SyncNN: Evaluating and Accelerating Spiking Neural Networks on FPGAs

Sathish Panchapakesan*, Zhenman Fang*, Jian Li†
* School of Engineering Science, Simon Fraser University, Canada
† Futurewei Technologies, Inc, USA
{sathishp, zhenman}@sfu.ca jian.li@futurewei.com

Abstract—**In this paper, we propose a novel synchronous approach for rate encoding based Spiking Neural Networks (SNNs), which is more hardware friendly than conventional asynchronous approaches. We also design and implement the SyncNN framework to accelerate SNNs on Xilinx ARM-FPGA SoCs in a synchronous fashion. To improve the computation and memory access efficiency, we first quantize the network weights to 16-bit, 8-bit, and 4-bit fixed-point values with the SNN friendly quantization technique. For the encoded neurons that have dynamic and irregular access patterns, we design parameterized compute engines to accelerate their performance on the FPGA, where we explore various parallelization strategies and memory access optimizations. Our experimental results on multiple Xilinx ARM-FPGA SoC boards demonstrate that our SyncNN is scalable to run multiple networks, such as LeNet, Network in Network, and VGG, on various datasets such as MNIST, SVHN, and CIFAR-10. SyncNN not only achieves competitive accuracy (99.6%) but also achieves state-of-the-art performance (13,086 frames per second) for the MNIST dataset.**

I. Introduction

Spiking Neural Networks (SNNs), often referred to as the third-generation Neural Networks (NNs), have attracted increasing attention because they are more biologically plausible and have more potential for hardware acceleration [1]. SNNs process spikes based on the membrane potential of the neurons and are modeled in accordance to the actual neural system of the human brain [2]. Compared to artificial NNs (ANNs)—such as the widely used convolutional NNs (CNNs)—where all neurons in each layer are activated and computed, SNNs only activate those neurons whose membrane potential exceeds the threshold potential [1]. This event-driven nature greatly reduces the computation and communication between neurons.

The process of representing neural spikes (i.e., information) in SNNs is called neural encoding and there are two major approaches: rate encoding and temporal encoding. Temporal encoding requires spike based training mechanisms to train the time interval between spikes along with the network parameters [3]–[8]. However, the temporal information processing capability limits the scope of exploring SNNs only for shallow networks [9]. On the other hand, rate encoding—where the number of spikes in an encoding window is proportional to the numerical value to be encoded—does not need any additional trained information and allows to directly convert the trained ANN models to SNN fashion for evaluation [10]–[13]. More recently, the CNN-to-SNN converted models have achieved very good accuracy for deep networks such as VGG and ResNet [9]. In this work, we focus on the conversion based rate encoding SNNs to explore deeper networks on FPGAs.

In rate encoding SNNs, the input information (e.g., image pixels) is converted to spikes for an encoding window (i.e., multiple timesteps). At each timestep, the spikes carry the information along the layers in the network. The default asynchronous execution flow of SNNs enables the hardware to run all the layers concurrently in a pipeline fashion at any timestep [14]. However, the asynchronous execution also poses great challenges for implementing deeper networks on hardware. First, the network parameters have to be on chip for all the layers, which makes it not feasible to run deeper networks on edge devices. Second, the layer with the highest workload becomes the bottleneck and the resources allocated to other layers in the network remain idle for most of the time. Lastly, for deeper networks, the encoding window grows much larger to achieve the required accuracy, and thus running the entire network for many timesteps could make the SNN even slower than the original ANN.

In this paper, we propose a novel synchronous SNN, called *SyncNN*, to overcome those challenges. Instead of running the entire network for multiple timesteps, we only run the input layer—that converts the inputs to spikes—for multiple timesteps, and encode the number of spikes for each neuron. After that, we compute the remaining layers of the network in a synchronous layer-by-layer fashion, for just one timestep. SyncNN preserves the event-driven feature of SNNs by only activating those neurons whose aggregated membrane potential exceeds the threshold; for each spiked neuron, we also encode the number of effective spikes based on its aggregated membrane potential. SyncNN achieves the same accuracy as the asynchronous approach, addresses the aforementioned challenges, and opens more opportunities for hardware acceleration.

To achieve real-time SNN inference, especially for deep SNNs that can achieve better accuracy, we accelerate SyncNN on Xilinx ARM-FPGA System-on-Chips (SoCs). To reduce the computing operations and memory accesses, we first quantize the network weights to 16 bits, 8 bits and 4 bits using SNN friendly quantization that puts more priority in weights with higher magnitude. Second, we design configurable and scalable neuron encoding and spike aggregation engines, which addresses the challenge of dynamic and irregular access patterns due to the event-driven nature of SNNs and explores different combinations of pipeline and parallelization techniques, as well as memory access optimizations. Finally, to support different network and layer sizes on different FPGA devices, we use a hierarchical on-chip buffering strategy.

Unlike prior FPGA studies [14]–[18] that only evaluated small networks such as MLP and LeNet, we have evaluated SyncNN for various CNN-based networks including LeNet, Network in Network (NiN) and VGG, for multiple datasets including MNIST, SVHN and CIFAR-10, on multiple ARM-FPGA SoCs, including Xilinx ZedBoard, ZCU104 and ZCU102 boards. Compared to state-of-the-art SNN acceleration work on FPGAs [14], for the same experimental setup—LeNet for MNIST dataset, on Xilinx ZCU102 board—SyncNN achieves 13,086 frames per second, which is 6.16x faster than [14], and 99.3% accuracy, which is higher than 99.2% in [14].

In summary, this paper makes the following contributions:

- A novel synchronous event-driven SNN with quantitative comparison to CNN and asynchronous SNN approaches.
- The first configurable and scalable FPGA engine of SNN that supports deep networks on multiple FPGA devices. The SyncNN framework is also open sourced at https://github.com/SFU-HiAccel/SyncNN.
- The first 4-bit SNN on FPGAs (for LeNet) that achieves a very high accuracy of 99.6% for the MNIST dataset.
- State-of-the-art SNN performance of 13,086 frames per second (FPS) for the MNIST dataset (using LeNet).

