Neuromorphic LIF Row-by-Row Multiconvolution Processor for FPGA

Ricardo Tapiador-Morales, Alejandro Linares-Barranco, *Senior Member, IEEE*, Angel Jimenez-Fernandez, and Gabriel Jimenez-Moreno

***Abstract*—Deep Learning algorithms have become state-of-the-art methods for multiple fields, including computer vision, speech recognition, natural language processing, and audio recognition, among others. In image vision, convolutional neural networks (CNN) stand out. This kind of network is expensive in terms of computational resources due to the large number of operations required to process a frame. In recent years, several frame-based chip solutions to deploy CNN for real time have been developed. Despite the good results in power and accuracy given by these solutions, the number of operations is still high, due the complexity of the current network models. However, it is possible to reduce the number of operations using different computer vision techniques other than frame-based, e.g., neuromorphic event-based techniques. There exist several neuromorphic vision sensors whose pixels detect changes in luminosity. Inspired in the leaky integrate-and-fire (LIF) neuron, we propose in this manuscript an event-based field-programmable gate array (FPGA) multiconvolution system. Its main novelty is the combination of a memory arbiter for efficient memory access to allow row-by-row kernel processing. This system is able to convolve 64 filters across multiple kernel sizes, from 1 $\times$ 1 to 7 $\times$ 7, with latencies of 1.3 μs and 9.01 μs, respectively, generating a continuous flow of output events. The proposed architecture will easily fit spike-based CNNs.**

***Index Terms*—Address-event-representation, artificial intelligence, computer vision, convolutional neural networks, deep learning, DVS, FPGA, neuromorphic engineering.**

I. Introduction

CONVOLUTIONAL Neural Networks (CNNs) have been demonstrated to be one of the most powerful approaches for solving machine vision tasks. The relatively simple supervised training and their efficiency extracting features from a scene, make this kind of network useful in different fields, such as medicine [1], [2] or robotics [3]. CNNs are usually trained using back propagation algorithms, which keep the CNNs output matched to a label for a given input image from a dataset. The training of the network is usually performed by hardware accelerators, such as Graphical Processor Units (GPUs) or high-end servers.

CNN models, such as VGG19 [4] or Alexnet [5], are computationally expensive because frame-based convolutions require billions of multiplication and accumulation operations per image. Despite the fact that many smart phones include small GPUs, deploying a CNN on it requires more power consumption and makes it not ideal due to the limited battery capacity of these devices. There exist many ASIC implementations of CNN accelerators that reduce the power consumption applying different techniques to decrease the number of operations (i.e., discarding the multiplication of those pixels whose value is null) [6] and pixel precision without losing accuracy [7]. Although these accelerators achieve good results running computationally expensive network models, as previously cited, the number of operations is still high. However, there are other techniques, such as neuromorphic event-based techniques, that can decrease the number of operations and power consumption.

The concept of neuromorphic engineering is based on the analogy between the behavior of transistors, which is biased in the subthreshold region, and the physics of biological neurons [8]. This approach opened a new processing paradigm, which takes inspiration from the structure and functioning of the human brain, because the information is encoded in spikes (also called events) that are processed in parallel by massive layers of neurons interconnected via synapses [9].

Based on this kind of processing, several event-based sensors have been developed, such as the Dynamic Vision Sensor (DVS) [10]–[12] or artificial cochleas [13]–[15]. DVS represents a scene in a visual way, where each pixel is a neuron that generates a spike stream depending on its luminosity changes. A frame-based camera records all the pixel values of the scene even if parts of it has not changed. However, a flow of events of the DVS represents only the moving reality; it does not need to load all the static pixels of an image. This property reduces the total number of pixels to be processed. Furthermore, there is no sample period in these devices. As soon as a pixel changes, an event is produced and sent out from the sensor. Event-based processing is, therefore, asynchronous and continuous [16].

Events from neuromorphic vision sensors can be used to solve machine vision tasks, such as object tracking or pattern recognition. Regarding pattern recognition, there are many techniques to extract patterns, such as HOTS [17], which does

This work was supported by the excellence project from the Spanish government grant (with support from the European Regional Development Fund) COFNET (TEC2016-77785-P). The work of R. Tapiador-Morales was supported by a "Formación de Personal Investigador" Scholarship from the University of Seville. This paper was rec-ommended by Associate Editor P. Hafliger. *(Corresponding author: Ricardo Tapiador-Morales.)*

The authors are with the Robotics and Computer Technology Lab, University of Seville, Seville 41012, Spain (e-mail: ricardo@atc.us.es; alinares@atc.us.es; ajimenez@atc.us.es; gaji@atc.us.es).

not implement any type of spiking neural network. Techniques of this kind are implemented in software or deployed in existing neuromorphic platforms, such as Spinnaker [18] or BrainScales [19]. Spiking convolutional neural networks (**SCNN**) replicate the same concept as CNNs in the spiking domain. SCNNs are highly accurate and provide high efficiency in terms of power and speed, compared to conventional frame-based CNNs [20], [21]. These spike- or event-based techniques hold the main advantage of pseudo-simultaneity [20] , which allows to start the processing with the arrival of the first event coming from the sensor, and therefore, obtaining an output while the sensor is still producing events. For SCNN it implies a different concept from frame-based CNN. The main disadvantage of SCNNs is the RAM memory: a CNN only needs RAM for computing one layer, while a SCNN requires to store the membrane potential state of the whole network to update it properly when an event arrives. In the same way as chip accelerators in frame-based domains, many solutions have been developed for FPGA and ASICs [22]–[24]. A common problem in FPGA convolution accelerators is the memory access for kernel weights; e.g., for a 7×7 kernel, 49 memory accesses with incrementing latency and bottlenecks are usually required. However, it can be improved by changing the way in which data are accessed. Row by row has been implemented on chips before [25], [26], but it was not successfully implemented in old FPGAs since the memory resources of these devices were smaller and slower (i.e., 18 Kb on Spartan 6 vs 36 Kb on 7-families, they not support of direct cascade of BRAM or absence of integrated FIFOs for Spartan-6 [27]), allowing to implement in an efficient way several independent convolution engines with pixel-by-pixel processing. In this paper, an event-based multi-convolution engine is presented. This engine is based on the leaky integrate-and-fire neuron model (**LIF**). It applies kernel values over neurons, storing the membrane potentials in embedded memory, and firing events when neurons reach threshold values. Properties of the LIF neuron, such as refractory period and leakage, have been implemented. The main novelty of this design is that memory is read/written row by row, reducing the latency to process an input event. The system was implemented in FPGA (Xilinx Zynq) and tested with the application of different convolution kernels, visualizing and studying the output.

The paper is organized as follows: Section II describes the Spiking Convolution, while Section III explains how the engine works. Section IV presents the test scenario that was used to verify the system functionality. And finally, the experimental results and conclusions are presented in section V and VI, respectively.

II. Spike-Based Convolutions

CNNs models are commonly formed by three layers: (1) a convolution layer, where images are convolved; (2) a pooling layer, where an image is sub-sampled to reduce its size in order to decrease computation in future layers, and (3) a non-linearity layer, e.g., RELU [28].

In frame-based image processing, the convolution operation consists in applying a kernel matrix to a pixel value, multiplying kernel values with corresponding pixels and adding these results together as a single value. The convolution is performed by sliding the kernel over all pixels of the image, commonly starting at the top left corner. In convolution, the zero padding technique is commonly used to process boundary pixels; other convolution implementations simplify the computation by skipping the boundaries. Mathematically, it is defined as shown in Equation 1, where K is an NxM kernel matrix, X is the input image and Y is the convolved image.

$$\forall_{i,j} \rightarrow Y(i,j) = \sum_{a=-\frac{N}{2}}^{\frac{N}{2}} \sum_{b=-\frac{M}{2}}^{\frac{M}{2}} K(a,b) \cdot X(a+i, b+j) \tag{1}$$

Y(i, j) is defined by the corresponding input pixel X(i,j) and the weighted adjacent pixels, scaled by K values [29].

