






# FireFly v2: Advancing Hardware Support for High-Performance Spiking Neural Network with a Spatiotemporal FPGA Accelerator

Jindong Li , Guobin Shen , Dongcheng Zhao , Qian Zhang , Yi Zeng 

**Abstract**—Spiking Neural Networks (SNNs) are expected to be a promising alternative to Artificial Neural Networks (ANNs) due to their strong biological interpretability and high energy efficiency. Specialized SNN hardware offers clear advantages over general-purpose devices in terms of power and performance. However, there’s still room to advance hardware support for state-of-the-art (SOTA) SNN algorithms and improve computation and memory efficiency. As a further step in supporting high-performance SNNs on specialized hardware, we introduce FireFly v2, an FPGA SNN accelerator that can address the issue of non-spike operation in current SOTA SNN algorithms, which presents an obstacle in the end-to-end deployment onto existing SNN hardware. To more effectively align with the SNN characteristics, we design a spatiotemporal dataflow that allows four dimensions of parallelism and eliminates the need for membrane potential storage, enabling on-the-fly spike processing and spike generation. To further improve hardware acceleration performance, we develop a high-performance spike computing engine as a backend based on a systolic array operating at 500-600MHz. To the best of our knowledge, FireFly v2 achieves the highest clock frequency among all FPGA-based implementations. Furthermore, it stands as the first SNN accelerator capable of supporting non-spike operations, which are commonly used in advanced SNN algorithms. FireFly v2 has doubled the throughput and DSP efficiency when compared to our previous version of FireFly and it exhibits  $\times 1.33$  the DSP efficiency and  $\times 1.42$  the power efficiency compared to the current most advanced FPGA accelerators.

**Index Terms**—Spiking Neural Networks, Field-programmable gate array, Hardware Accelerator, Non-Spike Operation, Spatiotemporal Dataflow

## I. INTRODUCTION

Manuscript created 28 September 2023; revised 5 January 2024 and 8 March 2024; accepted 16 March 2024. This work was supported by the Chinese Academy of Sciences Foundation Frontier Scientific Research Program (ZDBS-LY- JSC013). (Corresponding authors: Qian Zhang; Yi Zeng.)

Jindong Li and Qian Zhang are with the School of Artificial Intelligence, University of Chinese Academy of Sciences, Beijing 100049, China, and also with the Brain-inspired Cognitive Intelligence Lab, Institute of Automation, Chinese Academy of Sciences, Beijing 100190, China (e-mail: lijindong2022@ia.ac.cn, q.zhang@ia.ac.cn).

Guobin Shen is with the School of Future Technology, University of Chinese Academy of Sciences, Beijing 100049, China, and also with the Brain-inspired Cognitive Intelligence Lab, Institute of Automation, Chinese Academy of Sciences, Beijing 100190, China (e-mail: shenguobin2021@ia.ac.cn).

Dongcheng Zhao is with the Brain-inspired Cognitive Intelligence Lab, Institute of Automation, Chinese Academy of Sciences, Beijing 100190, China (e-mail: zhaodongcheng2016@ia.ac.cn).

Yi Zeng is with the Brain-inspired Cognitive Intelligence Lab, Institute of Automation, Chinese Academy of Sciences, Beijing 100190, China, and University of Chinese Academy of Sciences, Beijing 100049, China, and Center for Excellence in Brain Science and Intelligence Technology, Chinese Academy of Sciences, Shanghai 200031, China (e-mail: yi.zeng@ia.ac.cn).

Spiking neural networks (SNNs) are considered a promising alternative to artificial neural networks (ANNs) due to their high biological plausibility, event-driven nature, and low power consumption [1]. Recent advancements in SNN algorithms have drawn inspiration from both biological evidence and deep learning insights, narrowing the performance gap with ANNs [2]. However, current neuromorphic hardware [3] [4] cannot match the performance of ANN accelerator counterparts and, worse still, cannot support state-of-the-art SNN algorithms.

Ongoing research aimed at developing high-performance SNN algorithms has made significant strides in narrowing the benchmark accuracy gap between ANNs. However, it also presents challenges for specialized SNN hardware due to the introduction of hardware-unfriendly computation. Fig.1 shows four typical backbone structure of current SNN algorithms, Conv represents the convolution layer, BN represents the batchnormalization layer which can be fused into the convolution layer after training, LIF represents the leaky-and-fire neuron node. The blue block indicates the hardware-friendly spike convolution and the red block indicates the non-spike convolution not supported by existing specialized SNN hardware. As shown in Fig.1A, current SNN algorithms typically use direct input encoding [5] with analog pixel values applied to the initial convolutional layer, followed by spiking neurons for end-to-end backpropagation, improved benchmark accuracy and reduced time steps. However, the direct encoding convolutional layer poses compatibility challenges with existing specialized SNN hardware designed for spike-based computation. As shown in Fig.1B, current deep SNN models incorporate a residual connection by applying spike-element-wise summation between the residual path and the shortcut path [6]. However, the sum of spikes operations introduce non-spike operation in the next convolutional layer. Furthermore, the commonly employed Average Pooling function in SNN models introduces fractional-spike convolution as shown in Fig.1C, which is not supported by current SNN hardware equipped with spike-based computing engines. Existing specialized SNN hardware cannot support these non-spike operation, as shown in Table.I.

Advancements in current specialized SNN hardware continue to pursue low power and high performance through architectural designs. However, there is still room for improvement in terms of spatiotemporal dataflow, parallelism scheme and computing engine design, particularly in field-programmable gate array (FPGA) implementations. Current FPGA SNN

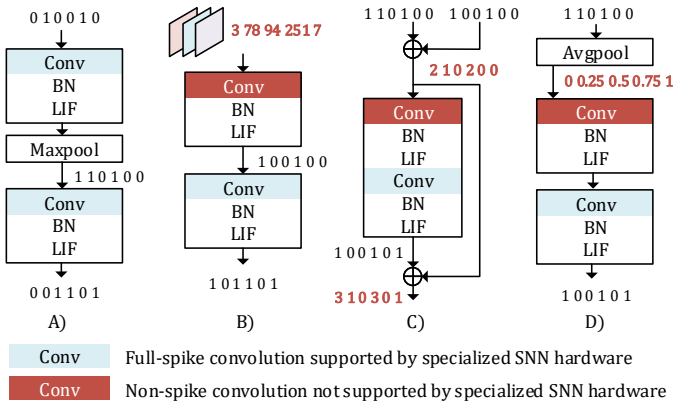


Fig. 1. A) An SNN backbone with full-spiking operation. B) The non-spiking convolution in the direct coding layer. C) The sum of spikes in SEW-ResNet introduces non-spiking convolution. D) The average pooling layer introduces fractional spike convolution.

TABLE I  
COMPARISON OVER OTHER SNN ACCELERATORS

	SpinalFlow	SATO	DeepFire2	FireFly	FireFly v2
Device	28nm ASIC	28nm ASIC	Cloud FPGA	Edge FPGA	Edge FPGA
Encode	Temp. <sup>1</sup>	Temp. <sup>1</sup>	/	Direct <sup>2</sup>	<b>Direct<sup>2</sup></b>
ResNet	Yes <sup>3</sup>	/	No	No	<b>Yes<sup>4</sup></b>
Avgpool	/	/	No	No	<b>Yes</b>
Sparse	Yes	Yes	No	No	No

<sup>1</sup> Temporal encoding.

<sup>2</sup> Although FireFly and FireFly v2 adopts direct encoding in our experiments, our hardware does not tightly coupled to the encoding scheme, while SpinalFlow and SATO only supports temporal encoded SNNs.

<sup>3</sup> SpinalFlow supports the conversion based ResNet. Addition happens before the LIF neuron. This ResNet variant does not achieve SOTA performance.

<sup>4</sup> We support the advanced SEW ResNet which achieves high accuracy.

<sup>5</sup> The slash "/" represents not mentioned.

accelerators either have no parallelism in the temporal domain [7] or sacrifice spatial parallelism for temporal parallelism [8]. Moreover, systolic-array-based SNN accelerators still have naive implementations of spiking computing engines [9] that run at low frequencies with limited parallelism. Our previous version of FireFly [10] employed a high-performance systolic array running at 300MHz with DSP optimizations, but it still had limited parallelism dimensions and ran at a frequency far from the extreme frequency on Xilinx Ultrascale FPGA. FireFly adopted a weight-stationary dataflow and designed a synaptic weight delivery hierarchy to enable efficient weight data reuse, but it still required large on-chip membrane potential storage and did not support temporal parallelism.

As agile development methodologies for customized hardware evolve, the gap between the development cycle of customized hardware and the iteration speed of algorithms is gradually closing. We recognize the importance of aligning research on SNN hardware accelerators with the advancements in SNN algorithms. In this work, we introduce FireFly v2 as another step to advance specialized hardware support for SOTA SNN algorithms while further enhancing hardware performance. FireFly v2 brings several significant improvements:

1) FireFly v2 is a general FPGA SNN accelerator that can

support A) non-spike operations in direct input encoding [5] and spike-element-wise ResNet [6]. B) multiple neurodynamics, such as IF [11], LIF [12] and RMP [13] neurons. C) arbitrary convolutional configurations such as different kernel sizes, strides, and pads. These integrations cover many recent SNN advancements.

2) FireFly v2 utilizes a spatiotemporal dataflow scheme for SNNs that enables four dimensions of parallelism. FireFly v2 not only process firing neurons on-the-fly but also generate spikes on-the-fly, eliminating the need for expensive membrane potential storage thus greatly reducing on-chip memory consumption and inference latency compared to the serial processing of spikes at each time step.

3) FireFly v2 integrates a high-performance spiking computing engine with a systolic array that supports four dimensions of parallelism and runs at 500-600MHz on different FPGA devices, which is closer to the extreme clock frequency of Xilinx Ultrascale FPGA and thus achieves  $\times 1.67 - \times 2$  throughput and DSP efficiency compared to the original FireFly [10]. Compared with the existing most advanced SNN accelerator DeepFire2, FireFly v2 achieves  $\times 1.33$  the DSP efficiency and  $\times 1.42$  power efficiency on a much smaller FPGA edge device.

The remaining sections of the paper are organized as follows: Section II presents related work that shares motivation with our research. Section III introduces how we address the non-spike operation challenge. Section IV describes our proposed spatiotemporal dataflow. Section V outlines the hardware architecture of FireFly v2, including the design of the 500-600MHz spike computing engine. Section VI provides details on experiments regarding hardware specifications and benchmark evaluations. Finally, Section VII concludes the paper.

## II. RELATED WORK

Rather than attempting to cover all neuromorphic hardware or SNN accelerators relevant to our research, we will focus on studies that share a similar motivation to our own and explore potential improvements to these works.