II. SNN Background

A. SNNs and Their Training

Figure 1 illustrates how SNNs work [1], [2]. When an input *spike* comes into a neuron, the *membrane potential* of the neuron is either increased or decreased based on the nature of the spike (excitatory or inhibitory). Once the membrane potential crosses the threshold value, the neuron generates spikes to the connected neurons in the next layer, and its membrane potential is reset. The neurons in SNNs are activated in a asynchronous fashion. At each timestep, a set of neurons are activated among different layers. Overall, at the end of the encoding window, there is a large fraction of neurons that are not activated and computed.

However, temporal training of SNNs is often more computationally intensive to achieve high accuracy and has been studied only on shallow networks [4]–[8]. As described in Section I, in this paper, we focus on rate encoding based SNNs that allow the direct conversion of trained ANNs to SNNs to explore deeper CNN networks on FPGAs. In the conversion, the weights obtained from the trained CNNs are mapped to a network of spiking units with the same network topology. And the activation of neurons in CNNs is proportional to the firing rate of SNNs [10]–[13]. Recently, the conversion based SNN models has been studied on complex VGG and residual networks [9] with high accuracy. In this paper, we have applied the ReLu activation, weight normalization, zero bias and other optimizations in [9]–[12] during the conversion.

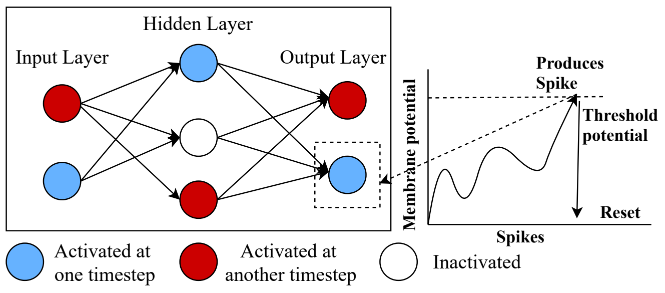

Fig. 1. Overview of the SNN working mechanism

B. Hardware Acceleration for SNNs

It is often inefficient to execute the event-driven SNNs on CPUs, especially embedded CPUs on edge devices. Therefore, there are many studies that try to accelerate SNNs using specialized hardware. First, several studies have explored GPU implementations for SNNs and showed significant speedup over CPUs [13], [19], [20]. Second, lots of efforts have been made to develop neuromorphic hardware for SNNs where the communication of spikes is event-driven [9], [21]–[26]. Third, FPGAs are also a very attractive alternative, especially for the SNN inference on edge devices, as they are commercially-available hardware that can be customized for the SNN computation. Several studies have explored the FPGA acceleration for SNNs [14]–[18], [27]–[31].

The goal of this paper is to accelerate rate encoding SNNs on FPGAs to achieve real-time inference on edge devices for deep networks with high accuracy. We will present the quantitative comparison of our work and prior studies using GPUs, FPGAs, and neuromorphic hardware in Section V-E.

III. SyncNN: Synchronous SNN Approach

A. Rate Encoding based SNN Algorithm

Algorithm 1 presents an overview of the conventional rate encoding SNN algorithm based on the integrate and fire (IF) model and Poisson spike generation [10]. For each input image, it iteratively calls the `SPIKING_NET` function (lines 1-4) for a particular encoding window. The number of simulation steps (*Sims*), i.e., timesteps, is based on the network model and the input. As the encoding window increases, more number of spikes are transmitted in the network. This `SPIKING_NET` function calls the following three major functions (lines 5-12):

1. *Function `POISSON_ENCODING` (lines 13-18).* The input nodes in the network are called *Poissons* as they use Poisson random variables to convert the input amplitudes to a random spike train. It takes the image (img*) as input and generates Poisson spikes (pSp*). These are activated based on the pixel intensity of the image: a Poisson random variable is compared with the pixel intensity, and based on the comparison, the Poisson unit is activated and it spikes. Let the time taken to complete this function be *PE*.
2. *Function `SPIKE_AGGREGATE` (lines 19-25).* The actual computation takes place here. Depending on the layer of

Algorithm 1 Pseudo code for conventional rate-based SNN

```
function MAIN
    for each image do
        for each simulation step do
            SPIKING_NET(img*)
function SPIKING_NET(img*)
    #Encodes input Poisson spikes: pSp*
    POISSON_ENCODING(img*,pSp*)
    for each layer in network do
        #Update membrane potential: Vm*
        SPIKE_AGGREGATE(weights*,pSp*,nSp*,Vm*)
        #Encodes neuron spikes: nSp*
        NEURON_ENCODING(Vm*,nSp*)
function POISSON_ENCODING(img*,pSp*)
    pSpiked = 0 #Reset counter for Poisson spikes
    for each image pixel do
        if pixelValue > PoissonThreshold then
            pSp[pSpiked] = pixelIndex
            pSpiked++
function SPIKE_AGGREGATE(weights*,pSp*,nSp*,Vm*)
    #Convolutional, Pooling or Dense
    for each spiked inputs in psp*/nsp* do
        for each weight w in the kernel do
            for each number of feature maps o do
                #Performs the aggregation operation
                Vm[o] += weights[w]
function NEURON_ENCODING(Vm*,nSp*)
    nSpiked = 0 #Reset counter for neuron spikes
    for each neuron n do
        if Vm[n] > VmThreshold then
            nSp[nSpiked] = n
            Vm[n] = Vreset
            nSpiked++
```

the network (convolutional, pooling or dense), we perform the aggregation operation of weights to the membrane potential (Vm*) of the neuron. While ANNs/CNNs perform operation for all the inputs, SNNs compute only for the encoded spiked inputs (Poisson Spikes - pSp*, Neuron Spikes - nSp*) in that simulation step, which is the key advantage of SNNs. Let the time taken to complete this function for the l_{th} layer be *SA[l]*.