However, in an event-based processing system, not all pixels are always processed because neuromorphic sensors get luminosity changes of the actual scene, and these changes are transmitted as events. An event is represented with an (x, y, p) tuple that corresponds to the pixel address (x, y) of an image that is changing, and a polarity bit (p) that indicates if the pixel is ON or OFF. ON polarity means that the luminosity detected by the pixel in the present state is higher than the luminosity detected in the previous state, while OFF polarity means the opposite: the luminosity detected in the present state is lower than the luminosity detected in the previous state [30]. In a spiking convolution an input image X is coded in such a way that each pixel X(i, j) is represented by a number of events of a visual source output (DVS retina). The results of convolution operations are stored in a Y matrix (capacitors for analog circuits or registers or RAM cells for digital circuits). When an input event arrives, the corresponding pixel and its neighbors are modified in Y, adding the convolution kernel. The following Equation (2) shows the operation for computing each incoming event with address (i, j):

$$Y(i+a, j+b) = Y(i+a, j+b) + K(a,b), \forall a, b$$
$$a \epsilon \left[-\frac{N}{2}, \frac{N}{2}\right], b \epsilon \left[-\frac{M}{2}, \frac{M}{2}\right], N, M = dim(K) \tag{2}$$

Once all the events of pixel X(i,j) have been received and calculated, the integrator value of the corresponding address Y (i, j) accumulates X(i + a, j + b) $\forall(a, b)$, times the value of the kernel, obeying equation (1). In other words we are adding the kernel value to a neighbor of outputs as many times as number of input events. This continuous addition is equivalent to multipling the intensity of a pixel by a kernel value in frame-based convolutions. The output of the convolution operation, at this point, is stored in a matrix of integrators Y. The resulting matrix Y can be sent out in several ways. In this paper, it is inspired by LIF neuron model [31]. The continuous sum of kernel values over a neuron increases or decreases its membrane potential depending on positive or negative coefficients, respectively. When the membrane potential of a neuron reaches a positive threshold (**PTH**), a spike is generated with positive polarity and (x, y) address, resetting its membrane potential, as is shown in Fig. 1.

A biological neuron decreases its membrane potential through leakage when it does not receive any excitation. We have mimicked this property of biological neurons, because a LIF neuron

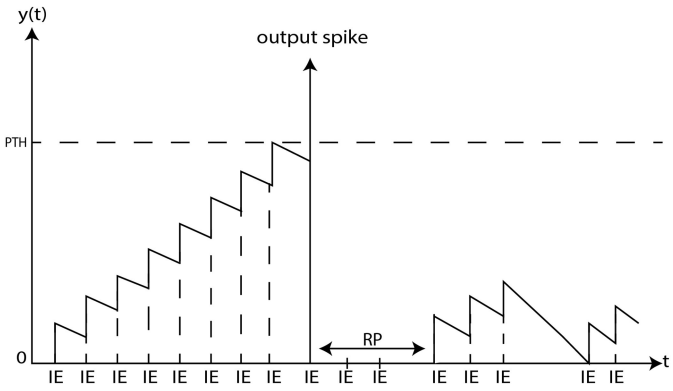

Fig. 1. Changes on the membrane potential of a neuron along time with input events (IE).

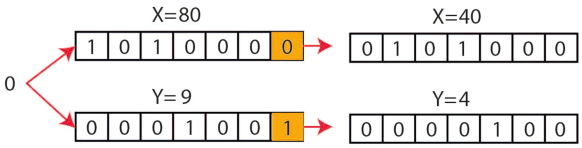

Fig. 2. Example of event sub-sampling. The input event is shifted to the right, dividing its value by 2.

that has not received any excitation means that it is not giving information about the scene. The leakage decay time of a neuron and the value that the neuron decreases its potential are parameters that allow controlling the output event rate. This implies that, with a small decay time, neurons would be resetting more often. On the other hand, a longer decay time makes neurons reset their potential rarely, increasing the number of output spikes.

Although leakage decreases the event rate at the output, the event rate would be higher when implementing SCNN, since one layer comprises several convolutions in parallel. A solution to stabilize the output rate is implementing the refractory period (**RP**) of the LIF neuron. Thus, if a neuron fires and generates a spike, it should wait for a period of time before receiving any kind of excitation.

In CNNs, a convolution layer is frequently followed by a pooling layer [32]. In frame-based CNNs, there exist multiple types of pooling; e.g., max-pooling applies max filter to the input image, filtering pixels with the maximum value. In SCNN, subsampling usually consists in dividing by 2 the x, y address of the output event of a convolution stage, reducing the image size (address space) [33]. In hardware, this step is easy to implement with a shift operation to the x,y address one position to the right, as is shown in Fig. 2.

III. Engine Architecture

The design developed is a fully programmable digital convolution system inspired in how the LIF neuron works. The architecture has three interfaces: two address-event representation interfaces (**AER**) for the asynchronous handshaking protocol of four steps, used to send and receive signals between neuromorphic systems. And a 32-bit digital interface used by a host micro controller to configure the accelerator. The system is able to compute a maximum number of 64 convolution operations in parallel with kernel sizes from 1×1 to 7×7. These convolutions are computed row-by-row in a fully shuffled way, where one complete row is computed per clock cycle. The engine is able to perform the pooling operation, decreasing the spatial size of the representation to reduce computation in a SCNN.

The main novelty of the design presented in this paper is that data from memory is accessed row by row, reducing the latency to convolve a square kernel. We first give a high-level overview to processing pipeline. When an input event arrives, each convolution engine reads the membrane potential, refractory timestamps and leakage timestamps from memory. Those values correspond to neurons around the address of the received event and to the size of the programmed kernel, and they are read row by row. The convolution engine compares timestamps with two global counters, one for the refractory period and the other one for leakage, in order to verify whether the refractory period has been met and if leakage must be applied. Then, the convolution engine convolves a membrane potential row with a kernel row. Those neurons that can fire and reach their threshold produce an event with x, y address. This process is repeated until all rows are convolved by all kernel rows.

The following subsections describe functional blocks of the accelerator and the processing pipeline in detail.

A. Membrane Potential, Timestamps and Kernel Memories

FPGAs are composed of different kinds of resources: lookup tables (**LUT**), flip flops (**FF**) and Block RAM memory (**BRAM**). BRAM is the main memory resource; it is a set of dual-port RAM modules instantiated into the FPGA fabric to provide on-chip storage for a relatively large set of data. The available amount of these memories is device specific. In this design, BRAM is divided into four different memory banks: membrane potential (**MP**), leakage timestamps (**LT**), refractory timestamps (**RT**) and kernel values (**KV**).

Memory banks for MP, LT and RT are organized in multiple blocks that store values in rows of 8 pixels. When a convolution engine accesses memory data, it transmits three x,y addresses of the corresponding row to be convolved and the convolution ID (**CID**). X address selects memory blocks to be accessed, using a decoder to enable or disable the banks.

Memories store values for a maximum image size of 128×128 for all convolution engines. Since the memory banks are shared by all the convolution engines, the CID specifies the memory region for the corresponding engine and the y address selects which pixel row is read/written.

Each BRAM memory row stores 8 pixels with a resolution of 8 bits for MP and 7 bits for timestamps; thus, 128 pixel row, 16 BRAMs blocks are needed. The depth of this memory is related to the number of convolution engines (**N**) multiplied by the number of rows of an image, which is 128.