### A. Supporting Versatile SNNs with a Single Hardware Engine

Using a single computation engine has emerged as a favored design option for FPGA-based neural network accelerators, allowing for the deployment of a variety of models without requiring fabric reconfiguration. While ANN variants primarily differ in convolutional configurations and structural designs, SNN variants are much more varied and complex as they also differ in input encoding schemes and neuron types. However, only a limited amount of research has explored the design of a unified SNN accelerator. Cerebon [7] designed a reconfigurable compute engine compatible with a variety of spiking convolutional layers including depthwise and pointwise convolutions. Ye et al. [14] designed a neuromorphic platform that supports SNNs with MLP and CNN Topologies. Zhang et al. [15] proposed an architecture that supports multiple coding schemes including rate coding and temporal coding. However, these studies combined only address a small fraction

of the many SNN variants and did not cover the recent SNN advancements.

In this paper, our objective is to narrow the gap between modern SNN algorithms and hardware accelerators by supporting general non-spike operations. Additionally, our approach supports various convolutional configurations in a dynamically reconfigurable fashion and offers versatility in neuron types through static reconfiguration.

### B. Dataflow and Parallelism Schemes for SNNs

Existing SNN accelerators with dataflow and parallelism schemes exploration have limited parallelism dimensions. Lee et al. [8] proposed a Psum-friendly dataflow and a 2D systolic array that enables spatiotemporal parallelism. However, they only support output channel parallelism in the spatial domain, which sacrifices spatial parallelism for temporal parallelism. Spinalflow [16] presented a specialized architecture and dataflow for SNNs that exclusively supports temporal coding, where neurons only fire once in all time steps. It sorts the synaptic input spikes chronologically and can handle a maximum of 2048 non-zero spikes within the receptive field, processing spikes sequentially. It updates 128 neurons from different channels one spike packet at a time, which means that it only supports output channel parallelism. SpinalFlow achieved a performance of  $70\text{GOP}/s/mm^2$ ,  $519\times$  higher than TrueNorth. SpinalFlow did not provide direct inference latency but provided normalized inference latency over vanilla Spiking Eyeriss. SpinalFlow exhibited  $5.6\times$  speed up over vanilla Spiking Eyeriss when the sparsity level is up to 90 percent. SATO [17] expands the neuron-level parallelism to additional temporal-level parallelism by parallelizing the integration of received spikes at each time step. SATO also achieved impressive sparsity acceleration and supported workload balancing. However, similar to Spinalflow, SATO only supports temporal-coded SNNs.

In this paper, we propose a spatiotemporal dataflow with four dimensions of parallelism, namely input channel, output channel, pixel-level and time step parallelism.

### C. FPGA Accelerator with DSP Optimization Techniques

DSP optimization techniques are commonly used in FPGA-based computational-intensive accelerator designs. Here, we focus on DSP48E2 optimization techniques in Xilinx Ultra-scale FPGAs. Previous ANN accelerators have demonstrated the ability to fully utilize the capabilities that DSP slices provide. Xilinx's white paper [18] documents that the  $27 \times 18$  multiplier in DSP48E2 can be split into two  $8 \times 8$  multipliers with a shared input operand. Vitis AI has adopted the DSP double data clock technique in its DPU design. This allows the systolic array to run at double the clock frequency while the rest of the system runs at the base frequency, achieving high inference performance for FPGA platforms.

Nonetheless, these optimization techniques are tailored for multiplication-intensive ANN accelerator designs and may not be directly applicable to the SNN computation workload. The effectiveness of DSP48E2 in accelerating SNN computation is not immediately apparent. DeepFire [19] was the first

research to utilize the SIMD feature of DSP48E2s in the SNN accelerator domain. They implemented the AND operations between spikes and synaptic weights using the LUT fabric and used one DSP48E2 and three fabric adders to build an 8-input synaptic currents integration circuit. DeepFire2 [20] recognized the inefficiency of implementing a 2-input AND operation using a 6-input LUT and avoided AND operations by considering the spike as the synchronous reset of the flip-flop register. This further improved the system frequency but was still not the most ideal implementation. Our previous version of FireFly [10] proposed the most efficient implementation of synaptic operation with a fabric-free approach. We utilized a wide-bus multiplexer, SIMD adder, and dedicated cascade path inside the DSP48E2 to construct a  $16 \times 4$  synaptic crossbar using eight cascaded DSP48E2 slices. This cascaded chain was used in FireFly to build a large systolic-array-based spiking computing engine. However, fabric circuits of other system components limit the frequency in a single clock system.

In this paper, we continue to adopt the fabric-free implementation of synaptic operation proposed in our previous version of FireFly but with a different dataflow. We further incorporate DSP double data rate techniques adopted by the Vitis AI DPU. As a result, FireFly v2 achieves significant speed up compared to its previous version.

## III. ADDRESSING THE NON-SPIKE OPERATION CHALLENGE

In recent SNN advancements, the inclusion of non-spike operations has been proven to be a challenging task for specialized SNN hardware. Without loss of generality, we consider three typical non-spike scenarios: pixel operation in direct encoding, multi-bit spike operation in SEW ResNet, and fractional spike operation introduced by average pooling.

Instead of designing separate hardware for these different non-spike operation cases, FireFly v2 draws inspiration from the field of bit-serial accelerators to address the non-spike operation challenge. In bit-serial accelerators [21], operands are divided into smaller bits, computations are carried out using low-bit arithmetic logic and the partial sums are shifted and merged to reconstruct full-precision results. In FireFly v2, we utilize a single spike computing engine to perform spike-weight multiplications, decompose the non-spike operands and flexibly merge the partial sums to support non-spike operations. We focus on the three aforementioned common scenarios of non-spike operations.

a) *Pixel Convolution in Direct Encoding Layer:* In the case of 8-bit pixel convolution using direct input encoding, we treat the 8-bit pixel as spikes occurring over 8 equivalent time steps. We then perform spike-weight convolution for each time step and combine the 8 partial sums using shift-add logic.

b) *Multi-bit Spike Convolution in SEW-ResNet:* In a certain SEW ResNet layer, a  $B$ -bit spike in  $T$  simulation time steps can be deconstructed into a sequence of binary spikes spanning  $B \times T$  equivalent time steps. Partial sums are then shifted and merged  $B$  in a group to reconstruct the actual partial sums of  $T$  actual time steps. As the sum of spikes accumulates, SEW-ResNet will produce  $\log(N+1)$ -bit spikes

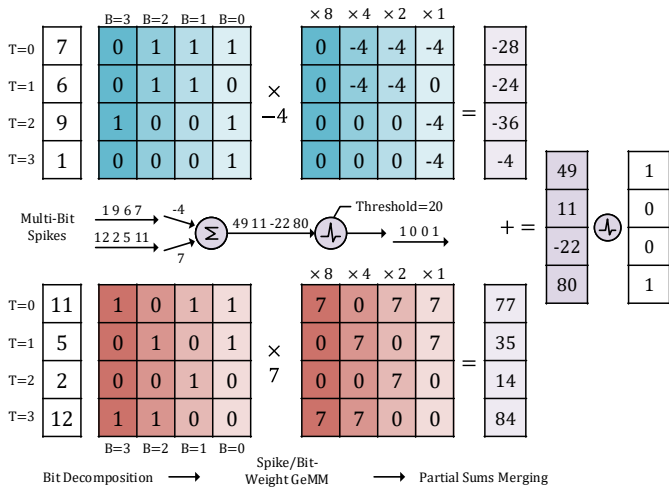


Fig. 2. Addressing the computation of multi-bit spikes through bit decomposition, bit-weighted GeMM, and partial sums merging. The figure illustrates a case of 4-bit spikes computation.

after  $N$  spike-element-wise residual connections. With a 4-bit spike representation, this accommodates up to 16 residual connections, aligning with the SEW-ResNet34 architecture.

*c) Fractional Spike Convolution introduced by Average-Pooling:* The fractional values can be left-shifted to integers, treated as multi-bit spikes, and the partial sums can be right-shifted by the same amount to obtain the accurate partial sum. For example, consider a  $2 \times 2$  average pooling operation that yields values such as 0, 0.25, 0.5, 0.75, 1. These values can be left-shifted by 2 positions, resulting in 0, 1, 2, 3, 4, and treated as 3-bit spikes.

Fig. 2 illustrates the computation flow for 4-bit spikes with  $T = 4$ . In this scenario, the spiking neuron receives two 4-bit spike inputs from synapses and generates binary spikes. The 4-bit spike sequence is initially decomposed bit by bit, creating a  $4 \times 4$  bit matrix. Each bit is then multiplied with the corresponding synaptic weight, and the partial sums of each bit are further scaled by factors of 8, 4, 2, and 1 (or left-shifted by 3, 2, 1, and 0), respectively. These scaled values are then summed together, resulting in the actual synaptic current at each time step. Subsequently, these synaptic currents are accumulated and reset when they reach a certain threshold, leading to the generation of spikes according to specific neurodynamic behaviors.

Although the bit-serial decomposition of the multi-bit spikes can fully support the non-spike operations existing in state-of-the-art SNN algorithms without any accuracy drop, it inevitably leads to an increased computational workload. After three cascaded residual connections in SEW ResNet, the maximum value of spikes reaches 4, necessitating a 3-bit representation. Similarly, spikes with fractional values yielded by the most commonly seen  $2 \times 2$  average pooling also need a minimum 3-bit representation. To prevent the potential escalation of spike bit-width as the SNN network increases in depth, we introduce a saturate-or-shift approach to confine the bit-width of spikes within 2 bits. In our approach, an analysis of data distribution will be conducted in advance on a batch of

TABLE II  
NOTATIONS

Notation	Description
$T$	Time Step
$H_o, W_o$	Size of Output Feature Map
$H_i, W_i$	Size of Input Feature Map
$K_h, K_w$	Kernel Size
$C_o, C_i$	Input Output Channels
$M$	Output Channel Parallelism
$V$	Input Channel Parallelism
$N$	Pixel Parallelism
$S$	Time Step Parallelism

representative samples in software, similar to the post-training quantization. This analysis will guide the decision of whether to saturate the spike value exceeding 4 to 3, or perform a right shift operation on all spike values, resulting in the range of 0, 1, 2, accompanied by a corresponding left shift of the partial sum to recover the results. This method ensures that the bit-width of spikes remains confined within 2 bits, effectively mitigating the escalation of the computational workload. In our analysis, we empirically find that the same saturation configuration for all non-spike layer could yield negligible accuracy drop. Nevertheless, this saturation-or-shift hardware function is reserved for more fine-grained layer-wise analysis.

#### IV. OPTIMIZED SPATIOTEMPORAL DATAFLOW

Although the dataflow of ANNs has undergone extensive study, achieving the balance between the spatial and temporal dimensions in the dataflow of SNNs proves to be a challenging task. In FireFly v2, we propose a spatiotemporal dataflow that builds upon the output stationary dataflow while incorporating variable tiling and parallelism schemes designed specifically for SNNs. In contrast to the previous version of FireFly [10], our approach significantly reduces memory consumption and enables a higher degree of parallelism and reconfigurability.