3. *Function* `NEURON_ENCODING` *(lines 26-32).* It takes the aggregated membrane potential as inputs (Vm*), and generates neuron spikes (nSp*). As shown in Figure 1, the membrane potential (Vm*) of the neuron is updated based on the spikes it receives, and once it reaches the threshold value, the neuron is activated with a spike and the membrane potential is set back to Vreset. Let the time taken to complete this function for the l_{th} layer be *NE[l]*.

The `SPIKE_AGGREGATE` function takes the encoded spiked inputs generated in the `NEURON_ENCODING` function as input and calculates the membrane potentials of neurons for the `NEURON_ENCODING` function in the next layer. This process is repeated for all the layers (*NL*) in the network. When this algorithm is implemented in a naive synchronous fashion, the time it takes to classify one image is:

$$T_{naive\_sync} = Sims * (PE + \sum_{l=1}^{NL}(NE[l] + SA[l])) \quad (1)$$

B. Asynchronous Approach of SNNs

One of the attractive features for SNNs is their asynchronous execution flow. If there is enough hardware resource, all the layers in the network can be computed concurrently in a pipeline fashion. The pipeline throughput is determined by the layer that takes the most of the time at that simulation step. Within the three major functions of SNNs, the `SPIKE_AGGREGATE` function at any simulation step is the most time consuming one. Therefore, the time it takes to classify one image is:

$$T_{async} = \sum_{s=1}^{Sims} \max(SA_s[1], SA_s[2], ..., SA_s[NL]) \quad (2)$$

However, the asynchronous approach faces several challenges when accelerated on edge devices.

1. **Network size limitation.** To run all the layers in parallel, the network parameters (inputs, weights, and outputs) have to be on-chip independently for every layer. Also, computing resources have to be allocated for all the layers. For edge devices, which has limited resources, it is difficult to implement larger networks in the asynchronous fashion.
2. **Resource underutilization.** The `SPIKE_AGGREGATE` function has the highest workload compared to the other functions. In the asynchronous approach, the overall processing time in a simulation step highly depends on the computation unit of the layer that has the longest latency. Therefore, all the resources for other functions have to remain idle until the slowest function finishes. Moreover, since all the layers are implemented on the hardware, the optimizations within each layer is also restricted due to limited resource.
3. **Impact of encoding window.** For the asynchronous approach, all the layers in the network still run for multiple simulation steps. For small networks like LeNet and MLP, the encoding window is very small to achieve a good accuracy. However, for larger networks like NiN and VGG, the encoding window is very large to achieve a good accuracy as shown in Section V-B. Based on Equation 2, the large encoding window ($Sims$) limits the performance.

C. SyncNN: Synchronous Approach of SNN Acceleration

To address the above challenges, we propose a novel synchronous approach, called SyncNN, to accelerate rate encoding SNNs on hardware. Unlike previous approaches, we do not run the entire network for multiple simulation steps. Only the input layer, i.e., the `POISSON_ENCODING` function, is run for multiple simulation steps. For the remaining layers, the `SPIKE_AGGREGATE` and `NEURON_ENCODING` functions are run only once for each layer in the network and all spikes are aggregated (increment to the membrane potential) together. For example, if the aggregated membrane potential value of a neuron is twice the value of the threshold, we consider that the neuron needs to spike twice. The key is that instead of spiking a neuron at multiple timesteps (e.g., timestep 1 and 4), SyncNN only spikes a neuron at the

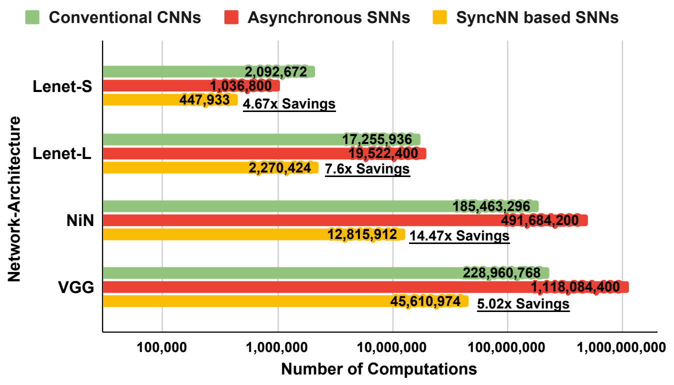

Fig. 2. Comparison of computation operations required in conventional CNNs, asynchronous SNNs, and our SyncNN based SNNs (savings is over CNNs)

final timestep (and only one timestep) and spikes it with the equivalent number of times. Due to the synchronous layer-by-layer execution, this is equivalent to the original SNN with multiple timesteps. SyncNN still preserves the event-driven feature of SNN as only those neurons whose membrane potential exceeds the threshold will be activated and computed. The time it takes to classify one image is:

$$T_{SyncNN} = (Sims * PE) + \sum_{l=1}^{NL} (NE[l] + SA[l])) \tag{3}$$

When to use SyncNN. From the algorithmic perspective, based on Equation 2 and 3, if we only consider the most time consuming term $SA[l]$, SyncNN is faster than the asynchronous approach for any network whose encoding window ($Sims$) is greater than the number of layers (NL). From the hardware perspective, since a network is executed layer by layer in SyncNN, the computing and memory resources can be reused between layers through time multiplexing. Hence, any deep networks can be implemented and the slowest function will not cause resource underutilization for other functions. Also, the encoding window only affects the performance of the input layer, i.e., the `POISSON_ENCODING` function, where parallelism can be explored among multiple timesteps.

D. *Computational Comparison for SyncNN*

Finally, we compare the number of computing operations required in the conventional CNNs, asynchronous SNNs, and our SyncNNs in Figure 2. The detailed experimental setup will be explained in Section V-A. Note that for asynchronous SNNs, we assumed an ideal case where all layers can execute in parallel and we only counted the computing operations in the largest layer. For CNNs and SyncNN based SNNs, we counted all computing operations in all layers since they execute layer by layer. Our SyncNN approach has a significant advantage over asynchronous SNNs, especially for deep networks (NiN and VGG) that require a large encoding window, since SyncNN requires only one timestep for all layers except the input layer. Compared to conventional CNNs, our SyncNN based event-driven SNNs can reduce the number of computing operations by 4.67x to 14.47x.

