upper). The accumulator is initialized to a preset value (usually zero), then input and PE's sub-tile of weights are loaded into SIMD lanes. SIMD lanes execute Equation 1 and accumulate the results in the accumulator. After processing all the inputs for the current output, the accumulator value is passed through the activation for generating the output.

Implementing the LIF Model: S2N2's contribution to the FINN platform is by providing support for the LIF model in FINN's computational core, MVTU. In the following, we describe our design in detail and explain how to utilize FINN's architecture for initializing membrane potentials for each input. In the next section, we describe a possible optimization for decaying membrane potentials without a multiplication operation to maintain FINN's high throughput.

As mentioned in Section 3.1, if the condition in Equation 6 is not met, using a binary tensor for addressing events has a lower memory utilization than encoding events. In Sections 4 and 5, we show that this condition does not hold for example applications. Therefore, we use the input as is (a binary tensor) for addressing events. In addition, because FINN is not a systolic implementation, and the input is processed in a streaming architecture, the size of the input buffer used in the MVTU can be smaller than the input [47]. E.g., for an input of size $I_W \times I_H \times I_{Ch}$ with a kernel size of $K_W \times K_H$, and 1-bit activation, the buffer size is equal to $K_W \times K_H \times I_{Ch}$.

To support SNNs with the LIF model (Figure 7.lower), we initialize the accumulator by the previous membrane voltage stored in the on-chip memory, multiplied by the decay value, d. The SIMD lanes are programmed to use the input as the address of events and only load weights if there is a spike in that input position. This is exactly executing Equation 2. After executing all the operations required by Equation 2, the value stored in the accumulator is equal to Equation 3. This value can then be passed to a comparator to execute Equation 4. The result of this comparator is our output spike and is also used as the input to a selector for implementing the reset mechanism $(1 - \text{out}_t)$ and storing the correct membrane voltage back to the memory.

The MVTU in a FINN implementation has a control signal defining the number of runs per input. We use this signal to indicate the last tick for an input. This signal can be used to reset membrane potentials to zero if required.

In FINN, the MVTU is used for implementing both convolutional and fully connected layers. Similarly, with our proposed additions and modifications to the MVTU, both convolutional and fully connected layers for SNNs can be implemented with the MVTU shown in Figure 7.lower.

The pooling unit (PU) described in FINN [47] is a binary max-pooling layer. We chose to use binary tensors for addressing our spikes. Therefore we use the PU as is.

As illustrated in Figure 8, an event-based input has a temporal dimension that is divided into a number of ticks. To produce the classification output, the last layer in a SNN has one counter per label. Each counter keeps track of all the spikes received for that label. At the last tick for an input, the value of these counters can be fed to a function for determining the classification result (e.g., SoftMax). These counters are reset to zero at the last tick of each input.

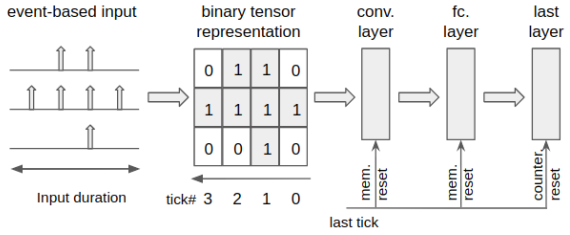


Figure 8: Binary tensor for addressing spikes in an event-based input. The last tick is used as a reset signal to reset membrane potentials (mem. reset) and counters at the last layer (counter reset).

As we explained earlier in this section, fixed-per-layer propagation delays in feedforward SNNs with only interlayer connections can be added as an accumulated delay to the output. Therefore, our proposed design can be used to implement such networks with fixed-per-layer axonal and synaptic delays. This design has no scheduler, and we do not queue encoded events. This gives us a number of advantages. 1) We can expand the parallelism of our design by processing events in parallel vs. sequential process in previous work. 2) By using a binary representation for addressing instead of addressing events by their position and tick number, we do not require a separate router. 3) we can support fixed-per-layer axonal and synaptic delays without a scheduler. 4) Addressing events with a binary tensor reduces our memory utilization when the condition in Equation 6 is not met.

4 S2N2 FOR AUTOMATIC MODULATION CLASSIFICATION

Deep learning for Radio Frequency (RF) applications is a relatively new field. In particular, using SNNs for RF applications is barely touched in the literature. One of the RF applications suitable for

ANNs is Automatic Modulation Classification (AMC). This important method can be used in radio fault detection, opportunistic mesh networking, dynamic spectrum access, and numerous regulatory and defense applications. Previous works have shown that ANNs can effectively perform modulation classification with high accuracy [24, 27, 32, 34].

This section introduces two new network architectures for AMC that are based on S2N2. The novelty of these architectures is that the input is fed to the network as a stream of events in the In-phase/Quadrature (I/Q) plane. To our knowledge, these are the only neural networks that consume a stream of RF samples as an event-based input. In the following, we describe the datasets used for training and explain our networks' architecture.