We focus on the dataflow of a single convolution layer. Although FireFly v2 does support non-spike convolution, we focus on the 1-bit spike case in this section to simplify the dataflow illustration. We first explain the variables used in this paper, which are also listed in Table.II.  $T$  represents the total number of time steps.  $H_o$  and  $W_o$  represent the height and width of the output feature map, respectively, while  $H_i$  and  $W_i$  represent the height and width of the input feature map, respectively.  $K_h$  and  $K_w$  denote the height and width of the kernel, while  $C_i$  and  $C_o$  represent the input and output channels, respectively. We leverage four dimensions of parallelism and perform variable tiling on the output channel  $C_o$ , input channel  $C_i$ , the width of the output feature map  $W_o$ , and equivalent time step  $T_e$ . We denote the output channel parallelism as  $M$ , input channel parallelism as  $V$ , pixel parallelism as  $N$  and time step parallelism as  $S$ .

Similar to convolution in ANNs, convolution in SNNs can be expressed using nested for-loops, accompanied by an additional time step loop [16] [17]. The permutation of loop order does not alter the computation results, yet it does influence data locality and data reuse opportunities [22]. To simplify the illustration, we employ an ordered tuple to represent the



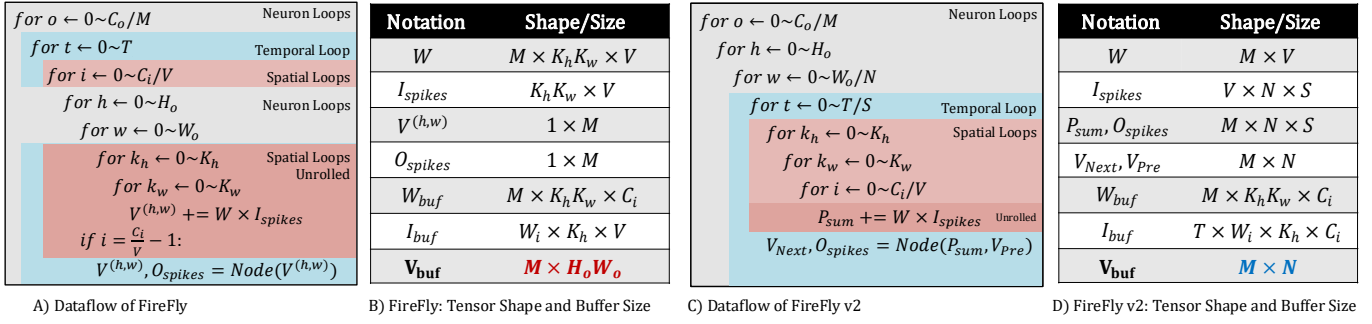


Fig. 3. Comparison of the dataflow in FireFly v2 to its previous version FireFly. A) FireFly’s dataflow design. B) Mentioned tensor shape or buffer size in FireFly’s dataflow. C) FireFly v2’s dataflow design. D) Mentioned tensor shape or buffer size in FireFly v2’s dataflow.

permutation of folded or unrolled for-loops, with the unrolled for-loops enclosed in square brackets, denoted as  $\{\}$ . The fully folded computation flow of an SNN convolution layer can be represented as  $(T, C_o, H_o, W_o, C_i, K_h, K_w)$ . Here, we employ a similar loop notation as presented in SATO [17]. The dimensions  $(C_o, H_o, W_o)$  correspond to the neuron loops, representing independent neurons within a convolutional layer, indicated by the color blue in Fig.3. On the other hand, the dimensions  $(C_i, K_h, W_h)$  represent the spatial loops, denoting spatially fan-in neurons and are marked with the color orange.

a) *Inefficient Dataflow in FireFly*: In the previous version of FireFly, we paralleled the computation for both the input-output channel dimension and the kernel dimension. This resulted in a dataflow represented as  $(\frac{C_o}{M}, T, \frac{C_i}{V}, H_o, W_o, \{M, K_h, K_w, V\})$ , as depicted in Fig. 3A. The spike computing engine within FireFly conducts matrix-vector multiplication between the  $K_h K_w \times V$  binary spikes and the  $M \times K_h K_w \times V$  synaptic weight matrix, yielding a  $1 \times M$  partial sum. Subsequently, when the last fragment of spikes from the tiled input channel passes through, the neurodynamics calculation is performed to generate a  $1 \times M$  output spike vector. To achieve weight data reuse across  $T$  time steps, an on-chip cache is necessary to store  $M \times K_h K_w \times C_i$  synaptic weights. To avoid the need for off-chip storage and loading of multi-bit membrane potential, it becomes imperative to temporarily store  $M \times H_o W_o$  membrane potential values on-chip. However, this poses potential issues, particularly when dealing with large feature maps. The tensor shapes or buffer sizes in FireFly’s architecture are denoted in Fig.3B.

b) *Spatiotemporal Dataflow in FireFly v2*: In FireFly v2, we tackle the following limitations in the FireFly architecture’s dataflow: 1) The need for large on-chip storage of membrane potential. 2)The constraint of a fixed convolution configuration resulting from parallelism on the kernel dimension. 3) The absence of parallelism at both the temporal and pixel levels. The adopted spatiotemporal dataflow scheme is depicted in Fig.3C. To address the long data dependencies spanning  $C_o \times H_o \times W_o$  neurons across  $T$  time steps, we’ve rearranged the loops by placing the temporal loop after the neuron loops and before the spatial loops. We have also tiled on both the width of the output feature map  $W_o$ , and the time step  $T$ , to introduce two additional dimensions of parallelism. After the reordering and tiling, the resulting dataflow takes the form of  $(\frac{C_o}{M}, H_o, \frac{W_o}{N}, \frac{T}{S}, K_h, K_w, \frac{C_i}{V}, \{M, V, N, S\})$ , shown

in Fig.3C. The spike computing engine performs matrix multiplication between the  $V \times N \times S$  binary spike matrix and the  $M \times V$  weight matrix, yielding a  $M \times N \times S$  partial sum matrix. After accumulating the fan-in pre-synaptic currents across the  $K_h, K_w, \frac{C_i}{V}$  spatial loops, neurodynamic calculations are carried out by incorporating the  $M \times N \times S$  partial sums along with the residual  $M \times N$  membrane potential  $V_{Pre}$  from the previous time step batch. This process results in the generation of  $M \times V \times S$  output spikes and the next residual membrane potentials  $V_{Next}$ . The aforementioned tensor shapes or buffer sizes in FireFly v2’s architecture are denoted in Fig.3D.

This spatiotemporal dataflow framework enables four dimensions of parallelism including output channel parallelism, input channel parallelism, pixel-level parallelism, and time step parallelism. Through the separation of the parallelism scheme from the kernel dimension, we enable support for various convolution configurations, accommodating differences in kernel size and stride. By positioning the temporal loop as the innermost loop before the spatial loops, the necessity to cache only  $M \times N$  residual membrane potentials on-chip becomes inconsequential. Without the explicit storage of the membrane potentials, spikes are not only processed but also generated on-the-fly. The only additional overhead is the need for increased storage space for input spikes. Given that input spikes require much less memory storage compared to membrane potentials, this added requirement is of minimal concern.

## V. HARDWARE ARCHITECTURE

The hardware architecture of FireFly v2 is illustrated in Fig.4. Fig.4A provides an overview of FireFly v2’s system block diagram. In the customized PL system of the Zynq Ultrascale SoC, a clock wizard IP is instantiated to generate two synchronous clocks, with one clock operating at twice the frequency of the other. One master M-AXI-HPM port of the Zynq Ultrascale SoC is utilized for command configuration status control, directly connected to the FireFly v2 IP. Two 128-bit S-AXI-HP ports are enabled and are connected to two read-only AXI DataMovers respectively, facilitating high-speed PS to PL data transfer. Additionally, another 128-bit S-AXI-HP port is employed and connected to a write-only AXI DataMover, enabling PL to PS data transfer. The FireFly v2 IP interfaces with all three AXI DataMovers. Please note that we have not utilized all of the S-AXI-HP ports available on

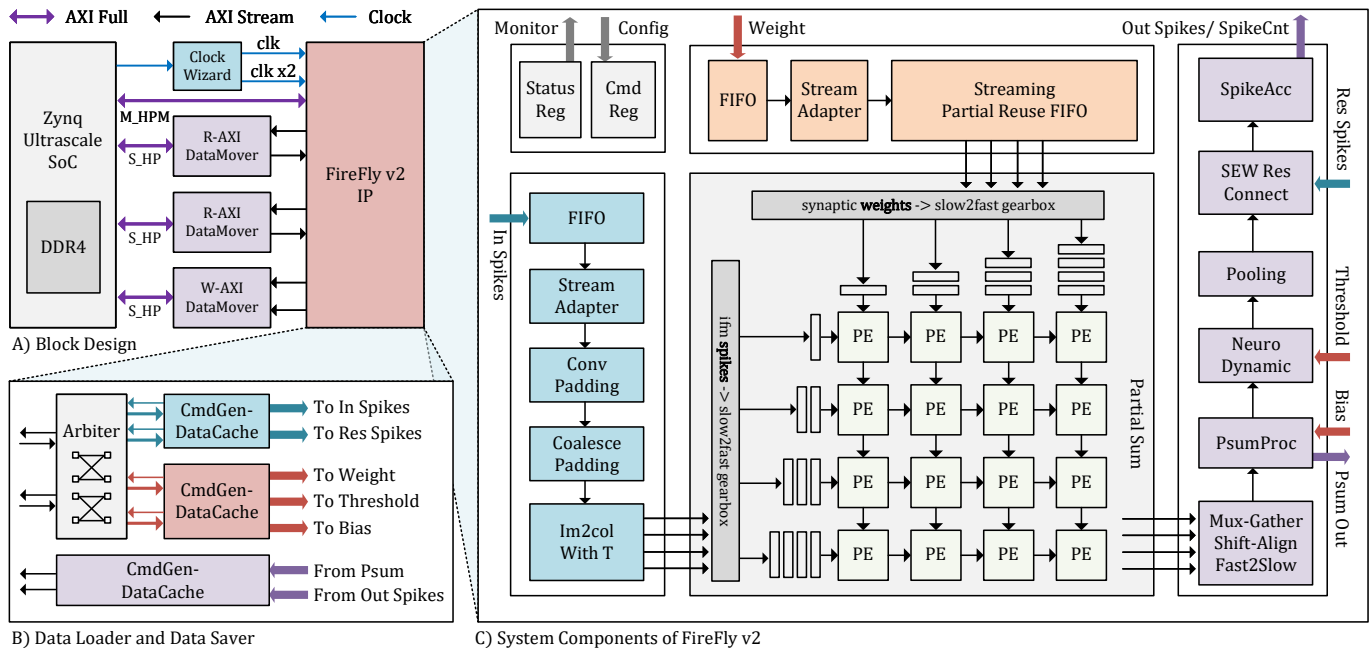


Fig. 4. Hardware Architecture of FireFly v2. A) The system block design of FireFly v2. B) The efficient data loader and data saver of FireFly v2. C) The key system components of FireFly v2. The blue blocks represent a series of input spike preprocessing modules, whereas the orange blocks signify a sequence of synaptic weight preprocessing modules. The purple blocks represent the post-processing modules for the partial sums and output spikes.

the Zynq SoC, and the peak memory bandwidth of the three instantiated S-AXI-HP ports is far from the Zynq Ultrascale device’s limit of 19.2 GB/s. We constrain ourselves in such setup to reduce power consumption and ease the routing complexity of the system design, while ensuring the required bandwidth is met.

