DeepFire: Acceleration of Convolutional Spiking Neural Network on Modern Field Programmable Gate Arrays

Myat Thu Linn Aung[1], Chuping Qu [1], Liwei Yang[1], Tao Luo[1], Rick Siow Mong Goh[1], Weng-Fai Wong[2]
Institute of High Performance Computing, Agency of Science Technology and Research, Singapore[1],
Dept. of Computer Science, National University of Singapore[2]
{aungmtl, qucp, yanglw, luo_tao, gohsm}@ihpc.a-star.edu.sg, wongwf@nus.edu.sg

***Abstract*—Spiking neural networks (SNN) with their 'integrate and fire' (I&F) neurons replace the hardware-intensive multiply-accumulate (MAC) operations in convolutional neural networks (CNN) with accumulate operations — not only making it easy to implement on FPGAs but also opening up the opportunities for energy-efficient hardware acceleration. In this paper, we propose DeepFire — the high-performance RTL IP — for accelerating convolutional SNN inference. The IP exploits various resources available on modern FPGAs, and it outperforms existing SNN implementations by more than $10\times$ in terms of both frame per second (FPS) and performance per watt (FPS/Watt). Our design achieves up to 40.1kFPS and 28.3kFPS on MNIST and CIFAR-10/SVHN datasets with 99.14% and 81.8%/93.1% accuracies respectively. IP was evaluated with 7-series and Ultrascale+ FPGAs from Xilinx achieving Fmax of 375MHz and 500MHz respectively.**

***Index Terms*—Neuromorphic Computing, Integrate and Fire, Acceleration, Hardware, FPGA**

I. Introduction

Spiking neural networks (SNN) have been widely studied in the past decades in different contexts. An interesting hardware proposal uses the resistive-RAM (RRAM) in an analog in-memory computation model [1]. This achieves significant savings in energy consumed by the digital counterparts. Numerous large-scale neuromorphic ASICs were developed including IBM's TrueNorth [2] and Loihi [3] from Intel. These systems demonstrate the power efficiency and speed of SNNs.

On the other hand, the FPGA community has been leveraging the reconfigurability of FPGAs to rapidly prototype their own neuromorphic systems on a small to medium scale. One of the examples is the Bluehive system [4] using the FPGAs cluster. It can handle up to 64k neurons per FPGA, thanks to the large external memory storage. Thomas and Luk [5] presented the simulation of 1k fully connected Izhikevich (IZ) neurons by just using on-chip Block RAMs (BRAM) on the Virtex-5 FPGA. Similarly, in 2017, Pani et. al. [6] has reported the 1.44k fully connected IZ neurons acceleration on the more advanced Virtex-6 FPGA. Since then, the FPGA technology has advanced tremendously and the latest generation of Virtex UltraScale+ VU9P has 47.7MB of on-chip memory [7]. This gives researchers greater flexibility to explore more complex SNN structures with massive data parallelism. Convolutional models are demonstrably more powerful than simple multilayer perception (MLP) models in their ability to extract features. However, they are also computationally much more complex. Recent studies of model conversion from CNN to SNN have shown to improve the SNN classification accuracy [2], [8]. Therefore, not only the SNN use case but also the demand for hardware acceleration has increased [9].

In this paper, we focus on the acceleration of convolutional SNN inference on standard image data sets such as MNIST, CIFAR-10, and SVHN. We propose the *DeepFire* (DF) RTL IP that utilizes all the facilities available on modern FPGAs to improve both throughput and performance per watt. At the same time, we will leverage recent advances in model conversion techniques from CNN to SNN to train our networks.

The main contributions of this paper are as follows.

- We propose the data flow-aware 'integrate and fire' (I&F) neuron core design with the balanced utilization of memories, LUTs, and DSPs.
- We discuss the neuron cores grouping methodology to maximize the RTL timing.
- DF IP supports end-to-end acceleration including pre-processing raw images to spikes.

Our experiment on ultrascale+ FPGAs shows that we are able to achieve 40kFPS with MNIST datasets at the clock speed of 500MHz. Using an Alexnet-like SNN model, we were able to achieve 28.3kFPS running at 425MHz on the CIFAR10 and SVHN data sets. Still, our IPs can run up to 375MHz on older 7-series FPGA and it is the highest performance recorded thus far for SNNs running on FPGA.