Datasets: We use two RF datasets to train our networks. *RadioML.2016* [32] is a collection of 11 different modulations (8PSK, AM-DSB, AM-SSB, BPSK, CPFSK, GFSK, PAM4, QAM16, QAM64, QPSK, and WBFM). Each class has samples recorded at 20 different Signal to Noise Ratio (SNR) levels (from -20dB to 18dB in increments of 2dB). Each pair {modulation, SNR} has 728 training examples, and Each training example is a time-series of 128 In-phase Quadrature (I/Q) sample pairs.

RadioML.2018 [34] is a collection of 24 different modulations (OOK, 4ASK, 8ASK, BPSK, QPSK, 8PSK, 16PSK, 32PSK, 16APSK, 32APSK, 64APSK, 128APSK, 16QAM, 32QAM, 64QAM, 128QAM, 256QAM, AM-SSBWC, AM-SSB-SC, AM-DSB-WC, AM-DSB-SC, FM, GMSK, and OQPSK). Each modulation class has samples recorded at 26 different SNR levels (from -20dB to 30dB in increments of 2dB). Each pair {modulation, SNR} has 4096 training examples and Each training example is a time-series of 1024 I/Q sample pairs. Both datasets are publicly available [2].

Two time-series examples from RadioML.2018 are shown in Figure 9.left. These examples are 1024 I/Q sample pairs. In all of the previous work, inputs are tensors with same shape as these examples. E.g., for RadioML.2016, inputs are 2×128 float tensors, and in RadioML.2018, inputs are 2×1024 float tensors.

S2N2 is not a systolic implementation. Meaning, we can feed the network with a stream of events. Therefore, in our networks, we use the constellation of signals (shown in Figure 9.middle), and at each tick, we feed the network with one sample (Figure 9.right). Therefore our input is a stream of binary tensors.

To our knowledge feeding RF samples as events to a neural network has never been done before. The only work on using SNNs for AMC is a preliminary investigation done by NASA [19] that implements a two-layer SNN in MATLAB for classifying three noise-free modulations (BPSK, QPSK, and 8PSK). In NASA's work, inputs are 8-bit images of constellations.

Feeding a neural network with RF samples as events come with two benefits. 1) Although we and all the previous works use recorded data, in a real-world setup, our network can consume RF samples one-by-one in a stream. Other works have to buffer samples (e.g., 128 or 1024 samples) before taking them as input. 2) We can aggressively quantize the I/Q plane; therefore the input size (in bits) can get smaller. The following explains the I/Q plane quantization.

Examples in RadioML.2016 and RadioML.2018 are 128 and 1024 pairs of float numbers, respectively. We construct the I/Q plane by quantizing the pair using a uniform quantizer. This will reduce our input size. Figure 10 illustrates three examples from RadioML.2018: OOK, 64QAM, and 32PSK classes at 30db, 16dB, and 2dB SNR, respectively. To the right of these examples, their constellations with quantized in-phase and quadrature values are shown. As it is shown, the shapes of the constellations are recognizable even at the lowest bit resolution. We used the 4-bit quantized constellations to train our networks.

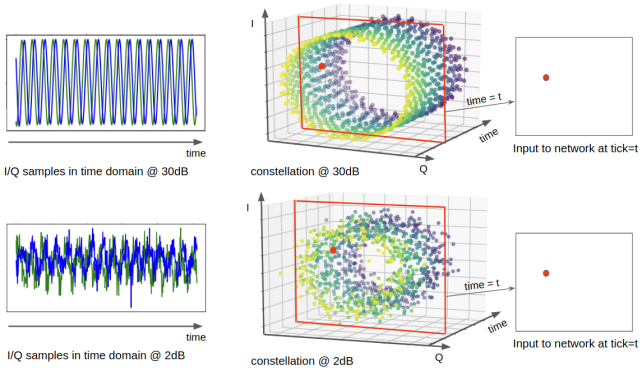


Figure 9: Examples of AM-DSB class from RadioML dataset [34]. On the left, two examples of AM-DSB I/Q samples are shown at 30dB and 2dB SNR at top and bottom, respectively. The middle illustrates the constellations of the same examples. On the right, input to the network at time (tick)=t is shown. Input to our networks are samples as events in I/Q plane.

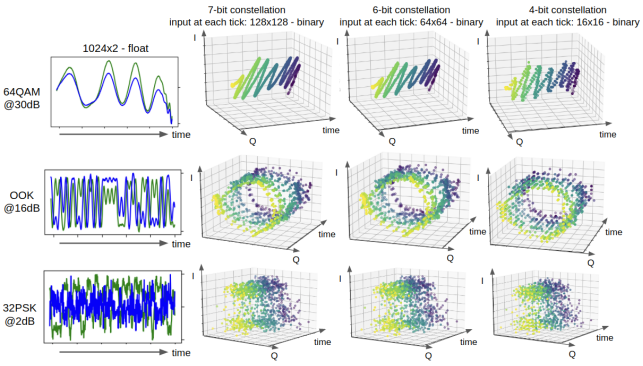


Figure 10: Applying quantization to the I/Q plane. Original examples from RadioML.2018 [34] dataset are 1024 pairs of float numbers (left). In-phase and Quadrature values can be quantized for a smaller input tensor (first three columns from the right). At each tick, we feed one slice of the quantized constellation tensor to the network. The figure shows that the constellation shape is recognizable while I/Q plane is aggressively quantized.