Fig. 3 shows an example of how MP, RT and LT are read/written. During an event processing, the convolution engine always reads one row of two consecutive BRAMs, the one where the input event belongs and the neighbor row. The reason

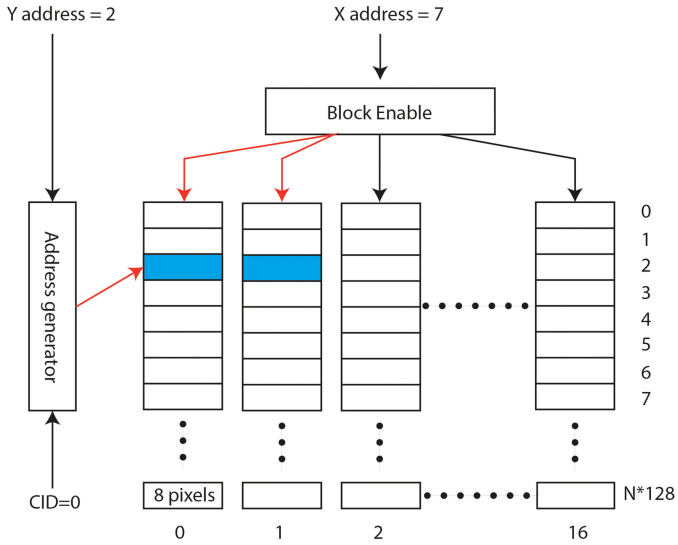

Fig. 3. Example of BRAM access for an input event with x, y address 7, 2, respectively. Pixel with x address 7 is in block 0; this block and its neighbor block are enabled for read/write. Y address and CID generate the memory address, which in that case is the address 2 of convolution engine 0.

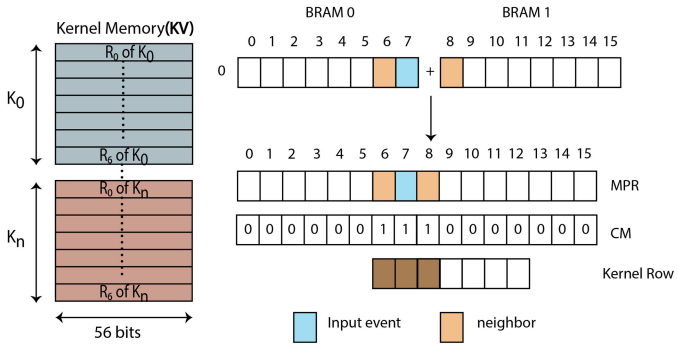

Fig. 4. Kernel memory structure and row generation for convolution operation.

for this multiple read is that neighbor pixels of the input event can be involved in the convolution operation, but they can be stored in a different bank.

Following the same concept, kernel memory is organized in one shared block, where KVs are stored row by row. When a kernel row is read, a mask (**CM**) is generated in order to let the convolution engines know which pixels of the combined membrane potential row (**MPR**) will be convolved. Fig. 4 shows the kernel memory structure and an example of how the kernel and the data are organized for the convolution operation.

B. *Leakage System*

The leakage of a neuron is applied during convolution decreasing the neuron value by a configurable decay value (**DV**). Leakage was implemented using a 32-bit counter. Since storing the content of this counter for each neuron would need a large number of resources, only 7 bits from them are stored in BRAM. These 7 bits are selected using a sliding window that configures the timing resolution as a function of the input event rate. During a convolution operation, when an input row arrives to be convolved, all the LTs of the neurons involved are compared with the current counter value of the leakage counter (**LC**) and then they are updated with the current value of LC. When the difference between LT and LC is higher than a configurable leakage period, DV is applied during convolution. However, there is one issue, even if the window of LC is properly configured for the input event rate, it will overflow, applying decay wrongly. In order to avoid that problem, a solution based on distributed RAM memory was implemented.

Multiple LUTs in a Slice Memory (SLICEM) can be combined in diverse ways to store larger amounts of data. LUTRAM, or Distributed RAM, is crucial to many high-performance applications that require relatively small embedded RAM blocks, such as FIFOs or small register banks.

The solution proposed in this paper consists of a leakage memory composed of an array of 128 $\times$ 128 LUTRAM cells of 2 bits each, one for each convolution engine. Fig. 5.1 shows how leakage LUTRAM memory bank (**LLM**) works for a 3 $\times$ 3 kernel size. Initially ($t = t_0$), **LLM** cells have a value of 0 (Fig. 5.1.a). When LC produces an overflow ($\mathbf{L_{ov}}$), each cell adds 1 to its content (Fig. 5.1.b). When an input event (IE) is convolved, the content of those rows that have been accessed during the convolution, are reset (Fig. 5.1.c). If another overflow occurs, the cells increment their values again; those cells that have reached a value of 2 indicate that long time has passed since the last access to that neuron. Therefore, those neurons would have decreased their membrane potential to 0; thus, the next time one of those neurons is accessed, their MP is reset (Fig. 5.1.d).

C. *Refractory Period System*

The refractory period is a property of biological neurons that guarantees a separation time between two spikes generated by the same neuron. The refractory period uses another counter with the same sliding window mechanism as the leakage counter. During the convolution process, RT is compared with the refractory counter time (**RCT**). If RT is higher than RCT, the refractory period is met and the neuron can fire if the threshold is reached. Otherwise, this neuron must wait until the period is met. If during convolution, a neuron fires, then it updates its RT by the result of the RCT plus a programmable refractory period value (**RPV**). The resulting timestamp indicates the next time that neuron will be able to fire. Otherwise, its RT is 0, thus the neuron can fire next time it will be accessed.

However, during this update, the sum operation of RCT and RPV may produce an overflow. Therefore, this neuron would be able to fire before meeting the refractory period. In that case, the solution proposed uses another LUTRAM bank as the leakage system. The refractory LUTRAM memory bank **RLM** is formed by 128 $\times$ 128 LUTRAM cells of 1 bit, When adding the actual time to the refractory period, an overflow occurs, which means that the neuron must wait one counter overflow and some time before it will be able to fire. Therefore, when the refractory counter (**RC**) produces an overflow, instead of adding 1, as for leakage memories, it subtracts 1 to all cells in the **RLM** indicating that an overflow has occurred. Fig. 5.2 shows how the

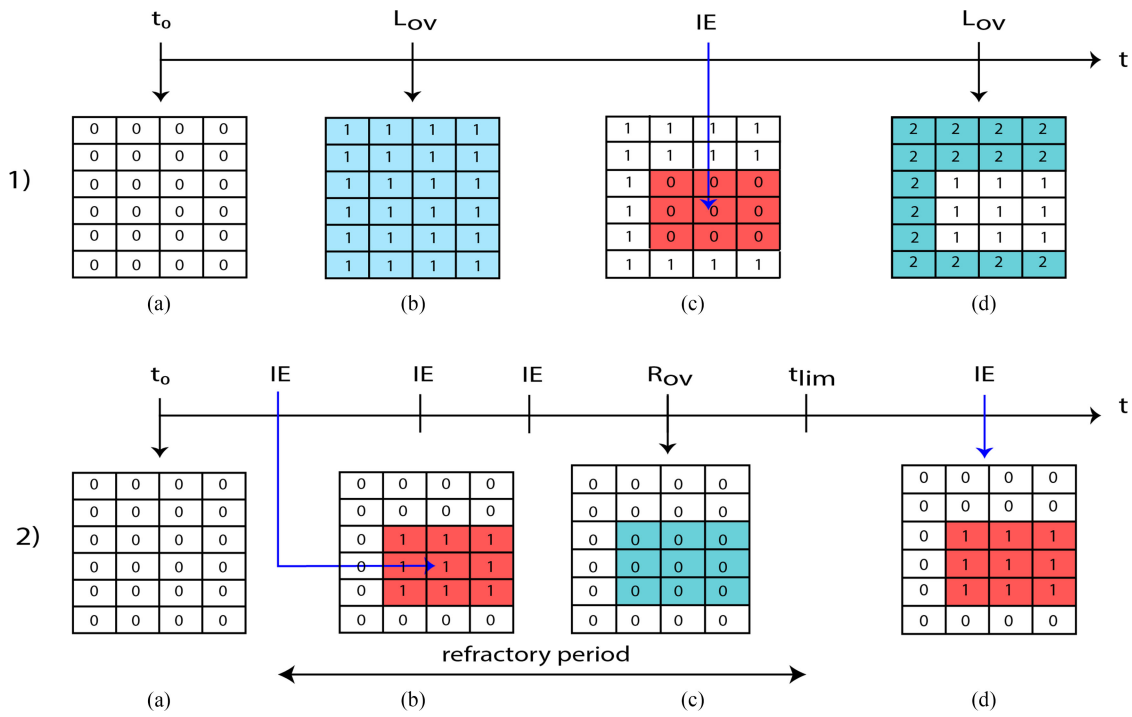

Fig. 5. Leakage and Refractory period memories system.