The rest of the paper is organized in the following sequence. In Section II, we discuss the brief background of neuromorphic computing and our motivation for this paper. The presentation of our main work begins in Section III, followed by the network design and the hardware implementation in Section IV. We discuss our experiment result in Section V and the conclusion is in Section VI.

II. Background

The goal of neuromorphic computing is to achieve fast real time inference using as little energy as possible like how our brain would do it. In 2016, TrueNorth [2] demonstrated deep

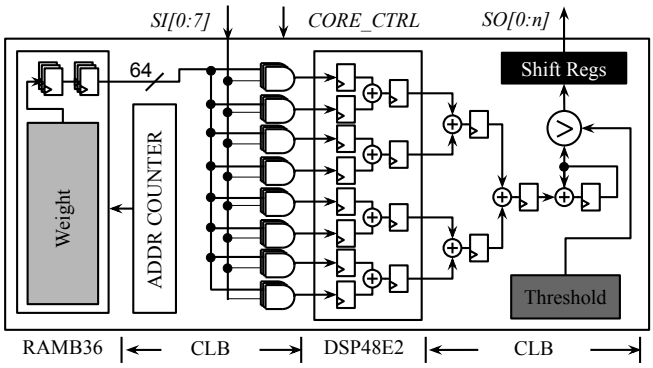

Fig. 1. Architecture of DeepFire neuron core.

convolution SNN inferencing with the capability of classifying 6,000 CIFAR-10 images in just one watt.

Apart from low power neuromorphic ASIC, researchers are also looking into FPGA platforms due to their flexibility to quickly adapt to the emerging neuromorphic research [4], [6], [10]. Some are using them as neuromorphic ASIC emulators [11], [12] to assist ASIC developments and others are building them as pure accelerators. FPGA-based systems are very scalable. By leveraging the massive I/O connectivities in between the FPGAs and external memory, multiple FPGAs can be connected [4], [13]. However, these large systems are not consistent with the energy efficiency goals of neuromorphic systems. Moreover, many of the recent works on the SNN for image classification are still use older generation of FPGAs, and treat them as prototypes rather than tuning them for high-performance. Hence, the potential of FPGAs as SNN hardware accelerators has largely been unexplored [13], [14].

The contribution of our work is mainly on the hardware level optimization and the effective utilization of FPGA on-chip primitives for accelerating inference in convolutional SNN. Our optimization strategies are based on throughput and performance per watt metrics so as to be comparable with recent works [14], [15].

III. DeepFire Architecture

A. Neuron Core Design

Our neuron core design shown in Figure 1 is an improved version from [5], [6]. One of the changes in the design is the use of full data bus-width (64-bits) when reading out the synapse weights from the 36kb BRAM block. With weights being quantized to 8-bit, our core can process 8 weights in one cycle, doubling the throughput compared to [5], [6]. This improvement requires a three-stage adder tree. Unlike [5] where it uses the CLB resources for the adder tree, we proposed a hybrid use of DSP and CLB to balance the hardware resource usage. The first stage of four adders are to be allocated in one DSP and the remaining three adders in the last two stages are implemented in CLB. At the end of add and accumulation for every neuron, the result is compared against the threshold to either shift out ONE for the result higher than the threshold or shift out ZERO for otherwise to the shift registers, *SO*. The 8-bits Spikes-Input, *SI* bus is streamed from the macro-level hierarchy which is discussed in the next section. *CORE_CTRL* orchestrates the neuron add-and-accumulate function while making sure the data flow from the weight memory to the spiking output is fully pipelined. In RTL, the threshold values are instantiated in the bitstream whereas the synapse weights are loaded from the host.