IV. Hardware Design of SyncNN Framework

The computation and memory access pattern of SNNs is irregular, because of their event-driven nature. In this section, we discuss optimization techniques implemented in SyncNN to address this challenge.

A. *Quantization*

To improve the computing and memory access efficiency of SyncNN, we first apply SNN friendly quantization to represent the weights in low precision fixed-point values. In SNNs, the weights with higher magnitude have more impact on altering the membrane potential of the layer. Hence, priority is given to represent those high magnitude weights for the number of bits chosen. After analyzing the distribution of the weights in a layer, we find that very few weights have large magnitudes and the majority of the weights are small. Therefore, based on the weight distribution of the layer, we choose the X-th percentile (e.g., 99-th percentile) of the weight and represent that weight with the maximum possible fixed point representation for the number of bits chosen (*max_fixed_point*). Now, we can decide the scaling factor (*scale*) based on the following equation:

$$\text{Xth\_percentile\_weight} * \text{scale} = \text{max\_fixed\_point} \tag{4}$$

We then multiply all the weights with the obtained scaling factor (*scale*) and round it to the nearest fixed point value. For the values whose magnitude is greater than the *max_fixed_point*, we clip it to the *max_fixed_point*.

B. *Computation Optimization*

In SyncNN, we design configurable and scalable compute engines for all major functions with pipeline and parallelization techniques, as shown in Figure 3, which can be deployed on different Xilinx FPGA boards based on the available memory and computing resources.

SPIKE_AGGREGATE. The topology of the function depends on the layer of the network. In SyncNN, we implement the widely used CNN computational units such as Convolutional, Pooling, Dense and Global Average Pooling. The major difference between the conventional CNNs and SyncNN based SNNs is the inputs presented to every layer. In SyncNNs, only the spiked neurons (random and event-driven) in the previous layer are presented as inputs for the current layer. That is, the output of this function is now dependent on the randomly spiked inputs, which makes parallelization challenging. We explain only the convolutional unit implementation and omit other types of units due to space limitation.

We first design a basic convolutional unit called *ConvPE* to process each input feature map. Inside the *ConvPE*, since the output feature map loop (*OMaps*) has no dependency, we swap it in as the innermost loop and parallelize it by partially unrolling it with a factor of *UOMaps*. Then we pipeline the rest of the loop nests (kernel loops and the event-driven encoded spikes loop) with II=1. Between the input maps (*IMaps*), there is a data dependency on the output membrane potential array (*Vm*), since multiple neurons from the previous layer can generate input spikes to the same neuron in the current

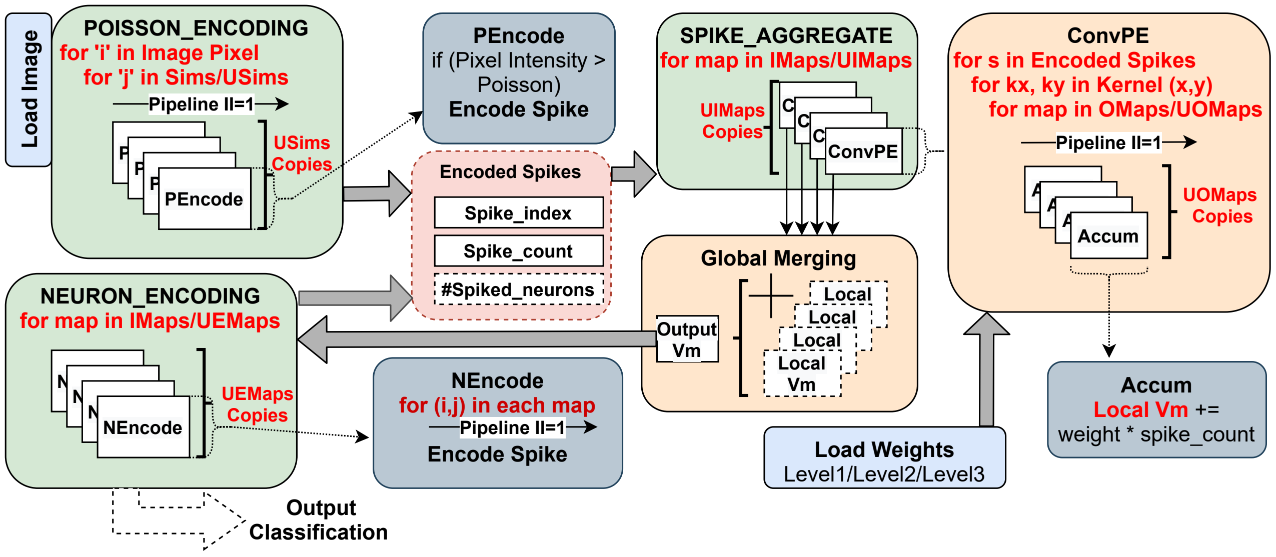

Fig. 3. Overview of the hardware architecture for SyncNN

layer. In SyncNN, we allocate duplicate copies (*UIMaps*) of output *Vm* array (*local_Vm*) for each group of input maps to enable parallel processing. Finally, we add a global merging module to aggregate all the *local_Vm* copies to produce the final updated output membrane potential array *Vm*.

POISSON_ENCODING. In SyncNN, all the input nodes (i.e., image pixels) in the `POISSON_ENCODING` function is run for multiple simulation steps *Sims*. We partially unroll (parallelize) the simulation step loop with a factor of *USims*, and pipeline the outer loop with pipeline initiation interval as 1 (II=1). The output of this function is the encoded spikes only for the activated neurons in the input layer.

NEURON_ENCODING. First, we parallelize the neuron encoding for each input feature map of a layer by partially unrolling the input feature map loop with a factor of *EMaps*. Inside each feature map, we pipeline the neural encoding of each neuron with II=1. The output of this function is the encoded spikes only for the activated neurons in that layer. If it is the final layer, it gives the classification output.