refractory memory system works. In this example, we suppose a 3×3 kernel, where each coefficient is higher than the threshold. Therefore, after applying the kernel, 9 events would fire, and the sum operation of RCT and RPV always produces an overflow. When $t = t_0$ (Fig. 5.2.a), the refractory memories are empty, since no event has been fired yet. When an input event (**IE**) arrives and it is convolved, the sum of RCT and RPV for each neuron produces an overflow, updating the content of the refractory memories (Fig. 5.2.b). Although many IE have arrived, the neurons cannot fire, since they have to wait for an overflow from RC and some time (t_{lim}) to meet the refractory period. When an overflow occurs, the cells are updated subtracting 1 from their content and indicating that an overflow ($t_{R_{ov}}$) has occurred (Fig. 5.2.c). In spite of the memories update, the neurons have to wait some time until the refractory period is met ($t = t_{lim}$). When the refractory period is met, neurons can fire as is shown in Fig. 5.2.d.

The leakage and refractory memory banks are inside each convolution engine and they are read/written row by row. When the leakage counter produces an overflow, the convolution engine cannot update the membrane potential memory until the content of each leakage memory has been updated, since the membrane potential could be reset. However, a convolution engine can read from memory during the update operation, reducing the waiting time.

D. Convolution Engine

The convolution engine module consists of a state machine, which communicates with the memory, calculating the address to access the membrane potential, timestamps and kernel rows. The convolution engine requests access to a memory arbiter, which gives access for reading or writing data. The main novelty of this engine is that the membrane potential and kernel values are read and processed row by row until the entire kernel is applied, considerably reducing the number of memory accesses.

Although the convolution operation can be performed in one clock cycle, it would need many resources, since several operations are required, such as check leakages and refractory times.

In order to reduce the resources, the convolution operation is divided into two phases for each row: Masks step and Convolution step.

In previous sections, it was explained how the refractory period and leakage work. In the Masks step, timestamps are verified to either apply decay or determine whether the refractory period has been met.

Inspired by how SIMD processors work over arrays of data [34], [35], the **Mask step** generates two binary masks, one for leakage and the other for the refractory period. The leakage mask (**LM**) indicates to each neuron if decay must be applied during convolution (logical 1) or not (logical 0). On the other hand, the refractory mask (**RM**) indicates if the refractory period of a neuron has been met or not.

The **Convolution step** updates the timestamps, leakage memories and refractory period memories. Before convolution, the corresponding row values of the leakage memory are checked, as was previously mentioned. If the leakage value is 2, the membrane potential is reset, otherwise the convolution operation is performed for that neuron. During the convolution operation, the leakage mask (**LM**) is multiplied by DV in order to apply leakage to the corresponding neurons. Therefore, the convolution operation consists in adding MP to KV and subtracting DV,

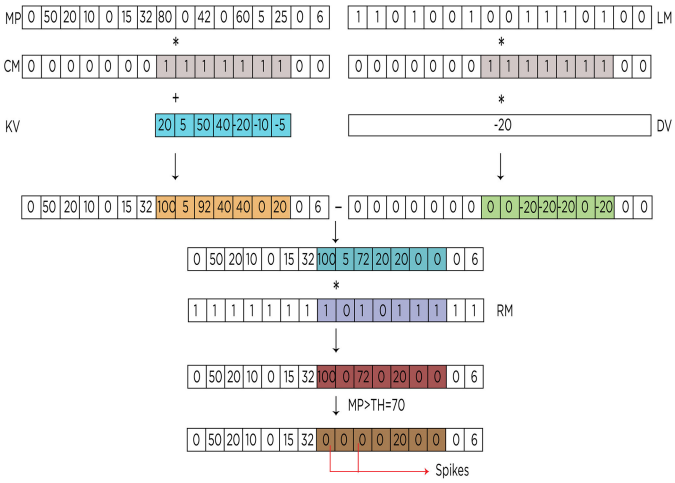

Fig. 6. Kernel memory structure and row generation for convolution operation.

if it must be applied. However, the LIF neuron cannot fire if the refractory period is not met; thus, the convolution operation result is multiplied by the refractory period mask. The Refractory period mask is a binary mask, which implies that, if the refractory period is not met, the result of the convolution operation is 0, since the neuron cannot receive any kind of excitation. Fig. 6 shows how the convolution operation is performed with the Leakage mask and the Refractory mask.

In the convolution operation, when applying a kernel value to a neuron, its membrane potential may reach the threshold. In that case, this neuron must fire, resetting its membrane potential value and storing its address to an output FIFO. The advantage of this implementation is that it decreases the number of memory accesses, since the data are read and convolved row by row. Fig. 7 describes the convolution process. This example highlights that MP values are kept positive and below the threshold.

E. Hardware Implementation

The design was described as RTL with System Verilog language and synthesized for a Zynq-7100 MMP platform using Vivado 2016.4. This platform contains a PSoC with a Dual ARM Cortex-A9 MPCore, called processing system (**PS**), and a Kintex-7 FPGA, called programmable logic (**PL**), with 444 K logic cells in the same chip.

In this implementation, Zynq is running an embedded operating system (OS) called Petalinux on the PS. The OS allows developers to configure the system easily. This configuration consists of a C++ program that reads a text file with the parameters values and transmits them to PL. PS and PL communicate through AXI bus.[1] When the system is configured, it receives and sends events using AER interfaces.

The interfaces of the accelerator are divided into two different buses: an AXI slave bus [36], which configures the parameters

[1]AXI stands for Advanced eXtensible Interface. It is an ARM-standard interface bus included in the Advanced Microcontroller Bus Architecture (AMBA) open-standard third generation. It is used in Zynq for PS and PL high performance communication interface.

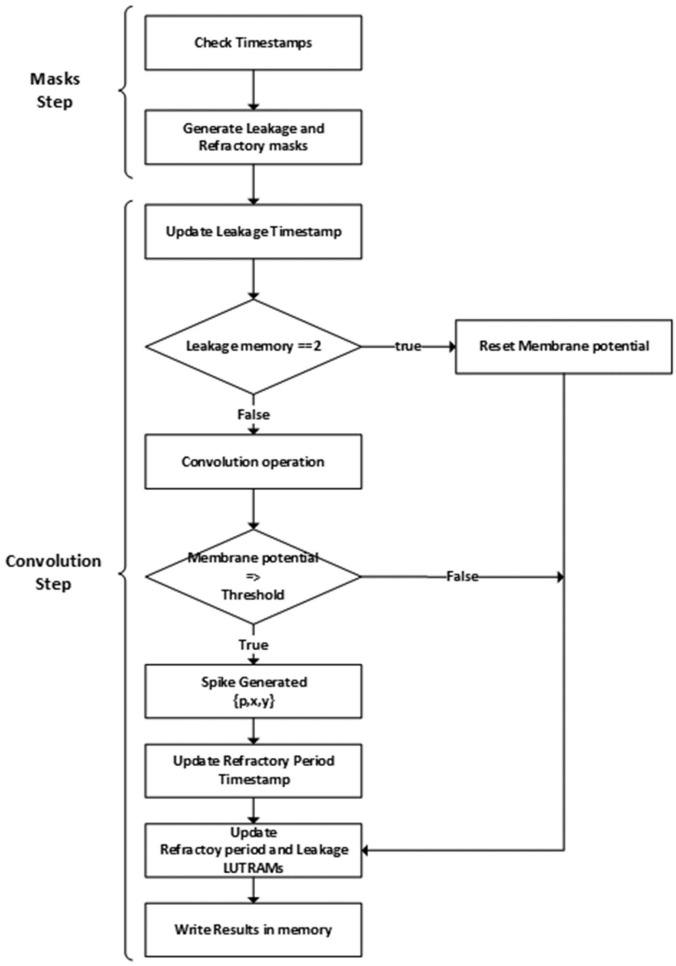

Fig. 7. Convolution phases diagram.

of the system (leakage time, decay, threshold and kernel values), and two AER buses [37]–[40], which receive events from an event-based sensor and send output events. An architecture diagram is shown in Fig. 8. In this work, a custom board called SoC-Dock, which adapts Zynq-7100 interfaces to AER, was designed and developed. The FPGA has a consumption of 1.608 W, measured using Xilinx Power Analysis tools of Vivado after the implementation step, and a clock frequency of 100 MHz for 64 convolution engines. The resources used for 64 convolution engines are shown in Table I.