We have designed an efficient data loader and data saver module to fully harness the bandwidth capacity of the three S-AXI-HP ports, as illustrated in Fig.4B. We have instantiated two CmdGen-DataCache Units to generate Address-Length data transfer commands for input or residual spikes, as well as parameters such as weights, bias, and thresholds, respectively. Additionally, another CmdGen-DataCache is dedicated to handling output spikes or partial sums transfer, utilizing the write-only AXI DataMover interface. Given the availability of two read-only AXI DataMover interfaces, the simplest approach would be to assign each CmdGen-DataCache unit to occupy one AXI DataMover interface. However, in various convolutional configurations, the required data bandwidth for input spikes and synaptic weights differs. For instance, in a  $1 \times 1$  convolution, the synaptic weights demand relatively less bandwidth, while input spikes require more. Conversely, in a  $3 \times 3$  convolution with small  $4 \times 4$  images, the synaptic weights require greater bandwidth, while the input spikes need less. To efficiently utilize the total available bandwidth and prevent one AXI DataMover from being fully utilized while the other remains idle, we have introduced an arbiter. This arbiter arbitrates incoming commands from the CmdGen-DataCache units and the data flow from the AXI DataMover interfaces, enabling flexible bandwidth balancing for varying spikes and weight workloads.

The key system components of FireFly v2 are arranged as

illustrated in Fig. 4C. The blue blocks in Fig.4C represent a series of input spike preprocessing modules, whereas the orange blocks signify a sequence of synaptic weight preprocessing modules. The purple blocks represent the post-processing modules for the partial sums and output spikes. The input spike stream and synaptic weight stream pass through the preprocessing modules before reaching the spike computing engine. Subsequently, the spike computing engine generates a partial sums stream, which is then further processed by the post-processing modules to produce output spikes. We will provide a brief overview of these key components below.

*a) Input Data Preprocessing:* The input spike preprocessing modules initially buffer and adjust the input spike stream’s stream width via a FIFO and stream width adapter. Subsequently, two sub-modules handle stream padding. One handles zero-padding based on the current convolution padding configurations, while the other focuses on memory coalescing to prevent bank conflicts. After padding, the data stream enters the im2col unit sequentially but is then read out in a strided fashion to perform the im2col transformation. The im2col unit comprises  $N$  memory banks, and a strided address generator generates  $N$  conflict-free addresses for these banks to read the spike data. An  $N$ -port crossbar is responsible for routing the data output from the memory banks to their respective ports, thereby delivering the data stream to the spike computing engine. The input weight stream undergoes a similar buffering and width adjustment procedure and then is pushed to the partial reuse FIFO, a component introduced in our prior version of FireFly [10].

*b) Clock Crossing in the Computing Engine:* The spike computing engine joins the spike stream from the im2col unit with the weight stream from the partial reuse FIFO and

sends them to the fast clock region. The spike computing engine consists of two slow-to-fast gearboxes, functioning as parallel-to-serial converters. These gearboxes reduce the data elements by half while doubling the clock rate, maintaining the data bandwidth. The PEs (processing elements) of the spike computing engine operate at the fast clock region, performing matrix multiplications between the  $V \times N \times S$  spike stream and the  $M \times V$  weight stream and generating  $M \times N \times S$  partial sum every  $K_h \times K_w \times \frac{C_i}{V}$  cycles. The partial sums are then gathered, aligned and sent back to the slow clock region.

c) *Partial Sums Postprocessing*: The post-processing modules first flexibly process the partial sums, dealing with spike and non-spike cases, and then generate spikes through the neurodynamic unit. The following pooling unit performs optional Maxpooling and AvgPooling or just bypasses the spike stream if pooling is not needed. The residual connection module of the FireFly v2 performs the optional spike-element-wise residual connection between the shortcut spikes from the data loader and the calculated spike stream. The spike accumulation module optionally counts spikes from the spike stream to record firing rates, a common operation in the last classification layer in SNN models. The output spike stream flows back to the external memory map through the data saver and serves as the input spike stream for the subsequent layer.

In the following subsections, we will first fully elaborate on the design of the spike computing engine. Next, we will provide thorough details of how the spike computing engine and the partial sum merging unit cooperate to perform non-spike operations, which is essential in supporting direct encoding and multi-bit spike convolution. Additionally, we will delve into the two-phase design of the neurodynamics unit which can generate spikes across multiple time steps. Lastly, we will present the residual connection unit that supports spike-element-wise functions in various cases.

### A. Spatiotemporal Spiking Computing Engine

The spike computing engine acts as the core of FireFly v2 since it is responsible for the heavy computing workload. *The key aspect of the spiking computing engine lies in its operation at twice the frequency, detached from the low-speed fabric, while supporting spatial and temporal parallelism.* We adopt the systolic architecture same as FireFly [10], shown in Fig.5A, but with several distinctions: 1) FireFly employs a weight-stationary systolic array, whereas FireFly v2 implements an output-stationary systolic array. This choice is better aligned with our spatiotemporal dataflow requirements. 2) The systolic array in FireFly enables spatial parallelism across input channels, kernel sizes, and output channel dimensions. However, the parallelism in the kernel dimension imposes constraints on the convolution scheme, as FireFly exclusively supports  $3 \times 3$  convolutions. In contrast, FireFly v2 leverages spatiotemporal parallelism in input channels, output channels, pixel level, and the time step dimension. We support various kinds of convolution configurations enabled by the flexible im2col unit. 3) In FireFly, the systolic array operates at a frequency of 300MHz, identical to the overall system clock frequency. FireFly v2 successfully decouples the slower fabric

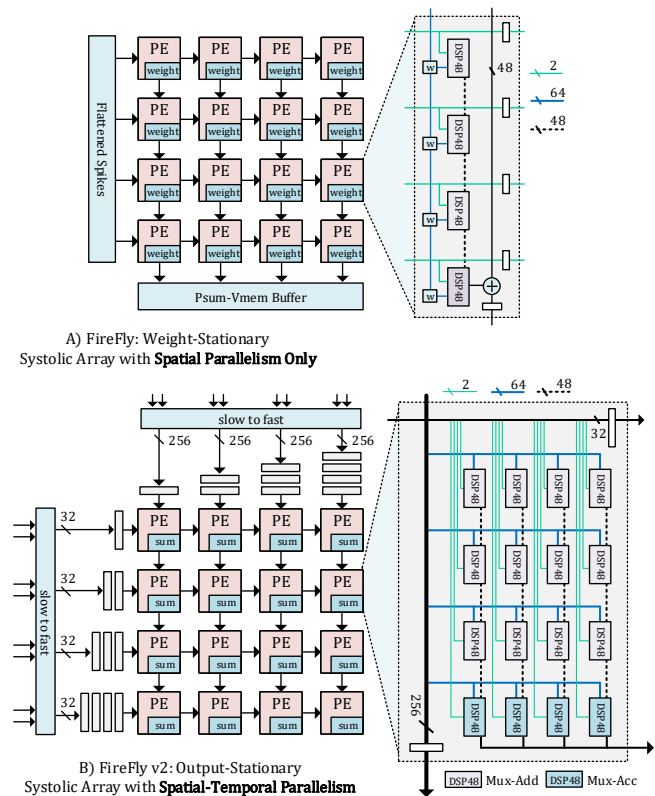


Fig. 5. Comparison of the spike computing engine in FireFly v2 to its previous version of FireFly. A) Spike computing engine in FireFly supporting only spatial parallelism B) Spatiotemporal spike computing engine in FireFly v2.

TABLE III  
THE SIZE OF THE SYSTOLIC ARRAY AND THE PE

Notation	Description	Determined by
$SA_h$	Systolic Array Height	$M/4$
$SA_w$	Systolic Array Width	$N$
$PE_h$	PE Height	$V/4$
$PE_w$	PE Width	$S$

logic from the faster DSP unit. This method enables the spike computing engine to operate at 500-600MHz, doubling its performance capabilities compared to FireFly.

Designing a high-performance systolic array is non-trivial. To bridge the gap between the operating frequency of DSP48E2 and its theoretical extreme frequency, we adopt the DSP double data rate technique as the Vitis AI DPU. We follow three key principles: First, the circuits in the doubled-frequency domain should be well-decoupled from the circuits in the low-frequency domain. Second, we avoid the use of LUTs in the doubled frequency domain, and instead, only use DSP48E2 and flip-flops. Third, we avoid high-fanout nets and instead use simple and local connections between circuit components. This helps to minimize net delays and reduce congestion.

Before sending the spike sub tensor and weight sub tensor to the spiking computing engine, a slow-to-fast converter, or the gearbox, is utilized to facilitate communication between circuits operating at different frequencies. This gearbox, which

operates at twice the frequency of the low-frequency domain, essentially functions as a multiplexer, selecting the data being transmitted from the low-frequency domain. The data being transmitted from the low-frequency domain has twice the data elements, but the data being transmitted to the doubled-frequency domain has twice the clock rate. As a result, the bandwidth at each side of the gearbox remains the same.

The spiking computing array is organized as a systolic array, with processing elements (PEs) arranged in a 2D fashion. To simplify the depiction, Fig.5B only illustrates a systolic array with  $4 \times 4$  PEs. The array employs an output stationary dataflow, with weight inputs flowing vertically from top to bottom and spike inputs flowing horizontally from left to right. Partial sums are stored in each PE and are collected once the accumulation process is complete. Each processing element (PE) comprises several columns of DSP48E2s. To simplify the illustration, Fig.5B only depicts a single PE with four columns of DSP48E2s, where each column contains four DSP48E2s cascaded in a chain. Similar to the previous version of FireFly [10], a single DSP48E2 slice functions as a  $2 \times 4$  synaptic crossbar, receiving two 1-bit spike inputs and eight 8-bit weight inputs. The dedicated cascaded path of the DSP48E2 in the same column behaves like dendrites, integrating the synaptic current through the DSP48E2 adder chain. Each column of the processing element produces four 12-bit partial sums, utilizing the single instruction, multiple data (SIMD) feature of DSP48E2. Within the same PE, DSP48E2s on the same row share the same weight inputs, while each DSP48E2 has its own spike inputs. In the illustrated example of a single PE with  $4 \times 4$  DSP48E2s, it receives  $4 \times 8 \times 8 = 256$ -bit weight inputs and  $4 \times 4 \times 4 = 32$ -bit spike inputs, generating  $4 \times 4 \times 12 = 192$ -bit partial sums. The weights and spikes are staged and then fed to the adjacent PEs, and the partial sums are collected after the accumulation process is completed.