B. Fully Connected Layer

The fully connected (FC) layer is formed by connecting multiple neuron cores together as shown in Figure 2(a). This example shows how the FC layer processes the $3 \times 3 \times 56$ feature map input (504 fan-ins). Since the feature map arrives one column at a time from the previous layer, three-time steps are required to push the entire feature map into FIFO, inferred in LUTRAM. FIFO streams 504 fan-ins on the *SI[7:0]* bus to neuron cores. For the example shown in Figure 2(a), only 8 neurons can fit into one BRAM in the core. Hence, 32 neuron cores are needed in this case to process a total of 256 neurons. Figure 2(a) also shows the memory mapping of this FC layer. The fan-in spikes from 9 pixels (*P*) is to add and accumulate with the first neuron of 32 cores. The accumulated results of the neurons are compared against the respective thresholds (*TH*) and spikes are sent to *SO* registers. This operation will repeat 8 times to process 8 neurons in each core and generates a total of *SO[255:0]* spike outputs.

In this FC layer, *CORE_CTRL* from the controller and the *SI* from FIFO becomes timing-critical due to growing fan-out loading with the more neuron cores. In our DF RTL, the re-timing register insertion is parameterizable as CORE_GROUP to allow users to optimize the IP performance. Our preliminary experiment shows that inserting registers (highlighted grey in Figure 2(a)) every two cores yields the best timing performance. We code the FC layer RTL as a macro — a template that can be re-purposed in the convolution layer.

C. Convolution Layer

In the convolution layer (CNV), each macro performs one sliding window. For the given 3×3 convolution in Figure 2(b), three macros are required to process 5×5 input feature map. Just like in the FC layer example, the FIFO will start streaming spikes to cores at the third time step. At the end of the fifth time step, the output feature map of $3 \times 3 \times 56$ is passed to the next FC layer in Figure 2(b).

Based on the sliding window size of $3 \times 3 \times 56$, each neuron has 504 fan-in. Only the synapse weights of the 8 neurons are stored in each BRAM. With 56 neurons in this layer, a total of 7 neuron cores are required per macro, and 21 cores in total for this layer. Only the top macro which is the master performs the handshake with the previous and next layers. Also, only the master macro has synapse weights that are shared with the other two macros. The fan-out loading of the weight-sharing bus also increases when more macros are needed. Re-timing registers insertion is parameterized as MACRO_GROUP in RTL. We find that the best performance is achieved when both MACRO_GROUP and CORE_GROUP are set to two. The

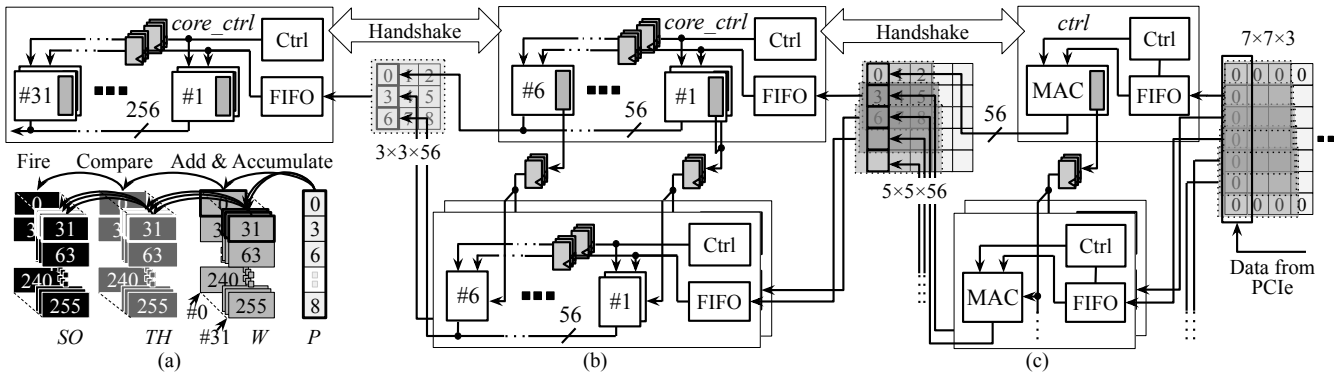

Fig. 2. DeepFire Architecture (a) fully connected layer (CORE_GROUP=2) (b) convolution layer (CORE/MACRO_GROUP=2) (c) transduction layer.