[2] https://www.deepsig.ai/datasets

Network Architecture for RadioML.2016

This network is a four layer architecture similar to the network described in [32] with different number of kernels and LIF model for activation (Figure 11).

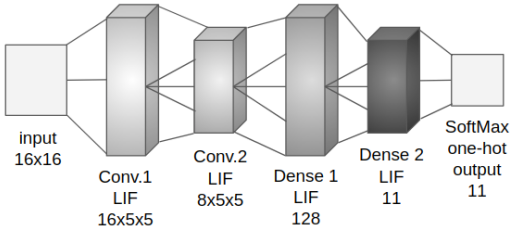


Figure 11: S2N2_rf1 architecture.

Inputs to each layer are binary tensors. We used 90% of the dataset for training and 10% for validation. For training, we used the method described in [37] as our baseline and changed the loss function to *smooth L1 loss*, and adjusted the hyperparameters. Throughout this paper, we refer to this network as *S2N2_rf1*.

Table 2: Comparing validation accuracy and network size for S2N2_rf1.

Network	Input	Conv.1	Conv.2	Dense 1	Dense 2	Accuracy
[32]	128x2 32-bit	64x1x3	16x2x3	128	11	87.4%
S2N2_rf1	16x16 binary	16x5x5	8x5x5	128	11	91.7%

We achieved 91.7% Top-1 and 100% Top-5 validation accuracy using all SNR levels in our training. A comparison between S2N2_rf1's size and accuracy with the previous work on RadioML.2016 is provided in Table 2.

Figure 12 illustrates the spike ratio in the input of each layer for S2N2_rf1. The first convolution layer (Conv.1) receives one event at each tick; this means that the spike ratio for this layer with an input of size 16×16 is $\frac{1}{16 \times 16} = 0.0039$.

Table 3: Required memory for buffering input at each layer of S2N2_rf1 is compared with tick-batching (Equation 5).

Layer	#Ticks	Input Size	Maximum Spike Ratio	Buffer Size Tick-Batching	Buffer Size S2N2_rf1	Improvement
Conv.1	128	16×16	0.39%	1,917 bits	25 bits	×77
Conv.2	128	16×16×16	7%	697,304 bits	400 bits	×1,744
Dense 1	128	8×12×12	8%	212,337 bits	128 bits	×1,658
Dense 2	128	128	12%	27,526 bits	11 bits	×2,502

As mentioned in Sections 2.3 and 3.1, previous works have used tick-batching and buffered encoded events. This means that for a total number of ticks=128, and input size of 16×16 at spike ration of 0.39%, according to Equation 5, tick-batching requires 1,917 bits to queue the input events. Because S2N2 is based on the streaming

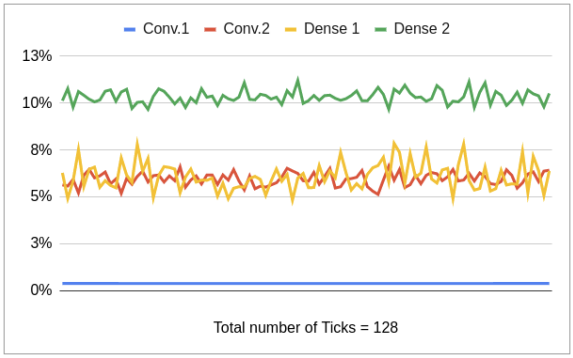


Figure 12: The ratio of spiking neurons in input to each layer of S2N2_rf1. Ratios are collected during classifying one input (128 ticks) with trained weights.

architecture of FINN [47], and only a portion of the input is buffered for processing. The size of this buffer used in MVTU is equal to *kernel size* × *#input channels*=25 bits for Conv.1 layer. In Table 3, we provide the same comparison for all the layers of this network. These results show that, on average, memory utilization for input buffers in S2N2_rf1 is improved by over three orders of magnitude.

Network Architecture for RadioML.2018

As mentioned in our introduction, training methods for spiking neural networks are not as mature as other ANNs. In particular, current training methods are evaluated on smaller networks, and simple datasets [11] and perform poorly when used for training very deep architectures [18, 40] and evaluated on more complex datasets [38]. Therefore we could not train a deep spiking network similar to the non-spiking networks used in previous works (VGG10 and Resnet33) [34, 46]. Instead, we chose a smaller network with only eight layers. We refer to this network as *S2N2_rf2*.

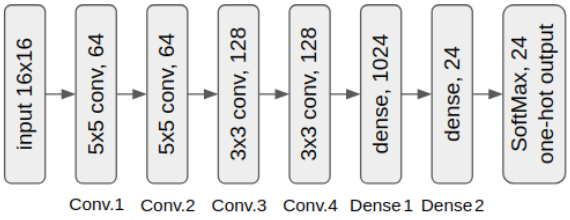


Figure 13: S2N2_rf2 architecture.