IV. Experimental Results

The experimental setup used in this paper is shown in Fig. 9. It consists of an AER board that receives events from the computer through USB and sends them using AER interface to the FPGA, where the events are processed. Output events from the FPGA are collected by the USBAERmini2 board and visualized by JAER software.

The experiment proposed in this paper consists in processing multiple images from POKER-DVS datasets in order to measure latencies and the response of the filter against different stimuli.

For the initial test, the FPGA is configured to use the 64 convolution engines with different kernel sizes in order to measure

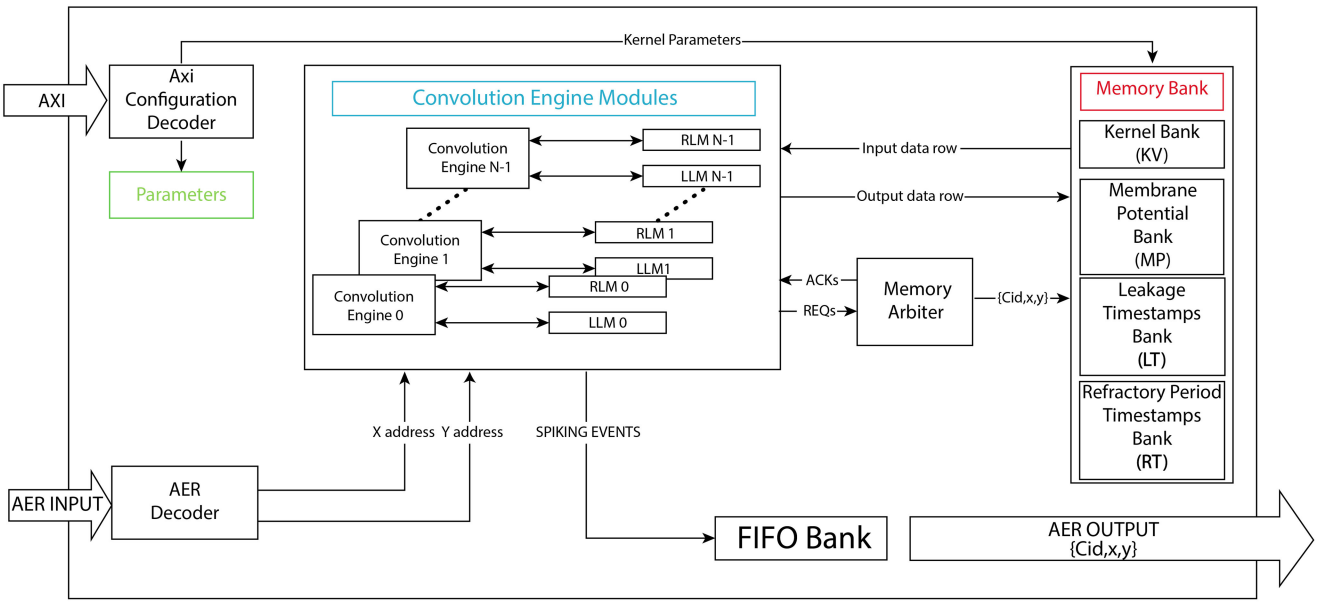

Fig. 8. Block diagram of the system architecture.

TABLE I
FPGA RESOURCES

Resource	Utilization	Available	Utilization %
LUT	247472	277400	89,21
LUTRAM	54983	108200	50,82
FF	172607	554800	31,11
BRAM	710	755	94,04
IO	37	362	10,22

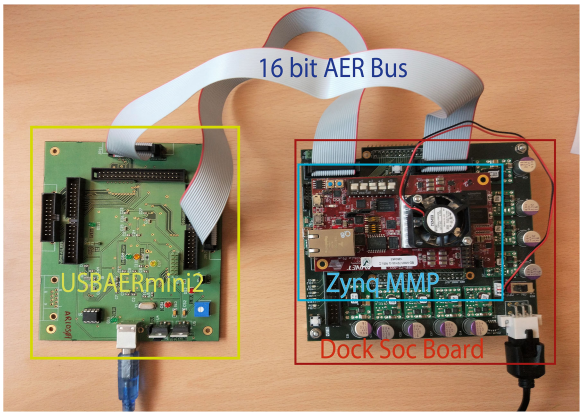

Fig. 9. Experimental Setup.

the latency from minimum kernel size 1×1 to maximum size 7×7. As was mentioned above, there exists one case that stops the system to refresh leakage memory. Updating a row memory takes 2 clock cycles (one to read and another one to write). Since the leakage memory has 128 rows and the clock period is 10 ns, the delay is 2.56 μs. Although this situation occurs sporadically, the update should coincide with the convolution operation step. In order to obtain a more precise processing time of the system, the average time to process an event has been calculated after processing 10000 events from each image with different kernel sizes.

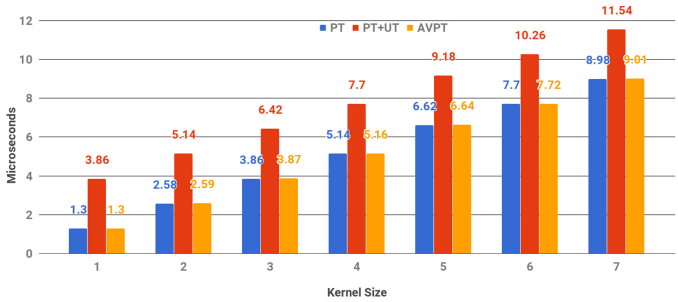

Fig. 10. Processing time of the different kernel size with 64 convolution engines.

Fig. 10 shows the resulting processing time obtained from the test. The blue bars (**PT**) represent the processing time obtained with different kernel sizes without any leakage memory delay, which is the best case. The red bars (**PT** + **UT**) show the worst case, which is the result of processing time plus leakage memory delay. The orange bars (**AVPT**) are the average time obtained after processing 10000 events from different poker images of Slow-Poker-DVS dataset [41]. The differences obtained between the best case and the average time are insignificant. This demonstrates that the worst case does not have an impact on the processing time and the behavior of the system has a tendency towards the best case.

The results show that the worst case would be a leakage overflow when processing the maximum kernel size (7×7). Comparing the time obtained with the latency of the neuromorphic sensors, such as the DVS retina, which has a latency of 12–20 μs, this system is able to process events in real time even if the worst case occurs. Fig. 11 shows the input event rate for the best case and worst case previously mentioned while Fig. 12 shows the output of Poker cards after being processed by different convolution engines with different kernel sizes.

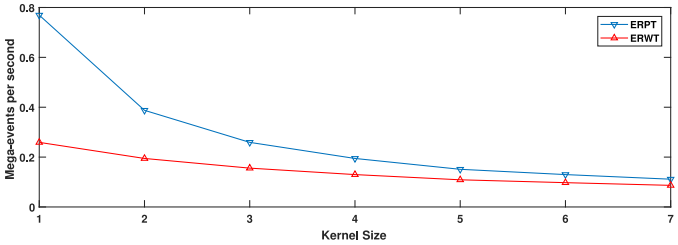

Fig. 11. Input event rate for 64 convolution engines. The blue line represents the event rate for the best processing time and the red line for the worst case.