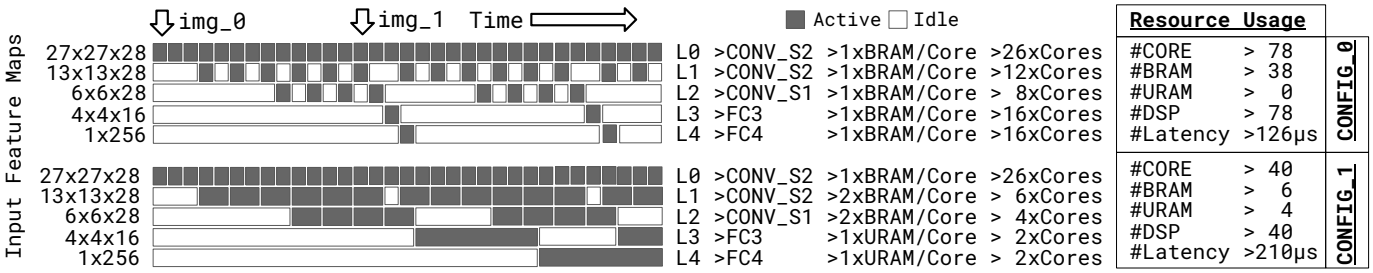

Fig. 3. Data dependency in convolutional SNN pipe-line.

trade-off between Fmax and the hardware overhead for smaller MACRO_GROUP sizes is discussed in Section IV-C.

D. Transduction Layer

The transduction layer in Figure 2(c) is responsible for transforming the raw images it received into spiking signal. It performs the standard convolution on the input image, and activations are binarized with batch-normalization to produce spikes. Since DF is a streaming processor, its communication with the PCIe bridge is the standard AXI-streaming interface.

E. Data Dependency

Data dependency arises when layers have to wait for data from their preceding layers. Figure 3 shows two activity maps of the neuron cores for the same 5-layer network model as an example. The only parameter that differs between the two activity maps is the synapse weight RAM size of the neuron cores. In CONFIG_0 with one BRAM per neuron core assignment, not all the neuron cores are fully active from layers 1 to 4 (L[4:1]) because of the data dependency. Since L1 is using a stride of 2, it has to idle every other time step. To resolve this, we merge two cores into one by instantiating two BRAMs per core (doubling the neurons in each core). Hence, the core count is reduced by half in L1 in CONFIG_1. The cores in L2 can be instantiated with two BRAMs because its stride is one. This RAM scaling based on stride number is only applicable if all filter sizes and neuron counts in the CNV layers are the same. Otherwise, the user has to experiment with the different RAM scaling factor for the best data flow. For the FC layer, we increase the RAM size in the core as long as it does not stall the data flow in the pipeline. In CONFIG_1, the FC layers use $8\times$BRAMs ($1\times$URAM) per neuron core. In RTL, we treat both BRAM and URAM as single-port RAMs with two internal pipeline registers.

The result of this optimization is summarized in Figure 3. The number of the cores and DSPs used is nearly halved in the optimized CONFIG_1 compared to the non-optimized CONFIG_0. However, latency nearly doubled in CONFIG_1. Hence, care must be taken for latency sensitive applications.

IV. Implementation

A. Network Model

The implemented networks only have convolution and fully connected layers without max pooling and padding. The detail of the network structure for different data sets is shown in Table I. For example, CNV-28-3-1 in the table is referring to the CNV layer with 28 neurons with 3×3 filter, one stride and FC-512 implies that the layer is fully connected with 512 neurons. The networks were trained as CNNs using standard back-propagation and batch normalization. However, CNN features such as padding and max-pooling are not the SNN native operations. Here, the max-pooling layers are replaced with the 3×3 kernel convolution and stride of 2.

Our IP accepts the parameters as shown in Figure 4 and infers the necessary blocks required for an Alexnet-like network (SNN<2>). CORE/MACRO_GROUP parameters are for re-timed register insertion to improve the timing closure (mentioned in the Section III-B and III-C) and RAM_SCALING is for resource optimization with data dependency as discussed