S2N2_rf2 architecture is shown in Figure 13. We used the same training script like the one we used for training S2N2_rf1 as the baseline. We then adjusted the script for the dataset and its increased number of labels.

This network can achieve 68.5% Top-1 and 95% Top-5 validation accuracy on 24 classes in RadioML.2018 dataset. Table 4 compares our accuracy with two related non-spiking networks.

Although that S2N2_rf2 does not have a high accuracy compared to deeper and non-spiking networks, it is included in this work to provide a comparison between S2N2 architecture and tick-batching

with regards to memory utilization. In particular, when larger RF inputs are used.

Table 4: Comparing validation accuracy and network size for S2N2_rf2.

Network	Input	#Layers	Accuracy
ResNet [34]	1024x2 (32-bit)	33	95.5%
VGG [34]	1024x2 (32-bit)	10	88.0%
S2N2_rf2	16x16 (binary)	6	68.5%

Figure 14 shows spike rations at the input of each layer of S2N2_rf2. These ratios are similar to the ratios in S2N2_rf1 (Figure 12). We expect that with future improvements in training methods for deeper SNNs, similar spike ratios with no significant reductions will hold for a spiking network with a higher accuracy.

We use these ratios to show the efficiency of S2N2 for reducing the input buffer size at each layer. Even if our assumption does not hold, and in the future networks with lower spike ratios provide a higher accuracy, S2N2 is still more efficient at the minimum possible spike ratio; only one spike at layer's input (first row in Tables 3 and 5).

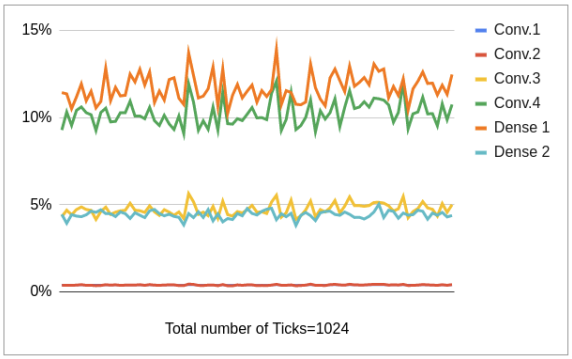


Figure 14: The ratio of spiking neurons in input to each layer of S2N2_rf2. Ratios are collected during classifying one input (1024 ticks) with trained weights.

Table 5 illustrates a comparison between input buffer sizes required for S2N2 and tick-batching. Equation 5 is used to calculate the buffer size for tick-batching. It is clear that for inputs with large temporal dimension, using a streaming architecture significantly reduces the memory utilization.

Synthesis Results

We used Vivado-HLS™tool for evaluating S2N2_rf1 and S2N2_rf2 network architectures. To increase the throughput and reduce our DSP utilization, we used fixed-points for our parameters and trained both networks with a decay factor equal to 0.875 (d in Equation 3). This way, $d \times m_{t-1}$ in Figure 7 can be replaced by $(m_{t-1} - m_{t-1} >> 3)$.

Table 5: Required memory for buffering input at each layer of S2N2_rf2 is compared with tick-batching (Equation 5).

Layer	#Ticks	Input Size	Maximum Spike Ratio	Buffer Size Tick-Batching	Buffer Size S2N2_rf2	Improvement
Conv.1	1024	16×16	0.39%	18,403 bits	25 bits	×737
Conv.2	1024	16×16×64	0.5%	2,013,266 bits	1,600 bits	×1,258
Conv.3	1024	12×12×64	6%	13,589,545 bits	576 bits	×23,592
Conv.4	1024	10×10×128	12%	37,748,736 bits	1,152 bits	×32,768
Dense 1	1024	10×10×128	14%	44,040,192 bits	1,024 bits	×43,008
Dense 2	1024	1024	5%	1,048,576 bits	24 bits	×43,690

We could fit S2N2_rf1 (smaller network) on a ZYNQ chip similar to the one used in the PYNQ development board. Because of the large size of S2N2_rf2, we selected the ZCU111 development board in our synthesis. This board is also used for implementing a non-spiking network for the same dataset [46].

Table 6: Synthesis results for S2N2_rf1 and S2N2_rf2 network architectures.

Network	Board	BRAM_18K	DSP48E	FF	LUT	Tick Resolution
S2N2_rf1	PYNQ	29%	5%	11%	52%	45 ns
S2N2_rf2	ZCU111	98%	<1%	4%	24%	30 ns

Our results are shown in Table 6. The high BRAM utilization is due to the required memory for storing membrane potentials. Tick resolution indicates how fast RF samples can be consumed by the network. E.g., at each second, S2N2_rf1 can classify 173.6k examples from the RadioML.2016 dataset (each example requires 128 ticks). And S2N2_rf2 can process 32.5k examples from the RadioML.2018 dataset (each example requires 1024 ticks).

5 IMAGE CLASSIFICATION ON S2N2

In this section, we provide an example network for image classification on MNIST dataset [3]. We used the method provided by DECOLLE [18] to convert MNIST dataset to trains of spikes [4]. We used a four-layer convolutional network similar to the one described in DECOLLE as our baseline.