For those activated neurons, we encode their original neuron index and aggregated number of spikes in a pair of arrays ($Spike\_index$ and $Spike\_count$); we also record the total number of spiked neurons ($\#Spike\_neurons$). While such encoding reduces the total number of computations as only activated neurons are encoded and computed, it also creates the dynamic and irregular access pattern challenge.

C. Memory Access Optimization

The aforementioned event-driven nature of SNNs makes the weight access irregular and it is critical to buffer the weights on chip. In SyncNN, we use a hierarchical on-chip buffering technique to buffer as many weights as possible, depending on the network size and the on-chip memory size available on the FPGA board. We load the weights in a coalesced (widened bus) and burst fashion [32], [33] from the off-chip memory to on-chip memory at different granularity.

1. **Level 1:** If the FPGA board can load the weights corresponding to all the layers in the neural network, the weights are prefetched at network level and therefore the entire off-chip access for the network is done only once. This is the best case scenario for loading the network weights and is often possible for smaller networks.
2. **Level 2:** For bigger networks or smaller FPGA boards, where we cannot load all the weights on-chip, we load them layer-wise so that the weights buffer can be reused for every layer in network.
3. **Level 3:** For very big networks (like NiN, VGG) or very small FPGA boards, it is not possible to load even one layer's weight on-chip. In this case, we go one step further in granularity and load the weights for every map of the layer and perform the computation for that map. For the next map, we again load the weights from off-chip.

V. Experimental Results

A. Experimental Setup

SyncNN framework implements a wide range of image classification network architectures (CNN models) including LeNet, Network in Network (NiN) and VGG-13. The network configurations for each network is summarized in Table I. n*C*s denotes the convolutional layer with kernel size s*s and n is the number of kernels, *P*s denotes the pooling layer with size s*s and GAP denotes Global Average Pooling Layer, and a pure number n denotes the dense layer with n neurons. The first LeNet network *Lenet-S* is the same network configuration used in [14]. The second LeNet network *Lenet-L* is comparatively larger than *Lenet-S*, where it is difficult to hold all the network parameters on-chip, but has better accuracy. Three widely used image classification datasets are evaluated in SyncNN: MNIST, SVHN and CIFAR-10.

Our SyncNN framework is built using Xilinx SDSoC 2019.1 that integrates Vivado HLS C++. Three different Xilinx SoC boards are used to test the SyncNN framework: Xilinx Zynq

Table I. Neural network configurations

Network	Configuration
Lenet-S	Input-32C3-P2-32C3-P2-256-Output
Lenet-L	Input-32C5-P2-64C5-P2-2048-Output
NiN	Input-(192C5-192C1-192C1-P3)*2 -192C5-192C1-10C1-GAP-Output
VGG	Input-(64C3-64C3-P2)-(128C3-128C3-P2) -(256C3-256C3-P2)-(512C3-512C3-P2)*2 -256-256-Output

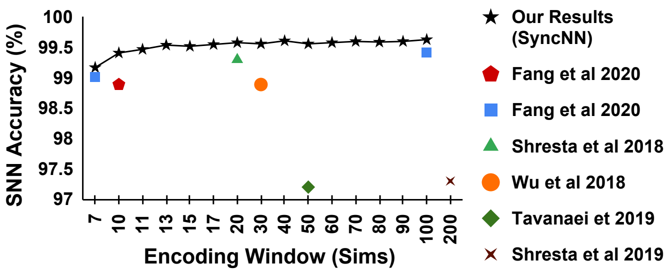

Fig. 4. Impact of encoding window on the accuracy for MNIST dataset

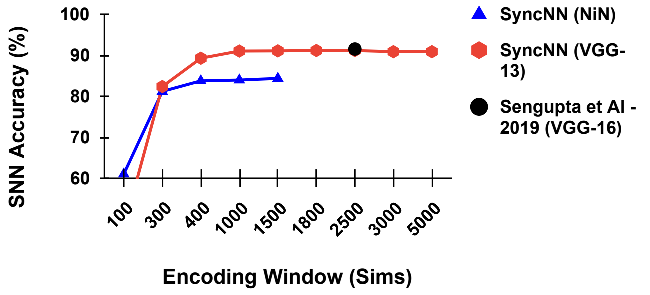

Fig. 5. Impact of encoding window on the accuracy for CIFAR-10 dataset

ZedBoard, Zynq UltraScale+ ZCU104 and ZCU102 boards. The main difference between the boards is the available resources: ZCU102 > ZCU104 > ZedBoard. Our designs run at 100MHz on ZedBoard, 150MHz on ZCU104, and 200MHz on ZCU102.

B. Impact of Encoding Window

As the encoding window increases, the network transmits more spikes to predict the output correctly. For MNIST dataset with smaller networks, as shown in Figure 4, the encoding window required to achieve a good accuracy is relatively small. Shrestha et al. [35], Tavanaei et al. [36], Wu et al. [37], Shrestha et al. [38] achieves 97.3%, 97.2%, 98.89%, 99.3% accuracy for 200, 50, 30, 20 Sims respectively. More recently, Fang et al. [14] achieves 99.01% at 7 Sims with a peak accuracy of 99.43%. Our work, SyncNN achieves better accuracy of 99.17% for the same 7 Sims and achieves a peak accuracy of 99.63% with 100 Sims, which is more accurate than all prior FPGA implementations.

For CIFAR-10 dataset, with larger networks like NiN and VGG, as shown in Figure 5, the required encoding window is very large to transmit the needed spikes in the network. In our SyncNN approach, the NiN achieves 88.19% accuracy at 400 Sims. And the VGG-13 network achieves 90.79% accuracy at 1,800 Sims. The recent study by Sengupta et al. [9] also confirms the need for a larger encoding window as they achieve the peak accuracy at 2,500 Sims with the VGG-16 network.