TABLE I
IMPLEMENTED NETWORK STRUCTURES

	Dataset <SNN#>		
	MNIST <0>	*MNIST <1>*	*CIFAR-10/SVHN <2>*
Layer-0	CNV-1-3-1	CNV-1-3-1	CNV-28-3-1
Layer-1	FC-512	CNV-64-5-1	CNV-96-3-1
Layer-2	FC-384	CNV-64-3-2	CNV-256-3-2
Layer-3	FC-10	CNV-64-5-1	CNV-384-3-1
Layer-4		CNV-64-3-2	CNV-384-3-2
Layer-5		FC-128	CNV-256-3-1
Layer-6		FC-10	FC-2048
Layer-7			FC-2048
Layer-8			FC-10

```
INPUT_IMG_CHANNEL = 3;
FEATURE_MAP [8:0] = { 1,   1,   3,  5, 11, 13, 27,29,31};
NEURON_CNT  [8:0] = {10,2048,2048,256,384,384,256,96,28};
FILTER_SIZE [8:0] = { 1,   1,   3,  3,  3,  3,  3, 3, 3};
STRIDE_SIZE [8:0] = { 1,   1,   1,  1,  2,  1,  2, 1, 1};
CORE_GROUP  [8:0] = { 1,   2,   2,  2,  2,  2,  2, 2, 2};
MACRO_GROUP [8:0] = { 1,   1,   1,  2,  2,  2,  2, 2, 2};
RAM_SCALING [8:0] = { 1,  16,  16,  4,  4,  2,  2, 1, 2};
CONNECT_S   [8:0] = { 0,   1,   0,  1,  1,  1,  1, 0, 1};
CONNECT_M   [8:0] = { 0,   0,   1,  0,  1,  1,  1, 1, 0};
```

Fig. 4. DeepFire RTL configuration for SNN<2>

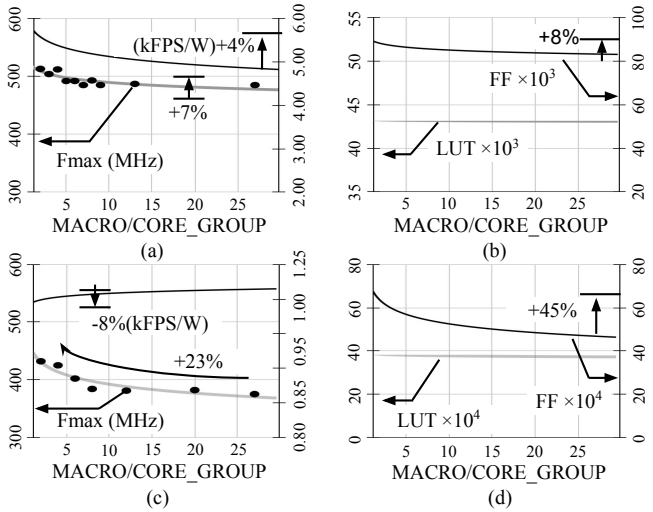

Fig. 5. DeepFire Fmax and the power efficient for (a) SNN< 1 > (c) SNN< 2 >, and resources utilization for (b) SNN< 1 > and (d) SNN< 2 > w.r.t MACRO/ CORE_GROUP settings

in the Section III-E. BRAMs are instantiated in the core when the RAM_SCALING factor is less than or equal to four. Otherwise, URAMs get instantiated. CONNECT_S/M is the configuration of the slave and master interconnect of each layer. If the value is one, that interface is marked to use registers at the SLR crossing site or the Laguna site. With this CONNECT_S/M setting, we assign the resource-heavy layer [3] to SLR0, layer [2, 4, 7, 8] to SLR1, and layer [0, 1, 5, 6] to SLR2. Clock rooting for Laguna site is based on [16].

The other parameters are network-specific. DF supports various network structures except for padding and pooling, as well as various filter sizes as long as the horizontal dimensions and strides are equal to the vertical ones, respectively.

B. Scalability

DF communicates with the host through PCIe DMA-bridge with two AXI-streaming interfaces (the slave AXIS for receiving raw images and synapse weight loading to neuron cores and the master AXIS for delivering the classification result back to the host). Due to the adoption of the standard interfaces, DF is scalable. The handshake protocol between the layers is AXI-streaming compliance and hence, the network can be split easily and deploy across the FPGA cluster.