Figure 15.left shows the network structure we used for image classification. We refer to this network as S2N2_cv. We applied two modifications to the original structure. First, as shown in the figure, in the original network, convolutional filters are applied to the membrane voltage. This means MAC operations similar to Equation 1. We modified the layer, and instead, we apply the convolutional filters on the input to have the sparse accumulations similar to Equation 2. Second, the neuron model used in the original work is a LIF model with two internal variables. We changed the model to the one variable LIF model described in Section 2.1.

S2N2_cv is trained with the original training script [4], and adjusted hyperparameters. It can achieve competitive results compared to other works (see Table 7).

[3] http://yann.lecun.com/exdb/mnist/
[4] https://github.com/nmi-lab/dcll

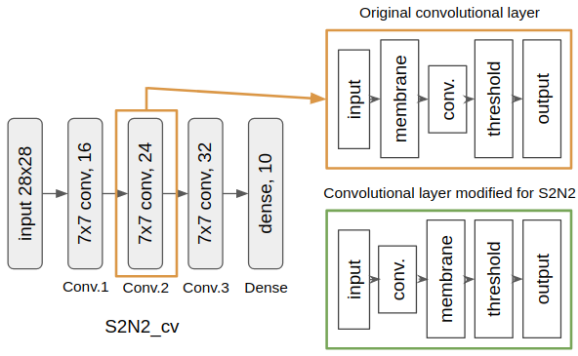


Figure 15: S2N2_cv structure. On the left, four-layer structure of the network. Orange box, original organization of one convolutional layer. Green box, convolutional layer modified for S2N2.

Table 7: Accuracy result of S2N2_cv on MNIST compared to similar SNNs.

Network	Architecture	Validation Accuracy
S2N2_cv	28x28-16c7-24c7-32c7-10	98.5%
[18]	28x28-16c7-24c7-32c7-10	98.0%
[40]	28x28-12c5-2a-64c5-2a-10c	99.3%

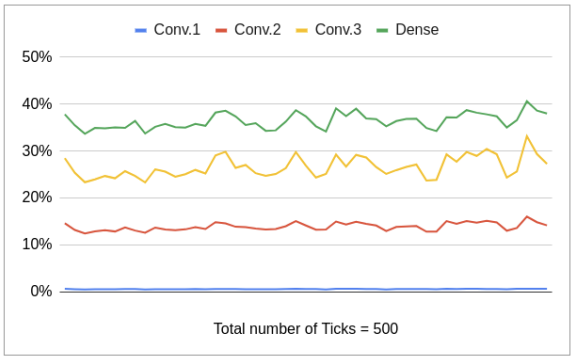


Figure 16: The ratio of spiking neurons in input to each layer of S2N2_cv. Ratios are collected during classifying one input (500 ticks) with trained weights.

Figure 16 shows the spike ratios at the input of each layer in S2N2_cv. The method for converting MNIST data to trains of spikes used in [18] converts each image to a $28 \times 28 \times 500$ binary tensor of spikes. Unlike RF samples, input to the first layer can have more than one spike at each tick; therefore, for vision applications, the input is less sparse. Consequently, the ratio of spikes at each layer is higher than the layers in S2N2_rf1 and S2N2_rf2.

With higher spike ratios, the buffer size for storing encoded events in tick-batching rapidly grows. While the buffer size used in S2N2 is independent of the input's spike ratio. Table 8 shows a comparison between these two buffer sizes for S2N2_cv network.

Table 8: Required memory for buffering input at each layer of S2N2_cv is compared with tick-batching (Equation 5).

Layer	#Ticks	Input Size	Maximum Spike Ratio	Buffer Size Tick-Batching	Buffer Size S2N2_cv1	Improvement
Conv.1	500	28×28	0.7%	5,2136 bits	49 bits	×1,064
Conv.2	500	16×13×13	16%	4,542,720 bits	784 bits	×5,794
Conv.2	500	24× 11 ×11	33%	10,062,360 bits	1,176 bits	×8,556
Dense	500	32×4×4	41%	1,889,280 bits	10 bits	×188,928

Synthesis Results

S2N2_cv is evaluated with Vivado-HLS™tool. This network is relatively small, and we can fit it on the PYNQ development board. Our results are shown in Table 9.

To reduce our DSP utilization, we took a similar approach as what we did for training our two other networks and trained S2N2_cv with a decay factor equal to 0.875 (d in Equation 3). This way, $d \times m_{t-1}$ in Figure 7 is replaced with a shift and one subtractions ($m_{t-1} - m_{t-1} >> 3$).

Table 9: Synthesis results for S2N2_cv network architecture.

Network	Board	BRAM_18K	DSP48E	FF	LUT	Tick Resolution
S2N2_cv	PYNQ	35%	<1%	2%	6%	30 ns

The high BRAM utilization in Table 9 is because of the memory required to store membrane potentials for the LIF model (Figure 7).

6 CONCLUSION

In this work, we introduced a streaming accelerator for spiking neural networks, namely S2N2. We showed that in batch-ticking, the buffer size used for storing encoded events depends on the input's spike ratios. This method is used in previous work, assuming a low spike ratio in the input. We showed that this assumption could be a tight condition on input's spike ratio. We then described how a binary tensor could address events and confirmed that a binary tensor with our streaming architecture requires less memory in our example applications.