The parallelism factors  $M, V, N, S$  play a vital role in determining the dimensions of the systolic array. Specifically, Table.III outlines the relationship between these factors and the corresponding dimensions of the systolic array and PEs. The height of the systolic array  $SA_h$  is determined by  $\frac{M}{4}$ , where each column of DSP48E2 in one PE can compute 4 channels. The width of the systolic array  $SA_w$  is directly equal to  $N$ . The height of the PE  $PE_h$  is determined by  $\frac{V}{4}$  due to the fact that the computing engine operates at a doubled frequency, and each DSP48E2 in one PE can integrate two synaptic currents. The width of the PE is determined by  $S$ . It's worth noting that within a single Processing Element (PE), synaptic weights are broadcast to  $S$  columns of DSP48E2 units. A critical consideration here is the choice of  $S$ , where a larger value would lead to a larger fan-out for synaptic weights, potentially failing step-up requirements. Conversely, opting for a smaller value of  $S$  would elevate the consumption of flip-flops. Based on experimental insights, we have determined the optimal value for  $S$  to be 4. This empirical setting strikes a balance between managing fan-out effects and optimizing flip-flop usage. We use different  $SA_h$  and  $SA_w$  in different FPGA devices with different amounts of resources.

The spike computing engine generates  $M \times N \times S$  partial sums every  $K_h \times K_w \times \frac{C_i}{V}$  cycles. Given that  $K_h \times K_w \times \frac{C_i}{V}$  is

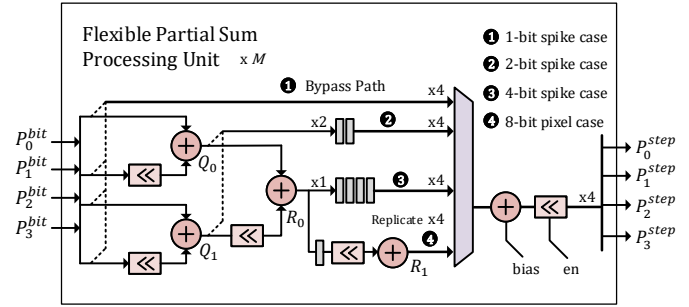


Fig. 6. Flexible partial sum processing unit dealing with four spike or non-spike cases.

larger than  $N$  in most scenarios, we aggregate the partial sums  $N$  in a group, align the partial sums from  $\frac{M}{4}$  PE columns and use a cross clock region FIFO to transfer the  $M \times N \times S$  partial sums back to slow clock region, shown in Fig.4.

### B. Flexible Partial Sum Processing Unit

The key aspect of the flexible partial sum processing unit is that it performs partial sums aggregation for the multiple spike or non-spike input scenarios. Multi-bit spikes are decomposed into equivalent time steps using the same spike computing engine to compute the decomposed partial sums. The flexible partial processing unit shown in Fig.6 reconstructs the partial sums by shift-add logic. The processing unit consists of  $M$  identical sub-modules to processing  $M$  channels of partial sums. As stated in the previous subsection,  $S$  is set to an empirical value of 4, so each sub-module receives partial sums, namely  $P_0, P_1, P_2, P_3$ , of 4 equivalent time steps, shown in Fig.6. The processing unit can handle 4 cases:

- 1) In cases where the input spike is binary, four partial sums are bypassed and directly sent to the next stage.
- 2) When dealing with a 2-bit input spike, two adjacent partial sums are shifted-merged, yielding  $Q_0 = P_0 + (P_1 \ll 1)$  and  $Q_1 = P_2 + (P_3 \ll 1)$ . The processing unit waits for another round of the shift-merge process to collect 4 partial sums and send them to the next stage.
- 3) When dealing with a 4-bit input spike, all four partial sums are shifted-merged, yielding  $R_0 = Q_0 + (Q_1 \ll 2)$ . The processing unit must wait for three additional rounds of the shift-merge process to gather four partial sums before transmitting them to the next stage.
- 4) When dealing with direct input coding where the input pixel is 8-bit, eight partial sums are shifted-merged, yielding  $R_1 = R_0 + (R_0 \ll 4)$ , in which  $R_0$  is the previous round of  $R_0$  temporarily stored in registers. In direct encoding, the convolution results of the static images are replicated  $T$  times and sent to the neurodynamics unit. Therefore, we directly replicate  $R_1$  for four times and send them to the next stage.

After the partial sums are shifted-merged, the bias is added, and the partial sums can be optionally left-shifted by 1 if the input spikes from the preceding layer are right-shifted by 1. The input precision of the partial sums is 12-bit. After being shifted and merged by the processing unit, the output precision of the partial sums is extended to 18-bit.



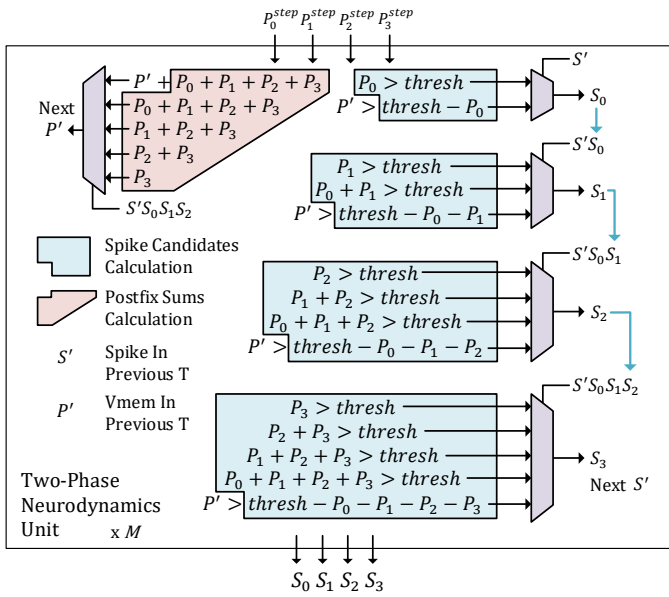


Fig. 7. Neurodynamics unit. The blue-shaded logic performs independent calculations of spike candidates for each time step. The orange-shaded logic, on the other hand, handles the calculation of postfix sums for each time step. The purple multiplexer selects the spike candidates and postfix sums, generating both the output spikes and the residual membrane potential.

### C. Two-Phase Neurodynamics Unit

The key insight of designing the neurodynamic unit is to support parallel spike generation given the synaptic currents already available at each time step. After the partial sums are flexibly merged (or just bypassed in the 1-bit spike case), the partial sums flow into the neurodynamics unit. In extreme cases, partial sums flowing into the neurodynamics unit are consecutively valid, the neurodynamics unit needs to process partial sums, generate spikes for all  $S$  time steps and calculate the membrane potential for the next batch of time steps in just a single clock cycle. The naive implementation of the neurodynamics unit is similar to the idea of the ripple carry adder, where each bit's sum depends on the carry generated by the previous bit, causing a serial carry propagation through the circuit. In the spike generation process, each spike and membrane potential depends on the previous spike and membrane potential, causing a series of data propagation through the circuit. It is impossible to achieve timing closure at 250-300MHz with such high logic levels.

To address this problem, we design a two-phase neurodynamics unit to tackle this problem. We take inspiration from the carry look-ahead adder, where the carry signals are pre-computed for each bit, enabling parallel carry calculation and faster addition. We decompose the spike generation process in two phases. Fig. 7 shows the integrate-and-fire neurodynamics generation process. In phase 1, we precompute the postfix sums of the partial sums of  $S$  time steps and compare them to the threshold, yielding the spike candidates of each time step. In this phase, calculations are pipelined since they have no data dependency on previous time steps. In phase 2, we select the spikes from these precomputed candidates based on spikes selected in previous time steps, as shown by blue arrows in

Fig. 7. In this phase, timing closure can be satisfied since the accumulation of logic levels is only determined by the number of cascaded multiplexers and comparators.

The neurodynamics unit is statically reconfigurable to support integrate-and-fire [11], leaky-integrate-and-fire [12] and residual membrane potential [13] neurons since most SNN models adopt a single neuron type across all layers. Designs of the neurodynamics unit of different neuron types share the same methodology.

### D. Residual Connection Unit

The key aspect of the residual connection unit is that it needs to support both IAND and ADD spike-element-wise add function. The residual connection unit receives spikes of the current backbone from the flow-to-stream unit and receives spikes of the shortcut branch from the data loader. The residual connection unit performs IAND or ADD spike-element-wise function to the two spike streams.

To align with the 8-bit byte standard, the bit-width values of spikes from the shortcut branch are restricted to one, two, or four. The bit-width of spikes from the current backbone is one in most cases unless the optional average pooling datapath is selected. In that case, the bit-width increases to two. FireFly v2 does not support residual connection after the backbone is downsampled by average pooling since such situations are not typical in most SNN models. Therefore, when performing residual connections, the bit-width of spikes from the current backbone is always one.

The residual connection unit contains dedicated logic of the IAND function for binary shortcut spikes and low-bit spike-element-wise ADD function for different shortcut spikes' bit-width. The IAND function always produces binary spikes, which is the most hardware-friendly spike-element-wise function. When performing the spike-element-wise ADD function, if the added results exceed the representation range of a two-bit integer, we can extend the added results to four bits or adopt the saturate-or-shift method to constrain the results back to two bits. If the added results exceed the representation range of a four-bit integer, we will directly saturate the results to four bits.

## VI. IMPLEMENTATION AND EXPERIMENTS

### A. Experiments Setup

Similar to its previous version, FireFly v2 targets FPGA edge devices to cut down the budget in real-world applications. FireFly v2 is mapped onto Ultra96v2, KV260 and ZCU104 evaluation boards with different  $(M, V, N, S)$  configurations. In Table VII, we present the total resources of the device employed. FireFly v2, when mapped onto these devices, utilizes a minimum of over fifty percent of the available resources.

FireFly v2 is designed using SpinalHDL. The Verilog codes generated by the SpinalHDL compiler are synthesized and implemented in the Xilinx Vivado 2022.2. Power consumption estimates are provided by the reports of the Vivado Design Suite. FireFly v2 is based on the Brain-Inspired Cognitive Engine (BrainCog) [23] and is another step toward the software-hardware codesigns for the BrainCog project

TABLE IV  
THEORETICAL AND ACTUAL INFERENCE LATENCY FOR TYPICAL SPIKING CONVOLUTION CASES ( $16 \times 16 \times 8 \times 4$  CORE@250MHZ, LATENCY MEASURED IN US)

$C_i, C_o$	Size	$K_{h,w}, Stride, Pad$	$T, B$	Theory	Actual
32, 64	64, 64	3, 1, 1	4, 1	147.5	151.8
32, 64	64, 64	3, 1, 1	4, 2	294.9	302.5
32, 64	64, 64	7, 2, 3	4, 1	200.7	210.8
16, 32	128,128	3, 1, 1	8, 1	294.9	305.3

(<http://www.brain-cog.network/>) [24]. All the evaluated SNN models are trained using the BrainCog’s infrastructures.

