Event-based Row-by-Row Multi-convolution engine for Dynamic-Vision Feature Extraction on FPGA

Ricardo Tapiador-Morales, Antonio Rios-Navarro, Juan P. Dominguez-Morales, D. Gutierrez-Galan,
M. Dominguez-Morales, A. Jimenez-Fernandez, Alejandro Linares-Barranco
Robotic and Tech of Computers Lab. University of Seville. Seville, Spain 41012
Email: ricardo@atc.us.es http://www.rtc.us.es/

***Abstract*—Neural networks algorithms are commonly used to recognize patterns from different data sources such as audio or vision. In image recognition, Convolutional Neural Networks are one of the most effective techniques due to the high accuracy they achieve. This kind of algorithms require billions of addition and multiplication operations over all pixels of an image. However, it is possible to reduce the number of operations using other computer vision techniques rather than frame-based ones, e.g. neuromorphic frame-free techniques. There exists many neuromorphic vision sensors that detect pixels that have changed their luminosity. In this study, an event-based convolution engine for FPGA is presented. This engine models an array of leaky integrate and fire neurons. It is able to apply different kernel sizes, from 1x1 to 7x7, which are computed row by row, with a maximum number of 64 different convolution kernels. The design presented is able to process 64 feature maps of 7x7 with a latency of 8.98μs.**

***Index Terms*—Convolutional Neural Networks, FPGA, computer vision, artificial intelligence, deep learning.**

I. Introduction

Convolutional Neural Networks (CNN) have become one of the most popular solutions for image classification in real-time, due to the relatively simple supervised training and their efficiency extracting features from a scene. This kind of neural networks are very useful in vision tasks, such as classification, recommender systems and natural language processing. CNNs are usually trained using backpropagation algorithm [1], where the CNN's output is constantly 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).

In spite of the good results extracting features of images given by this kind of networks[2][3], they require many computational resources due to the great number of operations that a convolution step requires per pixel (multiplications and accumulations).

Currently, CNNs are usually deployed in high-end servers with multiple GPUs. Despite of the accuracy and the reduced latency obtained from these architectures, this fact makes it not very useful for real time embedded applications. On the other hand, many mobile phones include a small GPU, but running a CNN on it requires more power demands, what makes it not ideal due to the limited battery capacity of these devices.

There exists many ASIC implementations of convolution accelerators that reduce the power consumption without losing accuracy, reducing the number of operations and pixel precision [4][5].

Although these accelerators manage to run big networks, such as VGG16 [6], frame-based convolutions require many operations due to the great number of pixels that are processed. Many accelerators reduce the number of operations discarding the multiplication of those pixels whose value is 0 [7]. However, there are other image-processing techniques, like neuromorphic engineering event-based ones, that can reduce the number of operations [8].

Neuromorphic engineering takes inspiration from the structure and operation of the human brain, where information, encoded in spikes (also called events), are processed in parallel by massive populations of neurons interconnected via synapses.

Based on this kind of processing, several event-based sensors have been developed, such as the Dynamic Vision Sensor (DVS) [9][10][11]. 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 have 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.

This stream of events can be processed in order to solve object detection for vision tasks, such as object tracking or pattern recognition. Within pattern recognition, there are many techniques to extract features from a scene [12] [13]. This kind of techniques are mostly not implemented in hardware, although there are some neuromorphic hardware solutions, such as SpiNNaker[14].

Spiking Convolutional Neural Networks (SCNN) are a particular type of SNN [15][16].

SCNNs have high accuracy and they are very efficient in terms of power and speed compared to the conventional frame-based ConvNets. This kind of network has been deployed in multiple chip solutions, such as FPGAs or ASICs [17][18].

In this paper a spiking multi-convolution engine is presented. This engine processes input events and applies kernel values row by row over neurons, storing the neuron membrane

978-1-5090-6014-6/18/$31.00 ©2018 IEEE

potential in embedded Block RAM memory of the FPGA and firing spikes if a configurable threshold has been reached. A leakage mechanism has also been implemented. The system has been simulated at FPGA post implementation applying different convolution kernels, visualizing and checking the output from the system.

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

II. Spiking Convolution

In image processing, convolution operation consists in applying a kernel matrix to a pixel, multiplying kernel values with corresponding pixels and adding these results as a single value. The convolution is performed by sliding the kernel over all pixels of the image, generally starting at the top left corner. For simplicity, the boundaries are usually not computed. Mathematically, it is defined as shown in Equation 1, where K is an NxM kernel matrix of the convolution, X the input image and Y the convolved image.