We also described how to efficiently support axonal and synaptic delays in a feedforward SNN with only interlayer connections. By using binary tensors as inputs, we built our architecture upon FINN platform. We provided support for the LIF model in FINN and optional initialization of membrane potentials for each input to support SNNs in FINN.

Our streaming SNN architecture is suitable for processing signals of large temporal dimension. Two novel SNN architectures for AMC are introduced in this work. In addition, an example of image classification on a SNN is described. All example applications are evaluated with the Vivado-HLS™tool. Our results achieve a minimum tick-resolution of 30 ns. S2N2 reduces input buffers' memory utilization by more than three orders of magnitude.

REFERENCES

[1] Gina Adam, Ali Khiat, and Themis Prodromakis. 2018. Challenges hindering memristive neuromorphic hardware from going mainstream. In *Nature Communications*, Vol. 9. https://doi.org/10.1038/s41467-018-07565-4

[2] F. Akopyan, J. Sawada, A. Cassidy, R. Alvarez-Icaza, J. Arthur, P. Merolla, N. Imam, Y. Nakamura, P. Datta, G. Nam, B. Taba, M. Beakes, B. Brezzo, J. B. Kuang, R. Manohar, W. P. Risk, B. Jackson, and D. S. Modha. 2015. TrueNorth: Design and Tool Flow of a 65 mW 1 Million Neuron Programmable Neurosynaptic Chip. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 34, 10 (2015), 1537–1557.

[3] B. V. Benjamin, P. Gao, E. McQuinn, S. Choudhary, A. R. Chandrasekaran, J. Bussat, R. Alvarez-Icaza, J. V. Arthur, P. A. Merolla, and K. Boahen. 2014. Neurogrid: A Mixed-Analog-Digital Multichip System for Large-Scale Neural Simulations. *Proc. IEEE* 102, 5 (2014), 699–716.

[4] Michaela Blott, Thomas B. Preußer, Nicholas J. Fraser, Giulio Gambardella, Kenneth O'brien, Yaman Umuroglu, Miriam Leeser, and Kees Vissers. 2018. FINN-R: An End-to-End Deep-Learning Framework for Fast Exploration of Quantized Neural Networks. *ACM Trans. Reconfigurable Technol. Syst.* 11, 3, Article 16 (Dec. 2018), 23 pages. https://doi.org/10.1145/3242897

[5] William Chan, Navdeep Jaitly, Quoc Le, and Oriol Vinyals. 2015. Listen, attend, and spell. *IEEE International Conference on Acoustic, Speech, and Signal Processing (ICASSP)* (2015).

[6] Y. Chen, T. Krishna, J. S. Emer, and V. Sze. 2017. Eyeriss: An Energy-Efficient Reconfigurable Accelerator for Deep Convolutional Neural Networks. *IEEE Journal of Solid-State Circuits* 52, 1 (2017), 127–138.

[7] Kit Cheung, Simon R Schultz, and Wayne Luk. 2012. A large-scale spiking neural network accelerator for FPGA systems. In *International Conference on Artificial Neural Networks*. Springer, 113–120.

[8] I. M. Comsa, K. Potempa, L. Versari, T. Fischbacher, A. Gesmundo, and J. Alakuijala. 2020. Temporal Coding in Spiking Neural Networks with Alpha Synaptic Function. In *ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 8529–8533.

[9] M. Davies, N. Srinivasa, T. Lin, G. Chinya, Y. Cao, S. H. Choday, G. Dimou, P. Joshi, N. Imam, S. Jain, Y. Liao, C. Lin, A. Lines, R. Liu, D. Mathaikutty, S. McCoy, A. Paul, J. Tse, G. Venkataramanan, Y. Weng, A. Wild, Y. Yang, and H. Wang. 2018. Loihi: A Neuromorphic Manycore Processor with On-Chip Learning. *IEEE Micro* 38, 1 (2018), 82–99.

[10] Steven K. Esser, Paul A. Merolla, John V. Arthur, Andrew S. Cassidy, Rathinakumar Appuswamy, Alexander Andreopoulos, David J. Berg, Jeffrey L. McKinstry, Timothy Melano, Davis R. Barch, Carmelo di Nolfo, Pallab Datta, Arnon Amir, Brian Taba, Myron D. Flickner, and Dharmendra S. Modha. 2016. Convolutional networks for fast, energy-efficient neuromorphic computing. *Proceedings of the National Academy of Sciences* 113, 41 (2016), 11441–11446. https://doi.org/10.1073/pnas.1604850113

[11] H. Fang, Z. Mei, A. Shrestha, Z. Zhao, Y. Li, and Q. Qiu. 2020. Encoding, Model, and Architecture: Systematic Optimization for Spiking Neural Network in FPGAs. In *2020 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*. 1–9.

[12] Samanwoy Ghosh-Dastidar and Hojjat Adeli. 2009. Third Generation Neural Networks: Spiking Neural Networks. In *Advances in Computational Intelligence*. Springer Berlin Heidelberg, Berlin, Heidelberg, 167–178.