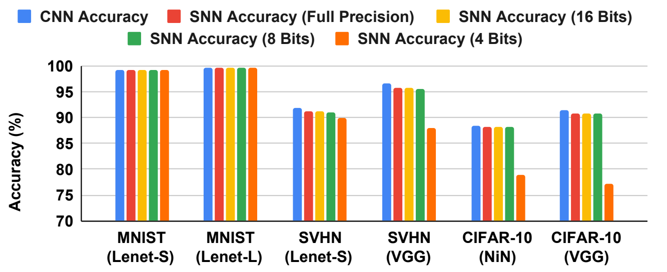

Fig. 6. Inference accuracy comparison of CNN and SyncNN based SNNs with different data precision

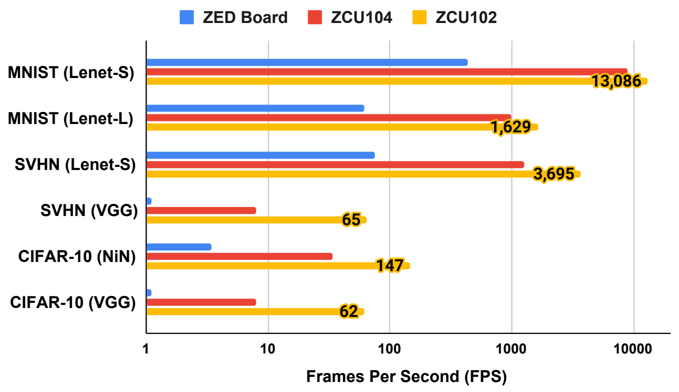

Fig. 7. Frames per second (FPS) for different networks and datasets running on different FPGA boards

C. Accuracy of CNN-SNN Conversion and Quantization

In conversion based SNNs, the accuracy of the SNN network is dependent on the underlying CNN accuracy. The better we train the CNN model, the higher SNN accuracy is achieved. During the CNN training phase, we use data normalization, augmentation, regularization and dropout techniques to improve the accuracy of the network. We do not use batch normalization during training, which is an important CNN optimization, because it affects the SNN accuracy after conversion. As summarized in Figure 6, the accuracy loss from converting CNN to SNN is negligible. When the compute units are offload to hardware, there is no drop in accuracy with 32 and 16 bits representation. For the LeNet networks there is negligible drop in accuracy for 8 and 4 bits representations. For NiN and VGG networks, there is negligible drop for 8 bits, but big drop for 4 bits. Therefore, we use 4 bits for LeNet networks and 8 bits for NiN and VGG networks in SyncNN hardware evaluation. We plan to further improve the 4-bit accuracy for deeper networks in future work.

D. Overall Performance

Figure 7 shows the overall performance of each network with the best configuration on each FPGA board (configuration parameters are presented in Section IV-B). The ZCU102 board is about 1.46x to 8.4x faster than the ZCU104 board and 26.7x to 60.9x faster than the ZED board for various networks.

Table II. Comparison with related work for image classification using SNNs for the MNIST dataset

Work	**Platform**		**Frequency**	**Training Method**	**Precision (bits)**	**Network**	**Accuracy (%)**	**FPS**	**FPS/ DSP**	**FPS per 1k LUTs**
Stromatias et al. (2015) [22]	ASIC	SpiNNaker	150MHz	Spike Based	16	MLP	95	50	-	-
Esser et al. (2015) [34]	ASIC	True North	-	Offline Training	Binary	-	95	1,000	-	-
Darwin (2017) [25]	ASIC	-	25MHz	-	16	MLP	93.8	6.25	-	-
Minitaur (2014) [17]	FPGA	Spartan-6 LX150	75MHz	Spike Based	16	MLP	92	108	-	1.22
Han et al. (2020) [15]	FPGA	Xilinx ZC706	200MHz	Converted SNNs	16	MLP	97.06	161	-	29.92
Ju et al. (2020) [16]	FPGA	Xilinx ZCU102	150MHz	Converted SNNs	8	CNN (LeNet)	98.94	164	-	1.52
Fang et al. (2020) [14]	ASIC	Intel Loihi	-	Spike Based	16	CNN (Lenet-S)	99.2	671	-	-
	CPU	Intel i9-9900K	3.7GHz					100	-	-
	GPU	Nvidia RTX 5000	1.62GHz					864	-	-
	Edge GPU	Nvidia AGX Xavier	1.37GHz					211	-	-
	FPGA	Xilinx ZCU102 (Simulation)	125MHz					2,124	1.18	13.62
SyncNN (Our Work)	**FPGA**	**Xilinx ZCU102**	**200MHz**	**Converted SNNs**	**4**	**CNN (Lenet-S)**	**99.3**	**13,086**	**19.23**	**61.2**
						CNN (Lenet-L)	**99.6**	**1,629**	**2.94**	**7.25**

On the ZCU102 board, the MNIST dataset for the Lenet-S network achieves a maximum of 13,086 FPS (frames per second) whereas it can achieve 1,629 FPS for the Lenet-L network. The SVHN dataset achieves a maximum of 3,695 FPS for the LeNet-S network and 65 FPS for the VGG network. The CIFAR-10 dataset achieves a maximum of 147 FPS for NiN network and 62 FPS for the VGG network.

E. Comparison with Related Work

Most of the related studies are evaluated on the MNIST dataset with MLP or smaller CNN networks like LeNet. Table II summarizes all the recent related work of SNNs on FPGAs and neuromorphic hardware for the MNIST dataset. The FPGA implementations, Minitaur [17], Han et al. [15], and Ju et al. [16], achieve 92%, 97.06%, and 98.94% accuracy with 108, 161 and 164 FPS, respectively. The prior neuromorphic hardware studies, Darwin [25], Stromatias et al. [22], and Esser et al. [34], achieve 93.8%, 95%, 95% accuracy with 6.25, 50, and 1,000 FPS respectively. More recently, a temporal encoding based SNN inference on Xilinx ZCU102 SoC board for MNIST image clasification achieves an accuracy of 99.2% and a simulation performance of 2,124 FPS [14] at 125MHz. Shown in Table II, the same work also demonstrates superior performance of FPGA-based SNNs over that on CPU, GPU, Intel Loihi neuromorphic chip [14].