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

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

However, in an event-based processing, not all pixels of a frame are processed because neuromorphic sensors get luminosity changes of the current scene, and these changes are transmitted as events. An event is represented by an (x,y,p) address 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 luminosity detected by the pixel in the current state is higher than the luminosity detected in the previous state, whereas OFF polarity means the opposite: the luminosity detected in the current state is lower than the luminosity detected in the previous state. In a spiking convolution, an input image X is coded in a way where each pixel X(i,j) is represented by a number of events of a visual source output, such as 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):

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

Once all the events of pixel X(i,j) have been received and calculated, the integrator value of the corresponding address Y(i,j) has accumulated X(i+a,j+b)$\forall(a,b)$,(number of events) times the value of the kernel, thus the multiplication operation has been reached. The output of the convolution operation, at this point, is stored in a Y matrix of integrators. There are several ways to send out the resulting Y matrix. In this paper, it is inspired in the leaky integrate-and-fire (LIF) neuron model[19]. The continuous addition of values of the kernel 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 (PTH) or negative threshold (NTH), a spike is generated with the corresponding polarity and x,y address, resetting its membrane potential as shown in Figure 1.

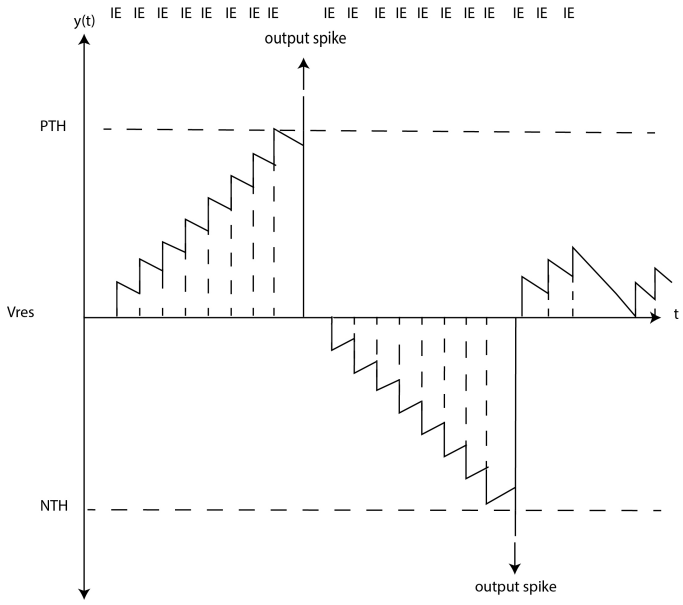

Fig. 1: Changes on the membrane potential of a neuron along time with input events (IE).

A biological neuron reduces its membrane potential due to a leakage in case that the neuron does not receive any kind of excitation. We have implemented this property of biological neurons, because a LIF neuron that has not received any excitation means that it is not giving information about the scene. The leakage decay time of a neuron and its reduced potential value, are parameters that allow controlling the output event rate. This implies that a short decay time indicates that neurons would be resetting more often. On the other hand, larger values for decay time make neurons reset their potential rarely, increasing the number of output spikes.

III. SPIKE CONVOLUTION ENGINE

In this section, three main modules of the architecture are explained for Xilinx FPGAs: (1) memories for membrane potential, timestamps and kernel, based on BRAMs, (2) leakage memories implemented on LUTRAMs and (3) Convolution engine module where a convolution operation is performed.

A. Membrane potential, Timestamps and Kernel memories

In Xilinx FPGAs the main memory resource is Block RAM memory (BRAM). BRAM 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 two types of BRAM memories available in a device can hold either 18k or 36k bits, and the available amount of these memories is device specific. In this design, BRAM is divided in three different memory banks: one bank stores the membrane potential, the second one stores timestamps and the last bank stores the values of the kernels.

The maximum image size that the system is able to compute is 128x128 pixels. The system can process 64 different convolution engines that can access the BRAM memories to read and write the membrane potential and timestamps values for each input event convolution operation. Membrane potential memory is organized in 16 BRAMs, where each one stores rows of 8 membrane potentials. Memories are enabled using the x address of the input event that selects which blocks are read or written, whereas the y address of the input event and convolution engine ID are used to address the memory. This way, each convolution engine has its own memory space using the same memory resources as shown in Fig. 3 a).