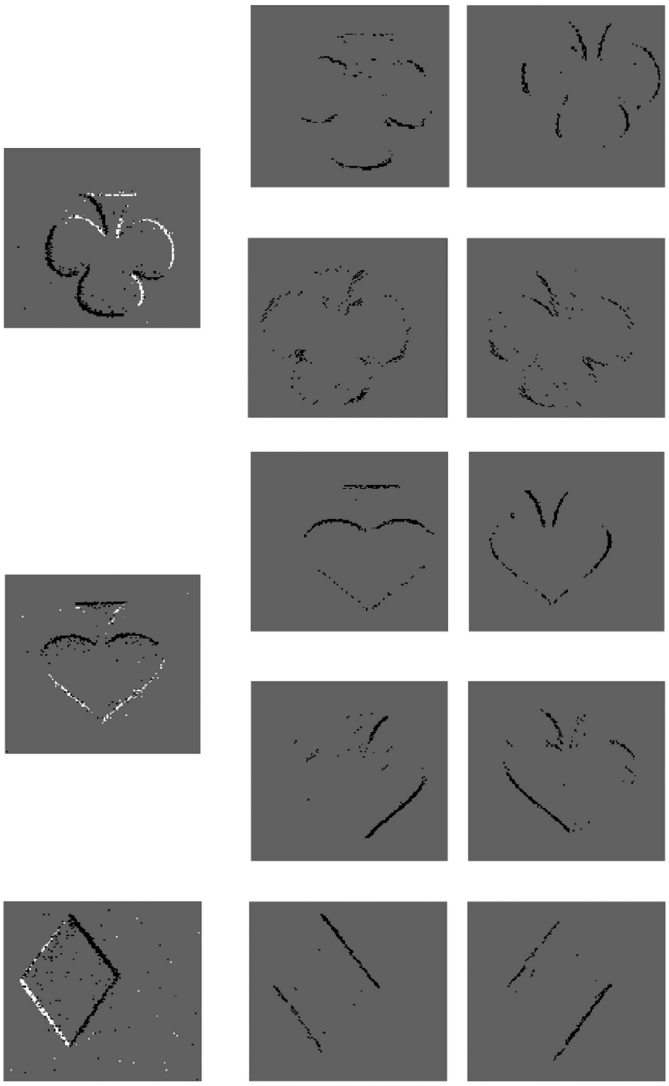

Fig. 12. Example of filters output for different images and kernel sizes. Club and spade card output are two Sobel filters (first row) and two directional Gabor filters (second), while diamond card output consist of Gabor filters to detect edges. The kernel sizes of filters for each image are 3×3 (club), 5×5 (spade) and 7×7 (diamond).

In order to demonstrate a practical approach of this kind of engine, a second experiment was developed. This experiment consists in showing to a DVS retina a propeller with a disk circle, with a dot inside, turning at high speed (2000 revolutions per second). The input stimulus contains two circles: one for the disk and another one for the dot, as shown in Fig. 13.a. The purpose of this experiment is to detect the internal circle with the 64 convolution engines configured with a 7×7 kernel. As can be seen in Fig. 13.b, the histogram of the output stream of events corresponds to the internal dot of the circle.

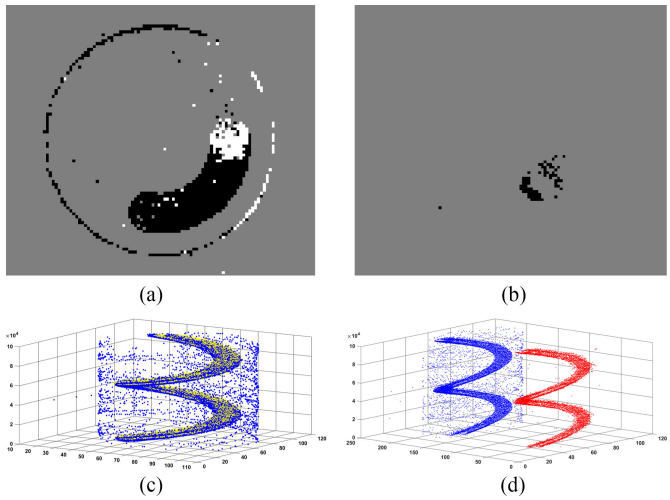

Fig. 13. Input Image (a) and convolution engine output (b) that detects internal circle. Image (c) shows a representation of all events in a period of time, blue points represents input events and yellow points correspond to system output. Image (d) represents the input events (blue) and output events of the system (red).

However, in order to give more precise results and to determine the latency between input and output, the events behavior in this experiment was analyzed. In Fig. 13.c, the input events(blue) are merged with the output events(yellow). As can be seen, the output events follow the internal circle (dot).

Another visualization of the same results is shown in Fig. 13.d where the input events(blue) are compared with the output events (red). In this representation, the effects of the convolution operation can be seen, where the events that do not correspond to the internal circle are filtered. Both figures show that the latency between the input and output events is insignificant, thus it can be stated that the system is able to process input without any delay. As is mentioned before, the design presented is able to process traffic from a DVS in real time. In order to have an equivalence with frame-based cameras, a DVS sensor provides timing resolutions better than 100 kFrames/s [12].

V. Analysis and Comparison

Regarding other FPGA/ASIC implementations of event-driven convolution engines, Serrano *et al.* [25] and Camuñas *et al.* [26], set the convolutional node to update neuron states row by row. They developed convolution chips that manage to convolve larger kernels (up to 16×16 and 32×32 with latencies of 0.35 μs and 0.14 μs, respectively), with row by row capabilities. Although these solutions manage to perform huge numbers of operations per second with low power consumption, these chips do not implement LIF neuron properties (i.e., refractory period) and/or can only perform one convolution, thus deploying medium or large SCNNs would be a complex task. More recent ASIC solutions, such as developed in Camuñas

TABLE II
COMPARISON WITH PRIOR WORK

	Convolution Modules	Max Kernel Size	Platform	Latency in-out (μs)	Input Event Throughput (Meps)	Frequency (MHz)	Refractory Period/ Leakage	Weights Resolution (bits)	Leakage/Refractory resolution (bits)	Adders	Mop/s	Power Consumption per Convolution node (mW)
This work	64	7x7	Zynq 7000 FPGA	1.3-9.01	0.11-0.78	100	yes/yes	8	7/7	7	348.06	0.92
Camuñas et al. [42]	22	10x10	Spartan 6 FPGA	0.5-32	0.05-3	50	yes/yes	8	8/8	22	68.75	0.35
Serrano et al.[25]	1	16x16	0.35um CMOS	0.059-0.35	2.8-16.7	120	No/yes	6	-/6	1	731.43	66.7 - 165
Camuñas et al. [26]	1	32x32	0.35um CMOS	0.05-0.14	1.77-20	120	No/yes	8	-/8	1	7314.29	200
Camuñas et al.[22]	24	32x32	0.35um CMOS	0.06-0.68	1.47-16.6	1.12	No/yes	4	-/4	24	36000	8.3

et al. [22], present a multi-kernel solution that is able to perform up to 24 row by row convolutions in parallel with a maximum kernel size of 32 $\times$ 32. The multi-kernel property allows to perform multiple convolutions in one chip (e.g., a feature map of a CNN); however, like the other ASICs chips, it does not implement the refractory period of LIF neurons that is needed for SCNNs.

On the other hand the solution proposed in Camuñas *et al.* [42], based on a Spartan-6 FPGA, implements the LIF neuron properties and 22 convolution modules, with multi-kernel capabilities, but without row-by-row support, in the same chip. This solution allows to implement many convolution cores in FPGA with low resources.

It has been previously attempted to develop a row-by-row multi convolution system; however, due to the memory distribution of previous FPGAs, all BRAM resources had to be used for just one convolution engine, which needs only a few rows of the memories. The idea of sharing wasted memory resources to implement several convolutions was also attempted but BRAM performance was not optimal to support it [27]. In the design proposed in this work, BRAM is shared by all convolution modules with their own memory space thanks to the memory arbiter proposed, which performs good with new BRAM of 7-families. This solution could be implemented in an ASIC, giving the best performance. Different convolution nodes could implement different layers of a SCNN working over a shared memory (at different memory spaces), without delaying neither the input nor the intermediate events.