C. Performance

The Fmax performance of the DF IP on VCU118 is evaluated at various MACRO_GROUP and CORE_GROUP settings with the convolutional SNN<1> and SNN<2> from Table I. Fmax in Figure 5 is determined by the iterative place-and-route that increases the target IP clock until *worst negative slack* (WNS) is almost zero. The dots are the data from the experiment and the solid lines are the fitted lines based on the experiment data. The small SNN<1> network achieves a Fmax of 475MHz without the grouping mechanism and it improves to 513MHz with the group size of 2 (Figure 5 (a)). From grouping of none to 2, the amount of additional flip flops added to the SNN<1> implementation is only 8% (Figure 5 (b)). Despite the hardware overhead, performance/watt improves about 4% for such a small network since the bulk of the power consumption is from the PCIe bridge itself.

For the Alexnet-like model (SNN<2>) trained for the CIFAR-10 and SVHN dataset, Fmax improves as much as 23% from no grouping to a grouping of 2 for cores and macros; topping as high as 432MHz (Figure 5 (c)). At the highest frequency configuration, about 45% of the additional flip-flops are added into the IP as the re-timed registers (Figure 5 (d)). Although throughput (kFPS) and Fmax of the Alexnet improves, the system performance per watt degrades as much as 8% due to the hardware overhead. The performance and the hardware utilization of DF are summarized in Table II. It represents the performance of the respective network when CORE/MACRO_GROUP is equal to 2.

V. DISCUSSION

Table II shows the performance comparison with our DF with other SNN implementations on the same platform. The DF network models are the closed approximation of the counterparts models from [14] [15] and the details are described in Table I. DF is compared against [14] in multi-layer perception (MLP) network at the first row of Table II. [14] claims to improve the throughput by adopting asynchronous design with the novel event-driven time step update. The design contains 16 neuron cores to perform parallel computation on spikes. On the contrary, DF contains 204 neuron cores for spike processing and spike conversion from the image input. Due to the large core count, DF's throughput is significantly more than that of [14] by 128× and yet, DF is 11× more energy-efficient than [14]'s. Given the same platform was used to benchmark,

TABLE II
DEEPFIRE (DF) HARDWARE PERFORMANCE BENCHMARK WITH PRIOR WORKS.

Dataset	*Ref.*	*SNN#*	*Wgt.*	*Platform*	*Prec.*	*Acc%*	*kFPS*	*MHz*	*kFPS/W*	*LUT*	*BRAM*	*URAM*	*DSP*	*$/FPS**
MNIST	DF	$<0>$	1.17M	VC707	8/1	98.1	115.5	375	14.6	44K	217.5	0	204	0.03
	[14]$^{+}$		1.21M	VC707	8/1	98	0.9	-	1.29	-	-	0	0	4.88
	DF	$<1>$	252K	ZCU102	8/1	99.14	40.1	500	5.64	55K	138.5	0	271	0.06
	[15]$^{+}$		269K	ZCU102	8/1	98.94	0.16	150	0.036	125K	264.5	0	0	15.59
CIFAR-10	DF	$<2>$	12M	VCU118	8/1	81.8	28.3	425	0.99	386K	969	385	2963	0.247
SVHN	DF	$<2>$	12M	VCU118	8/1	93.1	28.3	425	0.98	387K	969	385	2963	0.247

($^{+}$) Network does not have transduction layer. (-) The data is not reported (*) The pricing is from the Xilinx official website

DF offers two orders of cost-saving for each inferencing frame and yet, it also provides comparable inference accuracy.

The second row compares DF with recently reported convolutional SNNs with similar arithmetic precision on the MNIST dataset. DF achieves $250\times$ better in the frame throughput and $156\times$ more efficient in throughput per watt than [15]. DF high throughput is mainly contributed by the optimized data pipeline across the entire network. For this network model, DF has a total of 271 cores working in parallel almost without downtime. In contrast to the counterpart [15], there are only 6 neuron cores - one for each layer. Therefore, the first convolution layer has become a performance bottleneck in their design. Thanks to the mixed use of DSP and LUT for neuron core computation, we managed to fit in $45\times$ more cores into the same FPGA platform and yet, DF uses only half of the LUT and BRAM resources compared to [15]. Moreover, the computations in DF's cores are more energy-efficient due to the weight-sharing mechanism across the macros in the convolution layer. Thanks to throughput-driven optimization, DF can clock $2.8\times$ more than the counterpart. In addition, DF produces slightly better classification accuracy. In summary, Table II highlights the versatility of DF in both convolutional and MLP networks, and at the same time, demonstrates the DF's best-in-class performance, energy efficiency, and significant cost saving. The performance results of SNN<2> are shown in Table II without comparison because we could not find any credible comparison for the CIFAR-10/SVHN dataset in the SNN literature.