## VII. ANALYTICAL MODEL

The theoretical clock cycles required for a spiking convolutional layer can be represented by Equ.1. Throughout  $CC$  clock cycles,  $C_i H_i W_i T B$  input bits are retrieved from external memory. The average bandwidth required (Bits per Cycle) for input data can be depicted by Equ.2. Over  $CC$  clock cycles,  $C_o K_h K_w C_i$  8-bit synaptic weights are utilized. The average bandwidth required (Bits per Cycle) for synaptic weights is described by Equ.3.

$$CC = \left\lceil \frac{C_o}{M} \right\rceil H_o \left\lceil \frac{W_o}{N} \right\rceil K_h K_w \left\lceil \frac{C_i}{V} \right\rceil \left\lceil \frac{TB}{S} \right\rceil \quad (1)$$

$$BW_{input} = \frac{H_i W_i N V S}{H_o W_o K_h K_w} \quad (2)$$

$$BW_{weight} = 8 \times \frac{H_o W_o T B}{M V N S} \quad (3)$$

Considering a standard setup of  $16 \times 16 \times 8 \times 4$  parallelism and a  $16 \times 16$  binary spiking feature map with 4 time steps, the average bandwidth needed for synaptic weights is 64 bits per cycle. For a convolution configuration of a  $3 \times 3$  kernel with unit stride and same padding, the required bandwidth for input data is 57 bits per cycle.

The typical bandwidth requirements for input spikes and synaptic weights are generally met in most scenarios. We’ve performed multiple experiments using various spiking convolution configurations to compare the actual latency against the theoretical latency. These experiment findings are detailed in Table.IV. Remarkably, our streaming processing ensures that the actual latency closely aligns with the theoretical latency.

### A. Comparison with the Previous Version of FireFly

Table.V shows the comparison between FireFly v2 and its previous version FireFly in hardware specifications. In terms of LUT consumption, FireFly v2 mapped on Ultra96 consumes slightly more LUTs than FireFly. This difference arises from the increased complexity of the overall architecture in FireFly v2. However, FireFly v2 mapped on ZCU104 is roughly the same as FireFly since the proportion of the resource consumption taken up by the computing array becomes more significant as parallelism increases and FireFly v2 adopts a DSP-only and fabric-free spike computing engine.

In terms of DSP48E2 consumption, FireFly’s DSP48E2 consumption aligns with multiples of 9 since FireFly seeks

parallelism in kernel dimension by flattening the  $3 \times 3$  kernel window computation, while FireFly v2’s DSP48E2 consumption aligns with multiples of 8 with each dimension in FireFly v2’s parallelism being the power of two. Consequently, FireFly v2’s DSP48E2 consumption is equivalent to  $\frac{8}{9}$  of FireFly’s consumption on the same device. In terms of the DSP efficiency, power efficiency and throughput performance, FireFly v2 mapped on Ultra96 achieves the highest clock frequency of 600MHz and the highest peak DSP efficiency of 9.6 GOP/s/DSP, which is doubled compared to FireFly. The DSP efficiency improvement of FireFly v2 compared to its previous version is primarily attributed to the increased clock frequency. FireFly v2 mapped on KV260 achieves the highest peak power efficiency of 835.9 GOP/s/W, maintaining a low power draw of 4.9W. FireFly v2 mapped on ZCU104 achieves the highest peak throughput of 8192 GOP/s.

FireFly v2 mapped on Ultra96 can reach 600MHz with PerformaceWithRemap implementation strategy set in Vivado Design Suite. However, this strategy induces higher power consumption. But still, FireFly v2 can achieve similar power efficiency compared to FireFly on the same Ultra96 device.

FireFly v2 mapped on KV260 cannot reach 600MHz even with PerformaceWithRemap strategy being enabled. This limitation arises from the considerably inherent smaller CLB:DSP ratio of  $\frac{93}{1}$  in KV260 in comparison to Ultra96 with CLB:DSP ratio of  $\frac{196}{1}$ . This translates to a higher likelihood of routing congestion that will cause degrade in frequency performance. Nevertheless, FireFly v2 mapped on KV260 can reach timing closure at 500MHz and achieve excellent power efficiency when using PowerDefaultOpt implementation strategy. Since power consumption is tightly coupled to the clock frequency, we also run multiple experiments at lower frequencies using the same implementation strategy and find that FireFly v2 running at 500MHz achieves the best power efficiency, shown in Table.VIII. We also try a higher level of parallelism in KV260 since a  $16 \times 16 \times 8 \times 4$  configuration only utilizes 40% of DSP48E2s. A  $32 \times 16 \times 8 \times 4$  configuration can meet timing closure at 400MHz. Note that in this configuration we halve the depth of the local weight cache depth and the FIFO size of the AXI DataMover, reducing the BRAM consumption to relieve the tight setup requirements.

FireFly v2 mapped on ZCU104 adopts a greater degree of parallelism to fully utilize the on-chip resources since ZCU104 is the largest FPGA device among the mentioned devices. FireFly v2 achieves  $\times 2$  peak power efficiency and  $\times 1.67$  peak throughput than FireFly on the same ZCU104 device.

We then compare FireFly v2 with our previous work on the same four SNN models, as initially reported in FireFly and displayed in Table.VI. FireFly v2 mapped on xczu5ev shows  $\times 1.54$ ,  $\times 1.53$ ,  $\times 1.27$  and  $\times 1.76$  FPS/W improvements on the MNIST, CIFAR10, CIFAR100 and DVS-Gesture classification tasks respectively. While FireFly v2 mapped on xczu3eg may not excel in terms of the FPS/W metric due to the power inefficiencies brought by the complex routings operating under 600MHz, it still exhibits a substantial improvement in inference latency and actual GOP/s performance on the same device compared with FireFly. It’s worth noting that the inference latency of FireFly does not include the direct coding layer, as

TABLE V  
COMPARISON IN HARDWARE SPECIFICATIONS BETWEEN FIREFLY AND FIREFLY v2

	Device	LUTs(K)	DSPs	B/URAM	BUFGs	Freq.	Power	Array Size	PeakGOP/s	GOP/s/DSP	GOP/s/W
FireFly	xczu3eg	15	288	216/0	1(196)	300	3.1 <sup>1</sup>	144 × 16	1382.4	4.8	445.94
	xczu5ev	42	1152	25/50	1(544)	300	18.2	288 × 32	5529.6	4.8	303.82
FireFly v2	xczu3eg	23	256	103/0	3(196)	300/600	6.2	16 × 16 × 4 × 4	<b>2457.6</b>	<b>9.6</b>	396.39
	xczu5ev	26	512	87/8	3(352)	250/500	4.9	16 × 16 × 8 × 4	<b>4096</b>	<b>8</b>	<b>835.92</b>
	xczu7ev	41	1024	160/8	3(544)	250/500	13.5	32 × 16 × 8 × 4	<b>8192</b>	<b>8</b>	<b>606.81</b>

<sup>1</sup> The reported power consumption in FireFly [10] is 2.55W, excluding the power consumption of the PS-side CPUs. To ensure a fair comparison, we use the total power consumption metric of the whole Zynq Ultrascale SoC in this table.

TABLE VI  
COMPARISON WITH FIREFLY ON MULTIPLE SNN MODELS

Benchmark				FireFly(xczu3eg@300MHz)				FireFly v2(xczu3eg@600MHz)				FireFly v2(xczu5ev@500MHz)				
Net	Dataset	FLOP	T	Acc.	us	W	FPS/W	GOP/s	us	W	FPS/W	GOP/s	us	W	FPS/W	GOP/s
SNN5	MNIST	130M	4	98.2	491 <sup>1</sup>	3.1	656.9	1063.3	326	6.2	494.7	1601.6	201	4.9	1015.3	2597.6
SNN7	CIFAR10	284M	4	91.4	1035 <sup>1</sup>	3.1	311.6	1098.2	706	6.2	228.4	1609.9	427	4.9	477.9	2661.9
SNN11	CIFAR100	586M	4	64.3	2128 <sup>1</sup>	3.1	151.5	1101.7	1749	6.2	92.2	1340.5	1057	4.9	193.1	2218.1
SNN9	DVS-G	978M	4	89.3	3546 <sup>1</sup>	3.1	90.9	1103.7	1989	6.2	81.1	1967.6	1281	4.9	159.3	3055.2

<sup>1,2,3,4</sup> The reported inference latency in FireFly [10] do not include the direct coding layer, resulting the higher FPS/W and GOP/s metrics.

TABLE VII  
TOTAL RESOURCES OF DIFFERENT DEVICES

Board	Chip	LUTs	FFs	DSPs	BRAM	URAM
Ultra96	xczu3eg	70K	140K	360	216	0
KV260	xczu5ev	117K	234K	1248	133	64
ZCU104	xczu7ev	230K	460K	1728	312	96

TABLE VIII  
POWER CONSUMPTION VERSUS FREQUENCY (XCZU5EV)

Frequency(MHz)	300	350	400	450	<b>500</b>
Power(W)	4.04	4.28	4.54	4.73	<b>4.93</b>

FireFly does not support non-spike convolution. In contrast, the inference latency of FireFly v2 presented in Table VI is measured end-to-end. The actual improvements of FireFly v2 in these metrics should be even higher.

One might also notice that the actual performance improvement is not directly proportional to the peak performance improvement shown in Table.V. This discrepancy is primarily due to FireFly v2 adopting a coarser parallelism granularity, which can be fully leveraged when processing input feature maps from larger datasets, such as ImageNet. In FireFly, we specifically selected these four models with  $3 \times 3$  convolutional layers and max-pooling layers only, as FireFly is particularly well-suited for optimizing these types of layers. FireFly adopts a fixed convolution configuration and a fully flattened parallelism scheme in the kernel dimension. The spike pixels in the same feature map are processed sequentially in an on-the-fly manner. This allows FireFly to handle small feature maps more effectively. FireFly v2, on the other hand, supports general `torch.nn.Conv2d` operations but operates with a coarser granularity at the pixel level, as it supports pixel-level parallelism and can process  $N$  spike pixels at a time. As a result, we may not fully leverage its advantages when handling small feature maps on FireFly v2, especially when  $N \geq W_o$ . Taking the CIFAR-10 or CIFAR-100 dataset as an example, the size of the feature map is

initially only  $32 \times 32$ . After three  $2 \times 2$  pooling operations, the size becomes  $4 \times 4$ . When dealing with feature maps with a width or height smaller than 4, several inefficiencies become apparent: 1) The explicit same-padding processing time becomes noticeable, as only  $\frac{4 \times 4}{6 \times 6} = \frac{16}{36}$  spike pixels are valid. 2) When  $N > 4$ , the redundant processing elements allocated for pixel parallelism remain idle. 3) Dealing with small feature maps reduces the opportunities for reusing kernel weights within the same set of feature maps, making parameter bandwidth a bottleneck. These inefficiencies won't occur when dealing with large datasets such as ImageNet. Despite the listing inefficiencies, FireFly v2 still achieves improvements in latency and efficiency on the same benchmarks compared with our previous work.