Our SyncNN framework stands out in terms of all four metrics—accuracy, FPS, FPS per DSP, and FPS per 1K LUTs usage—for the MNIST dataset, compared to the prior neuromorphic hardware implementations, FPGA and GPU implementations. SyncNN achieves a maximum of 13,086 FPS at 99.3% accuracy for the same *Lenet-S* network and the same FPGA board used in [14]. Also, SyncNN achieves 99.6% accuracy with 1,629 FPS on a bigger network *Lenet-L*. For both networks, the hardware runs at 200MHz. SyncNN is the first work to explore 4 bits weights precision for SNNs on FPGAs. It is also the first SNN framework on FPGAs that supports deep CNN models like VGG and NiN.

VI. Conclusion and Future Work

In this paper, we have proposed a novel synchronous approach called SyncNN, to accelerate event-driven rate encoding SNNs on FPGAs. First, we quantitatively compared the CNNs, asynchronous SNNs, and SyncNNs, to demonstrate the advantage of SyncNNs. Second, we applied SNN-friendly quantization, to reduce the computing operations and memory accesses. Moreover, we developed configurable and scalable computing engines on FPGAs to accelerate different network models of SNNs across various Xilinx ARM-FPGA SoCs. SyncNN is capable to run any deep networks on a given Xilinx ARM-FPGA SoC and it is the first work to explore NiN and VGG networks for SNNs on FPGAs. SyncNN achieves the state-of-the-art performance for MNIST dataset with 13,086 FPS, which is 6.16x faster than the previous state-of-the-art implementation for the same network configuration and same FPGA board. SyncNN reports the best accuracy of 99.6% for MNIST, which is so far the highest accuracy recorded for SNNs on any hardware implementation, to the best of our knowledge. We also open source SyncNN (https://github.com/SFU-HiAccel/SyncNN) to the community to stimulate more researches in the area of FPGA-based SNNs that has a high potential in future deep learning systems. In future work, we plan to further improve the accuracy and performance of deeper SNN networks at lower data precision.

Acknowledgments

We acknowledge the support from Natural Sciences and Engineering Research Council of Canada (NSERC Discovery Grant RGPIN-2019-04613 and DGECR-2019-00120, Alliance Grant ALLRP-552042-2020); Canada Foundation for Innovation John R. Evans Leaders Fund and British Columbia Knowledge Dev. Fund; Simon Fraser University New Faculty Start-up Grant; Huawei, Xilinx, and Nvidia. We also thank Dr. Abhronil Sengupta and his group from Penn State University for helping us better understand training techniques for SNNs, and Kunpeng Xie from Nankai University for helping us collect data on Xilinx ZCU102 FPGA board.

References

[1] M. Pfeiffer and T. Pfeil, "Deep learning with spiking neurons: Opportunities and challenges," *Frontiers in Neuroscience*, vol. 12, p. 774, 2018.

[2] W. Maass, "Networks of spiking neurons: The third generation of neural network models," *Neural Networks*, vol. 10, no. 9, pp. 1659 – 1671, 1997.

[3] H. Fang, A. Shrestha, Z. Zhao, and Q. Qiu, "Exploiting neuron and synapse filter dynamics in spatial temporal learning of deep spiking neural network," in *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence (IJCAI-20)*, 07 2020, pp. 2771–2778.

[4] S. M. Bohte, J. N. Kok, and H. L. Poutré], "Error-backpropagation in temporally encoded networks of spiking neurons," *Neurocomputing*, vol. 48, no. 1, pp. 17 – 37, 2002.

[5] S. McKennoch, Dingding Liu, and L. G. Bushnell, "Fast modifications of the spikeprop algorithm," in *The 2006 IEEE International Joint Conference on Neural Network Proceedings*, 2006, pp. 3970–3977.

[6] F. Ponulak and A. Kasiundefinedski, "Supervised learning in spiking neural networks with resume: Sequence learning, classification, and spike shifting," *Neural Comput.*, vol. 22, no. 2, p. 467–510, Feb. 2010.

[7] A. Taherkhani, A. Belatreche, Y. Li, and L. P. Maguire, "Dl-resume: A delay learning-based remote supervised method for spiking neurons," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 26, no. 12, pp. 3137–3149, 2015.

[8] G.-q. Bi and M.-m. Poo, "Synaptic modifications in cultured hippocampal neurons: Dependence on spike timing, synaptic strength, and postsynaptic cell type," *Journal of Neuroscience*, vol. 18, no. 24, pp. 10 464–10 472, 1998.

[9] A. Sengupta, Y. Ye, R. Wang, C. Liu, and K. Roy, "Going deeper in spiking neural networks: Vgg and residual architectures," *Frontiers in Neuroscience*, vol. 13, p. 95, 2019.

[10] P. U. Diehl, D. Neil, J. Binas, M. Cook, S. 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)*, 2015, pp. 1–8.

[11] E. Hunsberger and C. Eliasmith, "Training spiking deep networks for neuromorphic hardware," *arXiv:1611.05141*, 2016.

[12] B. Rueckauer, I.-A. Lungu, Y. Hu, and M. Pfeiffer, "Theory and tools for the conversion of analog to spiking convolutional neural networks," *arXiv:1612.04052*, 2016.

[13] R. Brette and D. F. Goodman, "Simulating spiking neural networks on gpu," *Network: Computation in Neural Systems*, vol. 23, no. 4, pp. 167–182, 2012.

[14] H. Fang, Z. Mei, A. Shrestha, Z. Zhao, Y. Li, and Q. Qiu, "Encoding, model, and architecture: Systematic optimization for spiking neural network in fpgas," in *2020 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, 2020, pp. 1–9.

[15] J. Han, Z. Li, W. Zheng, and Y. Zhang, "Hardware implementation of spiking neural networks on fpga," *Tsinghua Science and Technology*, vol. 25, no. 4, pp. 479–486, 2020.