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 of 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 block. Both rows are combined in a larger row of 16 elements to be processed by a convolution engine as it shown in Fig 2

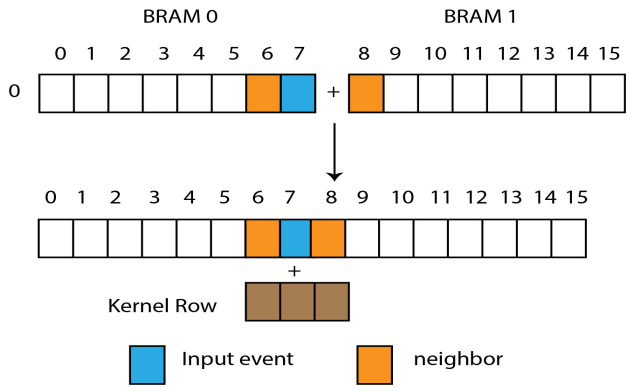

Fig. 2: Rows read for convolution operation.

In this design there is a global counter of 32 bits that is used to compare current time with the timestamps of an event in order to apply a configurable decay value to a neuron. Since all neurons need a timestamp in order to reduce the impact of storing all values, only 8 bits from a 32bit counter are stored. These 8 bits are selected using a configurable slide window. Sliding the window allows to change the timing resolution of timestamps depending on the event rate(Fig 4). The timestamps memory has the same structure as the membrane potential memory, storing the 8bit timestamp in another bank with 16 BRAMs. This value is compared to the global counter to apply the decay during the convolution operation.

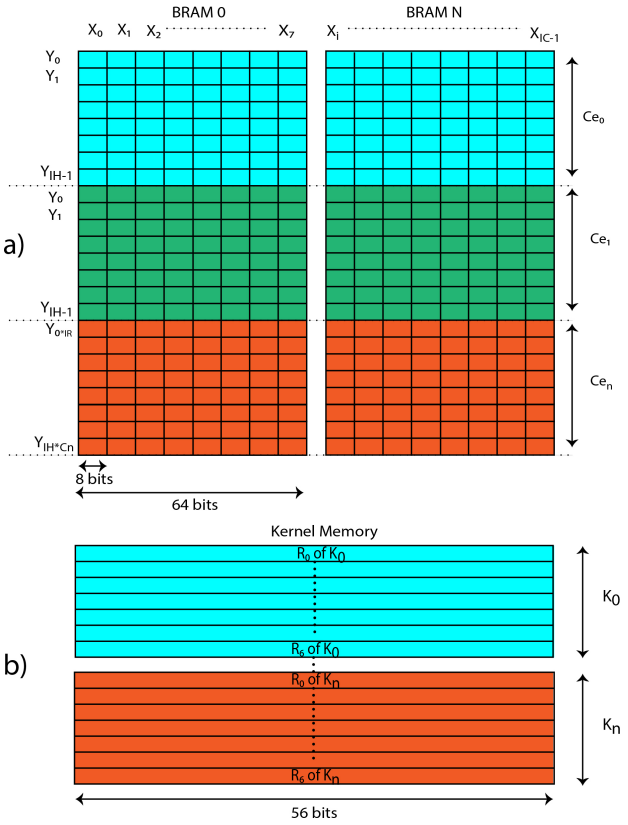

Fig. 3: a) Membrane potential and Timestamps memory structure. b) Kernel memory structure. Each coloured region represents memory space for each convolution engine

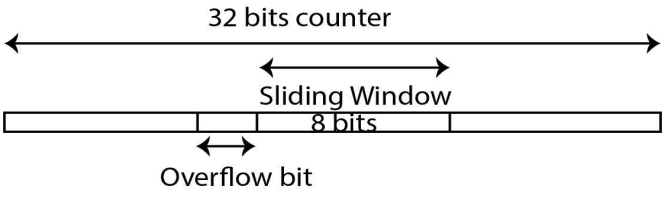

Fig. 4: Global counter system.

Kernel memory consists of a BRAM where values of each kernel are stored row by row. Each row contains a maximum of 7 kernel values with the same resolution as the membrane potential (8 bits) and a number of 7 rows per kernel, Fig 3.

Each memory bank is organized to be read/written row by row, reducing the number of memory accesses. The banks are shared for each convolution engine module, thus a memory arbiter is needed to coordinate the access.