Comparing this implementation with the results presented in Camuñas *et al.* [42], they achieved better latency for small kernel sizes around 0.8 μs (1 $\times$ 1) and 1.2 μs (2 $\times$ 2). Despite the fact that their convolution node has a similar latency for 3 $\times$ 3 and 4 $\times$ 4 kernel size, its latency get worsen for bigger kernel sizes, obtaining a latency of 8, 12 and 15 μs for 5 $\times$ 5, 6 $\times$ 6 and 7 $\times$ 7 kernel size, respectively. Apart from latencies, the use of computer units stands out, while other systems use digital signal processors (DSP) or one adder for each convolution module. In the engine presented in this paper, since the memory is shared, only one convolution engine can read a data row in a given clock cycle. Thus, one row operation is done per cycle, so 7 adders are used for all convolution engines, because 7 $\times$ 7 is the maximum kernel size supported.

With respect to the number of operations, our processor manages to perform a peak of 348.06 Mop/s when 64 convolvers are active and processing 7 $\times$ 7 kernels. Although the number of operations is below those of ASIC solutions: 731.429 Mop/s [25] and 7314.29 Mop/s [26]; these solutions only implement one convolution with larger kernels and lower latency. Our design surpasses the performance of the architecture presented in Camuñas *et al.* [42] which reaches a maximum number of 68.75 Mop/s.

Due to the fact that ASIC chips only perform one convolution operation, in order to make a fair comparison, we measured the power consumption for one synthesized convolution module, obtaining 0.92 mW, which is lower than the power consumption of 200 mW in Camuñas *et al.* [26]. However, regarding the implementation presented in Camuñas *et al.* [22], our design is lower in terms of Mop/s and power consumption (36 KOp/s and 200 mW, respectively), due to the fact that FPGAs consume more power and are usually slower than ASIC solutions. Information of the ASIC and FPGA designs compared in this section are shown in Table II.

FPGA has a high static power consumption, thus effective power is commonly measured as the difference of the system in idle state and processing data in real time. In Idle state the system is not processing events, and thus there is no computation. On the other hand a high input throughput, such as the propeller experiment, represents the worst scenario, because the system is continuously computing incomings events. The power consumption measured is 59 mW, which is lower than other frame-based CNN accelerators implemented in the same chip (Zynq 7100), such as NullHop [6]. NullHop accelerator obtains a power consumption of 750 mW. Although NullHop can perform 128 convolutions, the power consumption is much higher even if we duplicate our system. This comparison demonstrates the low power efficiency obtained by event-based systems.

VI. DISCUSSION AND CONCLUSION

In this paper, we have described an event-based multi-convolution engine system for FPGA. It is able to compute a maximum of 64 convolutions with different kernel sizes, from 1 $\times$ 1 to 7 $\times$ 7, with a latency of 1.3 μs and 9.01 μs. The presented engine is able to read and write data row by row, reducing memory accesses. It also implements LIF neuron properties, such as refractory period and leakage, which enable the system for possible SCNN implementations. The maximum kernel size of 7 $\times$ 7 is limited not only by BRAM memory; larger kernels would need to read and write more than one neighbor row from BRAM banks, because of its data bus width. This would affect the LUT needed resources and the operation frequency of the system for the selected Zynq of this study. With new UltraRAM memories available in new Xilinx devices, such as Zynq Ultrascale, kernel sizes could be increased.

Neuromorphic systems such as the one presented in this work takes inspiration in human brain to process data in real time with low power consumption. Frame-based Convnets process huge amount of data per layer and each layer has to wait for the computation of its previous layer to start. Neuromorphic systems process events generated by event-based sensors (e.g., DVS), resulting in a large spare data that reduce the overall processing time because of the absence of long wait states, and therefore, the power consumption. Spiking convolutional neural networks allow input events from a sensor to propagate through layers without waiting for previous layers as in frame based Convnets. Thus, there are no delays between layers. Despite the advantages of neuromorphic systems, accuracies of frame-based accelerators are still better, due to the simple training algorithms, such as backpropagation and the large number of datasets. However, there are recent works in event-based pattern recognition using SCNNs [43] or other techniques, such as HATS [44], which suggest that it is a matter of time until neuromorphic systems become competitive in terms of speed, classification and power consumption.

In future works, we aim to deploy a SCNN in the convolution processor in order to recognize event based datasets, such as N-MNIST [45] or Poker-DVS [41], and test system behavior. Apart from SCNN implementations we are working on combining the presented system with other neuromorphic platforms, such as Spinnaker, which could perform the fully connected layer of a SCNN in a simpler way.

REFERENCES

[1] J. P. Domínguez-Morales, A. F. Jiménez-Fernández, M. J. Domínguez-Morales, and G. Jiménez-Moreno, "Deep neural networks for the recognition and classification of heart murmurs using neuromorphic auditory sensors," *IEEE Trans. Biomed. Circuits Syst.*, vol. 12, no. 1, pp. 24–34, Feb. 2018.

[2] A. Prasoon, K. Petersen, C. Igel, F. Lauze, E. Dam, and M. Nielsen, "Deep feature learning for knee cartilage segmentation using a triplanar convolutional neural network," in *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2013*, K. Mori, I. Sakuma, Y. Sato, C. Barillot, and N. Navab, Eds. Berlin, Germany: Springer, 2013, pp. 246–253.

[3] H. Cecotti and A. Graser, "Convolutional neural networks for p300 detection with application to brain-computer interfaces," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 33, no. 3, pp. 433–445, Mar. 2011.

[4] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *CoRR*, vol. abs/1409.1, 2014.

[5] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems 25*, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, Eds. Red Hook, NY, USA: Curran Associates, Inc., 2012, pp. 1097–1105. [Online]. Available: http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf

[6] A. Aimar *et al.*, "Nullhop: A flexible convolutional neural network accelerator based on sparse representations of feature maps," *IEEE Trans. Neural Networks and Learn. Syst.*, pp. 1–13, 2018.

[7] Y. H. Chen, T. Krishna, J. S. Emer, and V. Sze, "Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks," *IEEE J. Solid-State Circuits*, vol. 52, no. 1, pp. 127–138, Jan. 2017.

[8] C. Mead, *Analog VLSI and Neural Systems*. Reading, MA, USA: Addison-Wesley, 1989.

[9] P. Sterling and S. Laughlin, *Principles of Neural Design*. Cambridge, MA, USA: Cambridge, MA, USA, 2015.

[10] P. Lichtsteiner, C. Posch, and T. Delbrück, "A 128 × 128 120 dB 15 us latency asynchronous temporal contrast vision sensor," *IEEE J. Solid-State Circuits*, vol. 43, no. 2, pp. 566–576, Feb. 2008.

[11] C. Posch, D. Matolin, and R. Wohlgenannt, "A QVGA 143 dB dynamic range frame-free PWM image sensor with lossless pixel-level video compression and time-domain CDS," *IEEE J. Solid-State Circuits*, vol. 46, no. 1, pp. 259–275, Jan. 2011.

[12] T. Serrano-Gotarredona and B. Linares-Barranco, "A 128 × 128 1.5% contrast sensitivity 0.9% FPN 3 μs Latency 4 mW asynchronous frame-free dynamic vision sensor using transimpedance preamplifiers," *IEEE J. Solid-State Circuits*, vol. 48, no. 3, pp. 827–838, Mar. 2013.

[13] M. Yang, C.-H. Chien, T. Delbrück, and S.-C. Liu, "A 0.5 v 55 w 642-channel binaural silicon cochlea for event-driven stereo-audio sensing," in *Proc. IEEE Int. Solid-State Circuits Conf.*, 2016, pp. 388–389.

[14] A. Jiménez-Fernández *et al.*, "A binaural neuromorphic auditory sensor for FPGA: A spike signal processing approach," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 28, no. 4, pp. 804–818, Apr. 2017.