[16] X. Ju, B. Fang, R. Yan, X. Xu, and H. Tang, "An fpga implementation of deep spiking neural networks for low-power and fast classification," *Neural Computation*, vol. 32, no. 1, pp. 182–204, 2020.

[17] D. Neil and S. Liu, "Minitaur, an event-driven fpga-based spiking network accelerator," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 22, no. 12, pp. 2621–2628, 2014.

[18] Morcos, Benjamin, "Nengofpga: an fpga backend for the nengo neural simulator," 2019. [Online]. Available: http://hdl.handle.net/10012/14923

[19] J. Moorkanikara Nageswaran, N. Dutt, J. Krichmar, A. Nicolau, and A. Veidenbaum, "A configurable simulation environment for the efficient simulation of large-scale spiking neural networks on graphics processors," *Neural networks : the official journal of the International Neural Network Society*, vol. 22, pp. 791–800, 08 2009.

[20] A. K. Fidjeland and M. P. Shanahan, "Accelerated simulation of spiking neural networks using gpus," in *The 2010 International Joint Conference on Neural Networks (IJCNN)*, 2010, pp. 1–8.

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

[22] E. Stromatias, D. Neil, F. Galluppi, M. Pfeiffer, S.-C. Liu, and S. Furber, "Scalable energy-efficient, low-latency implementations of trained spiking deep belief networks on spinnaker," in *2015 International Joint Conference on Neural Networks (IJCNN)*, 07 2015.

[23] C. Mayr, S. Höppner, and S. Furber, "Spinnaker 2: A 10 million core processor system for brain simulation and machine learning," *arXiv:1911.02385*, 2019.

[24] 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, B. Brezzo, I. Vo, S. K. Esser, R. Appuswamy, B. Taba, A. Amir, M. D. Flickner, W. P. Risk, R. Manohar, and D. S. Modha, "A million spiking-neuron integrated circuit with a scalable communication network and interface," *Science*, vol. 345, no. 6197, pp. 668–673, 2014.

[25] M. De, S. JunCheng, G. ZongHua, Z. Ming, Z. XiaoLei, X. XiaoQiang, Q. Xu, S. YangJing, and G. Pan, "Darwin: a neuromorphic hardware co-processor based on spiking neural networks," *Journal of Systems Architecture*, vol. 77, 01 2017.

[26] S. K. Esser, P. A. Merolla, J. V. Arthur, A. S. Cassidy, R. Appuswamy, A. Andreopoulos, D. J. Berg, J. L. McKinstry, T. Melano, D. R. Barch, C. di Nolfo, P. Datta, A. Amir, B. Taba, M. D. Flickner, and D. S. Modha, "Convolutional networks for fast, energy-efficient neuromorphic computing," *Proceedings of the National Academy of Sciences*, vol. 113, no. 41, pp. 11 441–11 446, 2016.

[27] A. Rosado-Muñoz, M. Bataller-Mompeán, and J. Guerrero-Martínez, "Fpga implementation of spiking neural networks," *IFAC Proceedings Volumes*, vol. 45, no. 4, pp. 139 – 144, 2012, 1st IFAC Conference on Embedded Systems, Computational Intelligence and Telematics in Control.

[28] K. Cheung, S. R. Schultz, and W. Luk, "Neuroflow: A general purpose spiking neural network simulation platform using customizable processors," *Frontiers in Neuroscience*, vol. 9, p. 516, 2016.

[29] D. Pani, P. Meloni, G. Tuveri, F. Palumbo, P. Massobrio, and L. Raffo, "An fpga platform for real-time simulation of spiking neuronal networks," *Frontiers in Neuroscience*, vol. 11, p. 90, 2017.

[30] R. M. Wang, C. S. Thakur, and A. van Schaik, "An fpga-based massively parallel neuromorphic cortex simulator," *Frontiers in Neuroscience*, vol. 12, p. 213, 2018.

[31] T. Kawao, M. Neishi, T. Okamoto, A. M. Gharehbaghi, T. Kohno, and M. Fujita, "Spiking neural network simulation on fpgas with automatic and intensive pipelining," in *2016 International Symposium on Nonlinear Theory and Its Applications, NOLTA2016*, 11 2016, pp. 202–205.

[32] A. Lu, Z. Fang, W. Liu, and L. Shannon, "Demystifying the memory system of modern datacenter fpgas for software programmers through microbenchmarking," in *The 2021 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, ser. FPGA '21. Association for Computing Machinery, 2021, p. 105–115.

[33] C. Zhang, G. Sun, Z. Fang, P. Zhou, P. Pan, and J. Cong, "Caffeine: Toward uniformed representation and acceleration for deep convolutional neural networks," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 38, no. 11, pp. 2072–2085, 2019.

[34] S. K. Esser, R. Appuswamy, P. Merolla, J. V. Arthur, and D. S. Modha, "Backpropagation for energy-efficient neuromorphic computing," in *Advances in Neural Information Processing Systems*, C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett, Eds., vol. 28. Curran Associates, Inc., 2015, pp. 1117–1125.

[35] S. B. Shrestha and G. Orchard, "Slayer: Spike layer error reassignment in time," in *Advances in Neural Information Processing Systems*, S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, Eds., vol. 31. Curran Associates, Inc., 2018, pp. 1412–1421.

[36] A. Tavanaei and A. Maida, "Bp-stdp: Approximating backpropagation using spike timing dependent plasticity," *Neurocomputing*, vol. 330, pp. 39 – 47, 2019.

[37] Y. Wu, L. Deng, G. Li, J. Zhu, and L. Shi, "Spatio-temporal backpropagation for training high-performance spiking neural networks," *Frontiers in Neuroscience*, vol. 12, p. 331, 2018.

[38] A. Shrestha, H. Fang, Q. Wu, and Q. Qiu, "Approximating backpropagation for a biologically plausible local learning rule in spiking neural networks," in *Proceedings of the International Conference on Neuromorphic Systems*, ser. ICONS '19. New York, NY, USA: Association for Computing Machinery, 2019.