### B. Comparison with the DeepFire2

In FireFly, we've evaluated various systolic-array-based SNN accelerators. We won't repeat these comparisons, as FireFly has already shown superior performance compared to those prior studies [25] [26] [27] [28] [29] [14] [7] [30] [31]. In this paper, we compare FireFly v2 with DeepFire [19] and DeepFire2 [20], two recently published high-performance SNN accelerators also with DSP optimizations and operating at 450-600MHz high clock frequency. DeepFire series targets large multi-die FPGA devices and adopts layer-wise mapping of the entire SNN models. DeepFire2 achieves the highest clock frequency of 600MHz and throughput among all FPGA-based SNN implementations with deep pipelining.

It is important to figure out the experimental setup of DeepFire2 to ensure fair comparison. Despite adopting distinct SNN model mapping schemes (Folded for FireFly, Unrolled for DeepFire), both series of accelerators utilize the same GOP/s metrics. The FLOPS count for the SNN models is determined by calculating the FLOPS count of their equivalent ANN models using established tools like `ptflops`. This same experimental setup enables a fair and meaningful comparison. However, DeepFire2 did not provide information

TABLE IX  
COMPARISON WITH DEEPFIRE2 ON MULTIPLE SNN MODELS

	Network	Dataset	FLOPs	T	Acc.	us	DSP	W	MHz	GOP/s	DSP Eff. <sup>1</sup>	DSP Eff. <sup>2</sup>	GOP/s/W
FireFly v2 (xcvu5ev)	CIFAR-Net <sup>3</sup>	CIFAR10	2.58	4	<b>93.6</b>	2997	512	4.9	500	3443	8	<b>6.73</b>	<b>702.74</b>
	CIFAR-Net <sup>4</sup>	CIFAR100	4.08	4	<b>74.7</b>	4502	512	4.9	500	3625	8	<b>7.08</b>	<b>739.81</b>
	SEW-ResNet34 <sup>5</sup>	ImageNet	7.34	8	<b>67.3</b>	30613	512	4.9	500	1918	8	3.75	391.46
	SEW-ResNet34 <sup>6</sup>	ImageNet	7.34	8	62.4	20276	512	4.9	500	2896	8	<b>5.66</b>	<b>591.03</b>
DeepFire2 (xcvu9p)	SEW-ResNet34 <sup>7</sup>	ImageNet	9.58	8	62.4	24696	512	4.9	500	3103	8	<b>6.06</b>	<b>633.33</b>
	VGG10-S	CIFAR10	0.45	1	87.10	43	2050	20.1	550	10400	8.8	5.10	517.93
	VGG10-L	CIFAR100	1.34	1	65.90	82	2881	29.8	500	15500	8	5.67	519.79
	VGG13-L	ImageNet	15.76	1	40.10	641	5400	47.2	450	21100	7.2	4.55	447.00

<sup>1,2</sup> The first DSP Eff. represents the theoretical peak GOP/s/DSP metric, while the second represents the actual average GOP/s/DSP metric.

<sup>3</sup> 3x32x32-32c3-256c3-256c3-mp2-256c3-256c3-256c3-mp2-512c3-mp2-1024c3-ap-10

<sup>4</sup> 3x32x32-64c3-256c3-256c3-mp2-256c3-512c3-512c3-mp2-512c3-mp2-1024c3-ap-10

<sup>5,6</sup> The spike element-wise function of the first SEW-ResNet34 is ADD, while the second is IAND.

<sup>7</sup> The spike element-wise function is IAND. The input image is resized to  $3 \times 256 \times 256$  to align with the computation granularity of FireFly v2.

about their time step configuration in their experiments, a critical parameter that significantly impacts inference latency. Furthermore, it's important to note that DeepFire2 does not support any form of time step aggregation or sparsity acceleration. Consequently, inference performance relies solely on the following factors: the total FLOPs of the model, simulation time step, clock frequency, and DSP usage, with the simulation time step being the only unknown variable. Based on the metrics reported in their research, it can be inferred that DeepFire2 adopts a simulation time step of one, which explains the exceptionally low reported inference latency. As the computation workload scales linearly with the time step, we quantify the computation workload as the product of FLOPs and the time step (FLOPs·T).

Moreover, in DeepFire series accelerators, SNN models for CIFAR-10, CIFAR-100, and ImageNet classification are meticulously crafted to ensure that the network parameters align seamlessly with the storage granularity of BRAM and URAM. Although the performance of FireFly v2 does not strongly correlate with specific SNN models, we choose SNN models that align with the parallelism granularity of FireFly v2's architecture to ensure a fair comparison.

We have the following several key observations in Table.IX.

1) Both FireFly v2 and DeepFire2 achieve significantly high clock frequencies, exceeding 400MHz. FireFly v2 exhibits stable frequency performance as a standalone engine, while DeepFire2 experiences a sharp frequency drop, dropping to 450MHz when deploying deep SNN models on large datasets.

2) DeepFire2 prioritizes inference latency over benchmark accuracy by adopting a  $T = 1$  SNN setup. In contrast, FireFly v2 targets SNN models capable of delivering high classification accuracy, particularly on more complex datasets such as CIFAR100 and ImageNet. The accuracy of 93.6%, 74.7%, and 67.3% achieved on CIFAR10, CIFAR100, and ImageNet are closely aligned with the state-of-the-art performance in SNN algorithms. The remaining performance gap is primarily attributed to the quantization process, which could potentially be mitigated through the adoption of a quantization-aware training approach in the future.

3) FireFly v2 falls short in achieving the same level of GOP/s performance and inference latency as DeepFire2, since xcvu9p, the FPGA used by DeepFire2, is considerably larger

than the edge devices we use. However, it's noteworthy that FireFly v2 has  $\times 1.32$ ,  $\times 1.25$ ,  $\times 1.33$  average GOP/s/DSP efficiency improvements, and  $\times 1.35$ ,  $\times 1.42$ ,  $\times 1.42$  power efficiency improvements on CIFAR10, CIFAR100 and ImageNet classification tasks compared to DeepFire2.

4) The SNN models benchmarked by FireFly v2 compared to DeepFire2 are not only larger in terms of FLOPs (2.58 vs. 0.45 on the CIFAR10 task and 4.08 vs. 1.34 on the CIFAR100 task) but also more complex(ResNet compared to VGGNet on the ImageNet task).

5) DeepFire2 relies on costly, large FPGA devices that may not be practical for deployment in embedded systems in edge scenarios. On the other hand, the KV260 device we employ is a commercially available and affordable FPGA device. It ensures that the budget for constructing customized edge systems for real-world applications remains manageable.

It is worth noting that FireFly v2 will exhibit a higher performance when the convolutional configuration aligns with its computation granularity. For instance, in the case of SEW-ResNet34 with a  $224 \times 224$  image input, the resulting feature map widths of 14 and 7 do not align with the  $\times 8$  pixel parallelism granularity of FireFly v2. However, when SEW-ResNet34 uses a  $256 \times 256$  image input, there is a notable improvement in efficiency, as the feature map size aligns with the  $\times 8$  granularity. It's also worth mentioning that relying on FLOP calculations of equivalent ANN models may not offer a fair measure of model capacity when running SNN models with multi-bit spikes. This is because the FLOP count of equivalent ANN models does not account for the bit-width of operands, resulting in an efficiency drop when benchmarking the SEW-ResNet34 network with the ADD function.

FireFly v2 and DeepFire2 both utilize similar DSP optimization techniques and operate at similar clock frequencies, resulting in comparable normalized inference efficiency. FireFly v2's higher efficiency compared to DeepFire2 is primarily attributed to its systolic array consistently operating without idle states. However, the remarkably low inference latency of DeepFire2 is primarily achieved through its use of single-time-step inference and extensive utilization of DSP48E2 resources.



### C. Scalability

1) *Scalability in Parallelism Configurations:* Being a highly parameterized design, FireFly v2 demonstrates scalability in parallelism configurations across three FPGA devices of varying sizes. The four distinct parallelism domains facilitate a broad spectrum of configuration options. Larger FPGA devices can support higher levels of parallelism.

2) *Scalability in Supported Model Size:* FireFly v2 represents a classic single computing core design wherein workload computation is temporarily folded. The SNN model size translates to inference time without a physical limitation, thereby exhibiting excellent model size scalability.

3) *Scalability in Supported Algorithm:* While our experiments focus on VGG-like and SEW-ResNet algorithms, the bit-serial decomposition approach adopted by FireFly v2 serves as a versatile solution. This approach can be scaled to accommodate a broad spectrum of multi-bit spikes SNNs.

### D. Discussion

In our experiments, our primary focus is on comparing FireFly v2 accelerators with DeepFire2 since both of its previous versions have already outperformed most SNN FPGA accelerators in terms of latency and efficiency.

The excellent performance of our previous work FireFly is mainly attributed to utilizing the DSP48E2s to build a large synaptic crossbar circuit. The improvements shown in FireFly v2 are attributed to the optimized spatiotemporal dataflow and the doubled clock frequency. However, we believe improving the performance of inference efficiency based on FPGA devices without sparsity acceleration becomes more and more challenging. The clock frequency of FireFly v2 and DeepFire2 is already close to the maximum supported frequency by Ultrascale+ FPGAs.

Another significant aspect of FireFly v2 is its advancement toward a general SNN-DPU solution, akin to Vitis-AI DPU—the ANN-DPU counterpart. The support for non-spike operations in FireFly v2 is crucial for end-to-end deployment without requiring algorithm modifications. This represents a milestone where SNN algorithmic research, such as DIET-SNN [5] and SEW-ResNet [6], can be directly deployed onto FireFly v2 with minimal impact on accuracy. It is worth noting that SNN algorithmic research is still rapidly evolving in terms of encoding schemes [32], neuron types [13] and connection topologies [6]. While the development cycle for hardware used to be significantly longer than that for algorithm software, the current trend in FPGA-based agile hardware development can now expedite the process and provide timely support for the latest algorithmic advancements.

## VIII. CONCLUSIONS

FireFly v2 exhibits significant improvements over our initial version of FireFly. It takes a significant step forward in advancing hardware support for current SNN algorithm developments by supporting non-spike operation, which presents an obstacle in the end-to-end deployment onto existing specialized SNN hardware. The spatiotemporal dataflow enables the processing

of incoming spikes and the generation of output spikes on-the-fly. Additionally, the double data rate technique enables the DSP48E2 systolic array to operate at a clock frequency of 500-600MHz, which is twice as fast as our previous version of FireFly. In this work, our focus remains on targeting commercially available and affordable embedded FPGA edge devices for use in edge scenarios. In the future, we will continue to develop SNN hardware infrastructures that can not only operate at higher speeds but also offer timely support for advancements in SNN algorithms, enabling higher-performance SNN software and hardware co-design.