[13] Alex Graves and Jaitly Navdeep. 2014. Towards end-to-end speech recognition with recurrent neural networks. In *2014 International Conference on Machine Learning (ICML)*, Vol. 14.

[14] Alan Hodgkin and Andrew Huxley. 1952. A quantitative description of membrane current and its application to conduction and excitation in nerve. In *The Journal of physiology*, Vol. 117. 500. Issue 4. https://doi.org/10.1113/jphysiol.1952.sp004764

[15] Zan Huang, Hsinchun Chen, Chia jung Hsu, Wun hwa Chen, and Soushan Wu. 2004. Credit Rating Analysis With Support Vector Machines and Neural Networks: A Market Comparative Study.

[16] Eugene. Izhikevich. 2003. Simple model of spiking neurons. In *Transactions on Neural Networks, IEEE*, Vol. 14. 1569 – 1572.

[17] X. Ju, B. Fang, R. Yan, X. Xu, and H. Tang. 2020. An FPGA Implementation of Deep Spiking Neural Networks for Low-Power and Fast Classification. *Neural Computation* 32, 1 (2020), 182–204. https://doi.org/10.1162/neco_a_01245

[18] Jacques Kaiser, Hesham Mostafa, and Emre Neftci. 2020. Synaptic Plasticity Dynamics for Deep Continuous Local Learning (DECOLLE). *Frontiers in Neuroscience* 14 (2020), 424. https://doi.org/10.3389/fnins.2020.00424

[19] E. J. Knoblock and H. R. Bahrami. 2019. Investigation of Spiking Neural Networks for Modulation Recognition using Spike-Timing-Dependent Plasticity. In *2019 IEEE Cognitive Communications for Aerospace Applications Workshop (CCAAW)*. 1–5.

[20] Chang W. Lee and Jung-A Park. 2001. Assessment of HIV/AIDS-related health performance using an artificial neural network. *Information & Management* 38, 4 (2001), 231 – 238. https://doi.org/10.1016/S0378-7206(00)00068-9

[21] Jun Haeng Lee, Tobi Delbruck, and Michael Pfeiffer. 2016. Training Deep Spiking Neural Networks Using Backpropagation. *Frontiers in Neuroscience* 10 (2016), 508. https://doi.org/10.3389/fnins.2016.00508

[22] Beiye Liu, Yiran Chen, Btyant Wysocki, and Tingwen Huang. 2015. Reconfigurable Neuromorphic Computing System with Memristor-Based Synapse Design. *Neural Processing Letters* 41 (2015), 159–167. https://doi.org/10.1007/s11063-013-9315-8

[23] C. Liu, B. Yan, C. Yang, L. Song, Z. Li, B. Liu, Y. Chen, H. Li, Qing Wu, and Hao Jiang. 2015. A spiking neuromorphic design with resistive crossbar. In *2015 52nd ACM/EDAC/IEEE Design Automation Conference (DAC)*. 1–6.

[24] X. Liu, D. Yang, and A. E. Gamal. 2017. Deep neural network architectures for modulation classification. In *2017 51st Asilomar Conference on Signals, Systems, and Computers*. 915–919.

[25] Wolfgang Maass. 1997. Networks of spiking neurons: the third generation of neural network models. In *1997 Neural Networks*, Vol. 10. 1659–1671.

[26] Misha Mahowald. 1994. *An analog VLSI system for sterescopic vision.* Kluwer, Boston, MA.

[27] G. J. Mendis, J. Wei, and A. Madanayake. 2016. Deep learning-based automated modulation classification for cognitive radio. In *2016 IEEE International Conference on Communication Systems (ICCS)*. 1–6.

[28] Simon W Moore, Paul J Fox, Steven JT Marsh, A Theodore Markettos, and Alan Mujumdar. 2012. Bluehive-a field-programable custom computing machine for extreme-scale real-time neural network simulation. In *2012 IEEE 20th International Symposium on Field-Programmable Custom Computing Machines*. IEEE, 133–140.

[29] Surya Narayanan, Ali Shafiee, and Rajeev Balasubramonian. 2017. INXS: Bridging the throughput and energy gap for spiking neural networks. In *2017 International Joint Conference on Neural Networks, IJCNN 2017, Anchorage, AK, USA, May 14-19, 2017*. IEEE, 2451–2459. https://doi.org/10.1109/IJCNN.2017.7966154

[30] Surya Narayanan, Karl Taht, Rajeev Balasubramonian, Edouard Giacomin, and Pierre-Emmanuel Gaillardon. 2020. SpinalFlow: An Architecture and Dataflow Tailored for Spiking Neural Networks. In *2020 47th International Symposium on Computer Architecture (ISCA-47)*.

[31] Daniel Neil and Shih-Chii Liu. 2014. Minitaur, an event-driven FPGA-based spiking network accelerator. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 22, 12 (2014), 2621–2628.