[15] Y. Xu, C. S. Thakur, R. K. Singh, T. J. Hamilton, R. M. Wang, and A. van Schaik, "A FPGA implementation of the CAR-FAC cochlear model," *Frontiers Neurosci.*, vol. 12, p. 198, 2018. [Online]. Available: https://www.frontiersin.org/article/10.3389/fnins.2018.00198

[16] Y. Tsividis, "Event-driven data acquisition and digital signal processing—A Tutorial," *IEEE Trans. Circuits Syst. II, Express Briefs*, vol. 57, no. 8, pp. 577–581, Aug. 2010.

[17] X. Lagorce, G. Orchard, F. Galluppi, B. E. Shi, and R. B. Benosman, "Hots: A hierarchy of event-based time-surfaces for pattern recognition," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 39, no. 7, pp. 1346–1359, Jul. 2017.

[18] S. B. Furber *et al.*, "Overview of the spinnaker system architecture," *IEEE Trans. Comput.*, vol. 62, no. 12, pp. 2454–2467, Dec. 2013.

[19] S. Schmitt *et al.*, "Neuromorphic hardware in the loop: Training a deep spiking network on the brainscales wafer-scale system," in *Proc. Int. Joint Conf. Neural Netw.*, 2017, pp. 2227–2234.

[20] C. Farabet *et al.*, "Comparison between frame-constrained fix-pixel-value and frame-free spiking-dynamic-pixel convnets for visual processing," *Frontiers Neurosci.*, vol. 6, 2012, Art. no. 32.

[21] J. H. Lee, T. Delbrück, and M. Pfeiffer, "Training deep spiking neural networks using backpropagation," *Frontiers Neurosci.*, vol. 10, p. 508, 2016. [Online]. Available: https://www.frontiersin.org/article/10.3389/fnins.2016.00508

[22] L. Camuñas-Mesa, C. Zamarreño-Ramos, A. Linares-Barranco, A. J. Acosta-Jiménez, T. Serrano-Gotarredona, and B. Linares-Barranco, "An event-driven multi-kernel convolution processor module for event-driven vision sensors," *IEEE J. Solid-State Circuits*, vol. 47, no. 2, pp. 504–517, Feb. 2012.

[23] A. Linares-Barranco *et al.*, "On the AER convolution processors for FPGA," in *Proc. IEEE Int. Symp. Circuits Syst.*, May 2010, pp. 4237–4240.

[24] C. Zamarreño-Ramos, A. Linares-Barranco, T. Serrano-Gotarredona, and B. Linares-Barranco, "Multicasting Mesh AER: A scalable assembly approach for reconfigurable neuromorphic structured AER systems. Application to ConvNets," *IEEE Trans. Biomed. Circuits Syst.*, vol. 7, no. 1, pp. 82–102, Feb. 2013.

[25] R. Serrano-Gotarredona *et al.*, "On real-time AER 2-d convolutions hardware for neuromorphic spike-based cortical processing," *IEEE Trans. Neural Netw.*, vol. 19, no. 7, pp. 1196–1219, Jul. 2008.

[26] L. Camuñas-Mesa, A. Acosta-Jiménez, C. Zamarreño-Ramos, T. Serrano-Gotarredona, and B. Linares-Barranco, "A 32 × 32 pixel convolution processor chip for address event vision sensors with 155 ns event latency and 20 meps throughput," *IEEE Trans. Circuits Syst. I, Regular Papers*, vol. 58, no. 4, pp. 777–790, Apr. 2011.

[27] *7 Series FPGAs Memory Resources. User Guide*, Xilinx, San Jose, CA, USA, 2016.

[28] V. Nair and G. E. Hinton, "Rectified linear units improve restricted Boltzmann machines," in *Proc. 27th Int. Conf. Mach. Learn.*, 2010, pp. 807–814.

[29] S. C. Wong, M. Jasiunas, and D. Kearney, "Fast 2d convolution using reconfigurable computing," in *Proc. 8th Int. Symp. Signal Process. Appl.*, Aug. 2005, vol. 2, pp. 791–794.

[30] C. Bartolozzi *et al.*, *Neuromorphic Systems*. Singapore: World Scientific, Nov. 2016.

[31] A. N. Burkitt, "A review of the integrate-and-fire neuron model: I. homogeneous synaptic input," *Biol. Cybern.*, vol. 95, no. 1, pp. 1–19, 2006.

[32] D. Scherer, A. Müller, and S. Behnke, "Evaluation of pooling operations in convolutional architectures for object recognition," in *Artificial Neural Networks - ICANN 2010 [Lecture Notes in Computer Science 6354 (Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)]*, 2010, pp. 92–101.

[33] J. A. Pérez-Carrasco *et al.*, "Mapping from frame-driven to frame-free event-driven vision systems by low-rate rate coding and coincidence processing—Application to feedforward convnets," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 35, no. 11, pp. 2706–2719, Nov. 2013.

[34] K. Hwang, S. P. Su, and L. M. Ni, "Vector computer architecture and processing techniques," *Adv. Comput.*, vol. 20, pp. 115–197, 1981.

[35] Z. Chen and D. Kaeli, "Balancing scalar and vector execution on GPU architectures," in *Proc. IEEE Int. Parallel Distrib. Process. Symp.*, May 2016, pp. 973–982.

[36] A. R. M. AMBA, "Axi4-stream protocol specification," 4.

[37] R. Berner, T. Delbrück, A. Civit-Balcells, and A. Linares-Barranco, "A 5 meps $100 USB2.0 address-event monitor-sequencer interface," in *Proc. IEEE Int. Symp. Circuits Syst.*, 2007, pp. 2451–2454.

[38] The Adress-Event Representation Communication Protocol. Online. Available: https://www.ini.uzh.ch/ amw/scx/std002.pdf

[39] T. Iakymchuk *et al.*, "An AER handshake-less modular infrastructure PCB with x8 2.5 Gbps LVDS serial links," in *Proc. IEEE Int. Symp. Circuits Syst.*, 2014, pp. 1556–1559.

[40] A. Ríos-Navarro, J. P. Domínguez-Morales, R. Tapiador-Morales, D. Gutierrez-Galan, A. Jiménez-Fernández, and A. Linares-Barranco, "A 20 mevps/32 mev event-based USB framework for neuromorphic systems debugging," in *Proc. 2nd Int. Conf. Event-Based Control, Commun. Signal Process.*, Jun. 2016, pp. 1–6.

[41] T. Serrano-Gotarredona and B. Linares-Barranco, "Poker-DVS and MNIST-DVS. Their history, how they were made, and other details," *Frontiers Neurosci.*, vol. 9, p. 481, 2015. [Online]. Available: https://www.frontiersin.org/article/10.3389/fnins.2015.00481

[42] L. A. Camuñas-Mesa, Y. L. Domínguez-Cordero, A. Linares-Barranco, T. Serrano-Gotarredona, and B. Linares-Barranco, "A configurable event-driven convolutional node with rate saturation mechanism for modular convnet systems implementation," *Frontiers Neurosci.*, vol. 12, p. 63, 2018. [Online]. Available: https://www.frontiersin.org/article/10.3389/fnins.2018.00063

[43] N. Zheng and P. Mazumder, "Online supervised learning for hardware-based multilayer spiking neural networks through the modulation of weight-dependent spike-timing-dependent plasticity," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 29, no. 9, pp. 4287–4302, Sep. 2018.

[44] A. Sironi, M. Brambilla, N. Bourdis, X. Lagorce, and R. Benosman, "HATS: Histograms of averaged time surfaces for robust event-based object classification," *CoRR*, vol. abs/1803.07913, 2018. [Online]. Available: http://arxiv.org/abs/1803.07913

[45] G. Orchard, A. Jayawant, G. K. Cohen, and N. Thakor, "Converting static image datasets to spiking neuromorphic datasets using saccades," *Frontiers Neurosci.*, vol. 9, p. 437, 2015. [Online]. Available: https://www.frontiersin.org/article/10.3389/fnins.2015.00437