VI. CONCLUSION

In this paper, we present DeepFire, an RTL IP that accelerates spiking convolutional neural networks on FPGAs. The goal of this IP development is to achieve the highest possible throughput by employing all of the diverse capabilities, including DSP blocks, available on modern FPGAs. To do so, the data flow in the IP is fully pipelined not only in the neuron core but also in the network itself. We also introduce the network floor planning and timing optimization at SLR crossings. As a result, DeepFire can be clocked up to 500MHz, achieving the best-in-class performance/watt among current FPGA SNN implementations. Unlike earlier works, our IP is also flexible enough to support both MLP and convolution networks. DeepFire will enable the efficient implementation of deep learning tasks on FPGAs.

ACKNOWLEDGMENT

This work was supported by the Singapore Government's Research, Innovation and Enterprise 2020 Plan (Advanced Manufacturing and Engineering) under Grant A1687b0033. Correspondence to Tao Luo

REFERENCES

[1] "Brain-inspired computing with resistive switching memory (rram): Devices, synapses and neural networks," *Microelectronic Engineering*, vol. 190, pp. 44 – 53, 2018.
[2] S. K. Esser *et al.*, "Convolutional networks for fast, energy-efficient neuromorphic computing," vol. 113, no. 41, pp. 11 441–11 446, 2016.
[3] M. Davies *et al.*, "Loihi: A neuromorphic manycore processor with on-chip learning," *IEEE Micro*, vol. 38, no. 1, pp. 82–99, 2018.
[4] S. W. Moore *et al.*, "Bluehive - a field-programable custom computing machine for extreme-scale real-time neural network simulation," in *2012 IEEE 20th International Symposium on Field-Programmable Custom Computing Machines*, 2012, pp. 133–140.
[5] D. Thomas *et al.*, "FPGA accelerated simulation of biologically plausible spiking neural networks," in *2009 17th IEEE Symposium on Field Programmable Custom Computing Machines*, 2009, pp. 45–52.
[6] D. Pani *et al.*, "An FPGA platform for real-time simulation of spiking neuronal networks," *Frontiers in Neuroscience*, vol. 11, p. 90, 2017.
[7] Xilinx. Ultrascale architecture. [Online]. Available: https://www.xilinx.com/support/documentation/data_sheets/ds890-ultrascale-overview.pdf
[8] S. Ioffe *et al.*, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," 2015.
[9] M. Pfeiffer *et al.*, "Deep learning with spiking neurons: Opportunities and challenges," *Frontiers in Neuroscience*, vol. 12, p. 774, 2018.
[10] K. Cheung *et al.*, "Neuroflow: A general purpose spiking neural network simulation platform using customizable processors," *Frontiers in Neuroscience*, vol. 9, p. 516, 2016.
[11] T. Luo *et al.*, "An FPGA-based hardware emulator for neuromorphic chip with rram," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, no. 2, pp. 438–450, 2020.
[12] S. Valancius *et al.*, "FPGA based emulation environment for neuromorphic architectures," in *2020 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, 2020, pp. 90–97.
[13] R. M. Wang *et al.*, "An FPGA-based massively parallel neuromorphic cortex simulator," *Frontiers in Neuroscience*, vol. 12, p. 213, 2018.
[14] J. Zhang *et al.*, "An asynchronous reconfigurable snn accelerator with event-driven time step update," in *2019 IEEE Asian Solid-State Circuits Conference (A-SSCC)*, 2019, pp. 213–216.
[15] X. Ju *et al.*, "An FPGA implementation of deep spiking neural networks for low-power and fast classification," *Neural Computation*, vol. 32, no. 1, pp. 182–204, 2020, pMID: 31703174.
[16] C. Lavin *et al.*, "RapidWright: Enabling custom crafted implementations for FPGAs," in *2018 IEEE 26th Int. Symp. on Field-Programmable Custom Computing Machines (FCCM)*, 2018, pp. 133–140.