[32] Timothy J. O'Shea, Johnathan Corgan, and T. Charles Clancy. 2016. Convolutional Radio Modulation Recognition Networks. In *Engineering Applications of Neural Networks*. Springer International Publishing, Cham, 213–226.

[33] T. J. O'Shea, T. Roy, and T. C. Clancy. 2018. Over-the-Air Deep Learning Based Radio Signal Classification. *IEEE Journal of Selected Topics in Signal Processing* 12, 1 (2018), 168–179.

[34] T. J. O'Shea, T. Roy, and T. C. Clancy. 2018. Over-the-Air Deep Learning Based Radio Signal Classification. *IEEE Journal of Selected Topics in Signal Processing* 12, 1 (2018), 168–179.

[35] E. Painkras, L. A. Plana, J. Garside, S. Temple, F. Galluppi, C. Patterson, D. R. Lester, A. D. Brown, and S. B. Furber. 2013. SpiNNaker: A 1-W 18-Core System-on-Chip for Massively-Parallel Neural Network Simulation. *IEEE Journal of Solid-State Circuits* 48, 8 (2013), 1943–1953.

[36] Priyadarshini Panda and Kaushik Roy. 2016. Unsupervised regenerative learning of hierarchical features in spiking deep networks for object recognition. In *International Joint Conference on Neural Networks (IJCNN)*. 299 – 306. https://doi.org/10.1109/IJCNN.2016.7727212

[37] Ali Samadzadeh, Fatemeh Sadat Tabatabaei Far, Ali Javadi, Ahmad Nickabadi, and Morteza Haghir Chehreghani. 2020. Convolutional Spiking Neural Networks for Spatio-Temporal Feature Extraction. arXiv:2003.12346 [cs.CV]

[38] Abhronil Sengupta, Yuting Ye, Robert Wang, Chiao Liu, and Kaushik Roy. 2019. Going Deeper in Spiking Neural Networks: VGG and Residual Architectures. *Frontiers in Neuroscience* 13 (2019), 95. https://doi.org/10.3389/fnins.2019.00095

[39] Pierre Sermanet, David Eigen, Xiang Zhang, Michael Mathieu, Rob Fergus, and Yann LeCun. 2014. OverFeat: Integrated Recognition, Localization and Detection using Convolutional Networks. In *2014 International Conference on Learning Representations (ICLR)*.

[40] Sumit Bam Shrestha and Garrick Orchard. 2018. SLAYER: Spike Layer Error Reassignment in Time. In *Advances in Neural Information Processing Systems 31*, S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett (Eds.). Curran Associates, Inc., 1419–1428. http://papers.nips.cc/paper/7415-slayer-spike-layer-error-reassignment-in-time.pdf

[41] Karen Simonyan and Andrew Zisserman. 2014. Very Deep Convolutional Networks for Large-Scale Image Recognition. arXiv:1409.1556 [cs.CV]

[42] James Smith. [n.d.]. *A Roadmap for Reverse-Architecting the Brain's Neocortex*. Federated Computing Research Conference (FCRC). https://iscaconf.org/isca2019/slides/JE_Smith_keynote.pdf

[43] Dmitri Strukov, Gregory Snider, Duncan Stewart, and Stanley Williams. 2008. The missing memristor found. In *Nature*, Vol. 453. 80–83. https://doi.org/10.1038/nature06932

[44] T. Tang, L. Xia, B. Li, R. Luo, Y. Chen, Y. Wang, and H. Yang. 2015. Spiking neural network with RRAM: Can we use it for real-world application?. In *2015 Design, Automation Test in Europe Conference Exhibition (DATE)*. 860–865.

[45] David Thomas and Wayne Luk. 2009. FPGA accelerated simulation of biologically plausible spiking neural networks. In *2009 17th IEEE symposium on field*

programmable custom computing machines. IEEE, 45–52.

[46] S. Tridgell, D. Boland, P. H. W. Leong, R. Kastner, A. Khodamoradi, and Siddhartha. 2020. Real-time Automatic Modulation Classification using RFSoC. In *2020 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*. 82–89.

[47] Yaman Umuroglu, Nicholas J. Fraser, Giulio Gambardella, Michaela Blott, Philip Leong, Magnus Jahre, and Kees Vissers. 2017. FINN: A Framework for Fast, Scalable Binarized Neural Network Inference. In *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays* (Monterey, California, USA) *(FPGA '17)*. Association for Computing Machinery, New York, NY, USA, 65–74. https://doi.org/10.1145/3020078.3021744

[48] S. I. Venieris and C. Bouganis. 2016. fpgaConvNet: A Framework for Mapping Convolutional Neural Networks on FPGAs. In *2016 IEEE 24th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. 40–47.

[49] Thomas Voegtlin. 2006. Temporal Coding Using the Response Properties of Spiking Neurons. In *Proceedings of the 19th International Conference on Neural Information Processing Systems (NIPS'06)*. MIT Press, Cambridge, MA, USA, 1457–1464.

[50] Friedemann Zenke and Surya Ganguli. 2018. SuperSpike: Supervised Learning in Multilayer Spiking Neural Networks. *Neural Computation* (2018), 1514–1541. https://doi.org/10.1162/neco_a_01086