## REFERENCES

- [1] Wolfgang Maass, "Networks of spiking neurons: the third generation of neural network models," *Neural Networks*, vol. 10, no. 9, pp. 1659–1671, 1997.
- [2] Guobin Shen, Dongcheng Zhao, and Yi Zeng, "Backpropagation with biologically plausible spatiotemporal adjustment for training deep spiking neural networks," *Patterns*, vol. 3, no. 6, p. 100522, 2022.
- [3] Mike Davies, Narayan Srinivasa, Tsung-Han Lin, Gautham Chinya, Yongqiang Cao, Sri Harsha Choday, Georgios Dimou, Prasad Joshi, Nabil Imam, Shweta Jain *et al.*, "Loihi: A neuromorphic manjocore processor with on-chip learning," *IEEE Micro*, vol. 38, no. 1, pp. 82–99, 2018.
- [4] Eustace Painkras, Luis A Plana, Jim Garside, Steve Temple, Francesco Galluppi, Cameron Patterson, David R Lester, Andrew D Brown, and Steve B Furber, "Spinnaker: A 1-w 18-core system-on-chip for massively-parallel neural network simulation," *IEEE Journal of Solid-State Circuits*, vol. 48, no. 8, pp. 1943–1953, 2013.
- [5] Nitin Rathi and Kaushik Roy, "Diet-snn: A low-latency spiking neural network with direct input encoding and leakage and threshold optimization," *IEEE Transactions on Neural Networks and Learning Systems*, 2021.
- [6] Wei Fang, Zhaofei Yu, Yanqi Chen, Tiejun Huang, Timothée Masquelier, and Yonghong Tian, "Deep residual learning in spiking neural networks," *Advances in Neural Information Processing Systems*, vol. 34, pp. 21 056–21 069, 2021.
- [7] Qinyu Chen, Chang Gao, and Yuxiang Fu, "Cerebron: A reconfigurable architecture for spatiotemporal sparse spiking neural networks," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 30, no. 10, pp. 1425–1437, 2022.
- [8] Jeong-Jun Lee and Peng Li, "Reconfigurable dataflow optimization for spatiotemporal spiking neural computation on systolic array accelerators," in *Proceedings of the 38th International Conference on Computer Design (ICCD)*. IEEE, 2020, pp. 57–64.
- [9] Shu-Quan Wang, Lei Wang, Yu Deng, Zhi-Jie Yang, Sha-Sha Guo, Zi-Yang Kang, Yu-Feng Guo, and Wei-Xia Xu, "Sies: A novel implementation of spiking convolutional neural network inference engine on field-programmable gate array," *Journal of Computer Science and Technology*, vol. 35, pp. 475–489, 2020.
- [10] Jindong Li, Guobin Shen, Dongcheng Zhao, Qian Zhang, and Yi Zeng, "Firefly: A high-throughput hardware accelerator for spiking neural networks with efficient dsp and memory optimization," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 2023.
- [11] Larry F Abbott, "Lapicque's introduction of the integrate-and-fire model neuron (1907)," *Brain Research Bulletin*, vol. 50, no. 5-6, pp. 303–304, 1999.
- [12] Peter Dayan, Laurence F Abbott *et al.*, "Theoretical neuroscience: computational and mathematical modeling of neural systems," *Journal of Cognitive Neuroscience*, vol. 15, no. 1, pp. 154–155, 2003.
- [13] Bing Han, Gopalakrishnan Srinivasan, and Kaushik Roy, "Rmp-snn: Residual membrane potential neuron for enabling deeper high-accuracy and low-latency spiking neural network," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2020, pp. 13 558–13 567.
- [14] Wujian Ye, Yuehai Chen, and Yijun Liu, "The implementation and optimization of neuromorphic hardware for supporting spiking neural networks with mlp and cnn topologies," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2022.
- [15] Jian Zhang, Ran Wang, Tengbo Wang, Jia Liu, Shibo Dang, and Guohe Zhang, "A configurable spiking convolution architecture supporting multiple coding schemes on fpga," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 69, no. 12, pp. 5089–5093, 2022.

- [16] Surya Narayanan, Karl Taht, Rajeev Balasubramanian, Edouard Giamcomin, and Pierre-Emmanuel Gaillardon, "Spinalflow: An architecture and dataflow tailored for spiking neural networks," in *Proceedings of the 47th ACM/IEEE Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2020, pp. 349–362.
- [17] Fangxin Liu, Wenbo Zhao, Zongwu Wang, Yongbiao Chen, Tao Yang, Zhezhi He, Xiaokang Yang, and Li Jiang, "Sato: spiking neural network acceleration via temporal-oriented dataflow and architecture," in *Proceedings of the 59th ACM/IEEE Design Automation Conference (DAC)*, 2022, pp. 1105–1110.
- [18] Yao Fu, Ephrem Wu, Ashish Sirasao, Sedny Attia, Kamran Khan, and Ralph Wittig, "Deep learning with int8 optimization on xilinx devices," *White Paper*, 2016.
- [19] Myat Thu Linn Aung, Chuping Qu, Liwei Yang, Tao Luo, Rick Siow Mong Goh, and Weng-Fai Wong, "Deepfire: Acceleration of convolutional spiking neural network on modern field programmable gate arrays," in *Proceedings of the 31st International Conference on Field-Programmable Logic and Applications (FPL)*. IEEE, 2021, pp. 28–32.
- [20] Myat Thu Linn Aung, Daniel Gerlinghoff, Chuping Qu, Liwei Yang, Tian Huang, Rick Siow Mong Goh, Tao Luo, and Weng-Fai Wong, "Deepfire2: A convolutional spiking neural network accelerator on fpgas," *IEEE Transactions on Computers*, no. 99, pp. 1–11, 2023.
- [21] Hardik Sharma, Jongse Park, Naveen Suda, Liangzhen Lai, Benson Chau, Joon Kyung Kim, Vikas Chandra, and Hadi Esmailzadeh, "Bit fusion: Bit-level dynamically composable architecture for accelerating deep neural network," in *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2018, pp. 764–775.
- [22] Chen Zhang, Peng Li, Guangyu Sun, Yijin Guan, Bingjun Xiao, and Jason Cong, "Optimizing fpga-based accelerator design for deep convolutional neural networks," in *Proceedings of the 2015 ACM/SIGDA international symposium on field-programmable gate arrays*, 2015, pp. 161–170.
- [23] Yi Zeng, Dongcheng Zhao, Feifei Zhao, Guobin Shen, Yiting Dong, Enmeng Lu, Qian Zhang, Yinqian Sun, Qian Liang, Yuxuan Zhao, Zhuoya Zhao, Hongjian Fang, Yuwei Wang, Yang Li, Xin Liu, Chengcheng Du, Qingqun Kong, Zizhe Ruan, and Weida Bi, "BrainCog: A spiking neural network based, brain-inspired cognitive intelligence engine for brain-inspired AI and brain simulation," *Patterns*, p. 100789, Jul. 2023. [Online]. Available: <https://doi.org/10.1016/j.patter.2023.100789>
- [24] "Braincog: Brain-inspired cognitive intelligence engine." [Online]. Available: <http://www.brain-cog.network>
- [25] Daniel Neil and Shih-Chii 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.
- [26] Jianhui Han, Zhaolin Li, Weimin Zheng, and Youhui Zhang, "Hardware implementation of spiking neural networks on fpga," *Tsinghua Science and Technology*, vol. 25, no. 4, pp. 479–486, 2020.
- [27] Jilin Zhang, Hui Wu, Jinsong Wei, Shaojun Wei, and Hong Chen, "An asynchronous reconfigurable snn accelerator with event-driven time step update," in *Proceedings of the IEEE Asian Solid-State Circuits Conference (A-SSCC)*. IEEE, 2019, pp. 213–216.
- [28] Xiping Ju, Biao Fang, Rui Yan, Xiaoliang Xu, and Huajin 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.
- [29] Haowen Fang, Zaidao Mei, Amar Shrestha, Ziyi Zhao, Yilan Li, and Qinru Qiu, "Encoding, model, and architecture: Systematic optimization for spiking neural network in fpgas," in *Proceedings of the 39th International Conference on Computer-Aided Design*, 2020, pp. 1–9.
- [30] Daniel Gerlinghoff, Zhehui Wang, Xiaozhe Gu, Rick Siow Mong Goh, and Tao Luo, "E3ne: An end-to-end framework for accelerating spiking neural networks with emerging neural encoding on fpgas," *IEEE Transactions on Parallel and Distributed Systems*, vol. 33, no. 11, pp. 3207–3219, 2021.
- [31] Sathish Panchapakesan, Zhenman Fang, and Jian Li, "Syncnn: Evaluating and accelerating spiking neural networks on fpgas," *ACM Transactions on Reconfigurable Technology and Systems*, vol. 15, no. 4, pp. 1–27, 2022.
- [32] Xuerui Qiu, Rui-Jie Zhu, Yuhong Chou, Zhaorui Wang, Liang-jian Deng, and Guoqi Li, "Gated attention coding for training high-performance and efficient spiking neural networks," *arXiv preprint arXiv:2308.06582*, 2023.



**Jindong Li** received his bachelor degree from Sun Yat-sen University in Guangzhou, Guangdong, China in 2022. He is now a master student in the Brain-inspired Cognitive Intelligence Lab, at the Institute of Automation, Chinese Academy of Sciences, under the supervision of Prof. Qian Zhang and Prof. Yi Zeng. His research focuses on hardware acceleration of brain-inspired algorithms, domain-specific architecture and FPGA system design.



**Guobin Shen** received his bachelor degree from Sun Yat-sen University in Guangzhou, Guangdong, China in 2021. He is now a PhD candidate in the Brain-inspired Cognitive Intelligence Lab, at the Institute of Automation, Chinese Academy of Sciences, under the supervision of Prof. Yi Zeng. His research focuses on biologically-inspired learning algorithms and spiking neural network architecture design and training strategies.



**Dongcheng Zhao** received the bachelor degree from XiDian University, Xi'an, Shaanxi, China, in 2016 and Ph.D degree from University of Chinese Academy of Sciences, Beijing, China, in 2021. He is currently an assistant professor in the Brain-inspired Cognitive Intelligence Lab, Institute of Automation, Chinese Academy of Sciences, China. His current research interests include learning algorithms in spiking neural networks, thalamus-cortex interaction, visual object tracking, etc.



**Qian Zhang** obtained her bachelor degree in 2009 and Ph.D degree in 2014 from Xidian University, China. She is currently an associate professor and director in the Brain-inspired Cognitive Intelligence Lab, Institute of Automation, Chinese Academy of Sciences, China. Her research interests include brain simulation and brain-inspired cognitive computing modeling, especially working memory modeling and simulation of brain rhythms at different levels of consciousness.



**Yi Zeng** obtained his bachelor degree in 2004 and Ph.D degree in 2010 from Beijing University of Technology, China. He is currently a professor and director in the Brain-inspired Cognitive Intelligence Lab, Institute of Automation, Chinese Academy of Sciences, China. He is a principal investigator in the Center for Excellence in Brain Science and Intelligence Technology, Chinese Academy of Sciences, China, and a Professor in the School of Future Technology, and School of Humanities, University of Chinese Academy of Sciences, China. His research interests include brain-inspired artificial intelligence, brain-inspired cognitive robotics, ethics and governance of Artificial Intelligence, etc.