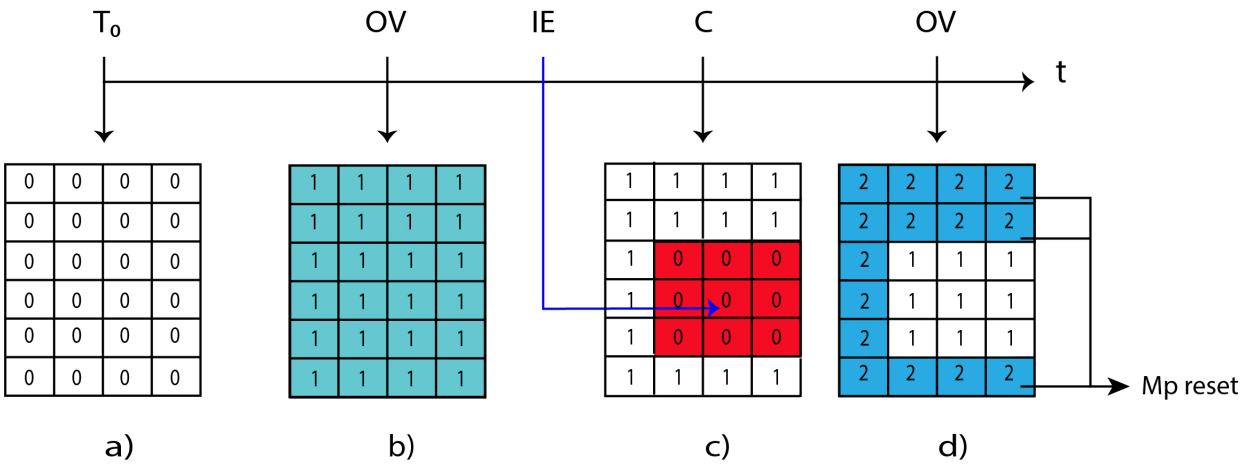

Fig. 5: Neuron leakage illustration based on LUTRAMs for a small region of leakage memory.

B. Leakage memories

Leakage is computed and applied to membrane potentials when a new event arrives to the system. Although there exists a global counter to apply decay, it is possible that this counter has overflowed many times. If this situation occurs, any membrane potential of events that have not been accessed in a long time should have been reset. Thus, a leakage system based in LUTRAM has been implemented.

Multiple LUTs in a Slice Memory (SLICEM) can be combined in various 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 files.

This memory bank consists of 128x128 LUTRAM memories of two bits each, storing the leakage state for each pixel. Fig 5 shows how leakage memory works for a 3x3 kernel. Initially (T0), leakage memory cells have a value of 0 (Fig 5a)). When the global counter produces an overflow (OV), each cell adds 1 to its content (Fig 5b)). When an input event (IE) is convolved (C), the content of those rows that have been accessed during the convolution are reset (Fig 5c)). In case that another overflow occurs and the cells increase their values again, those cells that have reached a value of 2 indicate that a long time has passed from the last access to that event. Therefore, the next time one of those events will be accessed, their membrane potential (MP) will be reset(Fig 5d)).

When an overflow occurs, the convolution engine cannot update the membrane potential memory until the content of each leakage memory has been updated. However, a convolution engine can read from memory during the update, reducing the waiting time. This bank is inside each convolution engine and is read/written in rows, as the aforementioned memories.

C. Convolution Engine

For each incoming event, a convolution operation is performed by the convolution engine module. This module consists of a state machine that 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, reducing the number of memory accesses.

Algorithm 1 Convolution Engine steps

```
Decay mask step
Check timestamps and generate decay mask
Convolution step
for i = 1:16 do
    if Leakage Memory[i] == 2 then
        Reset Membrane Potential
    else
        Add Kernel row to MP row and apply decay
        if Convolution operation result > Threshold then
            (P,X,Y) → Output FIFO
            Reset Membrane Potential
        end if
    end if
    Update Timestamps and Leakage Memory
end for
Write Leakage, Timestamps and Membrane Potential Memories
```

The convolution operation is divided into two phases for each row. The decay mask step calculates the difference between the timestamps of each neuron and the global counter value. If this difference exceeds a configurable leakage time, a decay is applied. This process generates a row-mask in order to apply decay to the correct rows during the convolution operation. The Convolution step updates timestamps and leakage memory for mask-enabled neurons. Before the convolution, the corresponding row values of leakage memory are checked, as was mentioned before. If the leakage value is 2, the membrane potential is reset; otherwise, the convolution operation is done for that neuron. During the convolution operation, when applying a kernel value to a neuron, its membrane potential can reach the threshold. In that case,

the neuron has to fire, resetting the value of the membrane potential and storing its address to an output FIFO. These steps are shown in algorithm 1.

D. Hardware Implementation

The design has been implemented for a Zynq-7100 MMP platform. This platform contains a PSoC with a Dual ARM® Cortex™-A9 MPCore™. The PSoC includes a Kintex-7 FPGA with 444K logic cells in the same chip. Although this chip is used, the design is customizable by reducing/increasing the number of feature maps or the image size to be processed in order to be implemented for any FPGA chip. The interfaces of the accelerator are divided into two different buses: an AXI slave bus [20], which configures the parameters of the system (leakage time, decay, threshold and kernel values), and an AER bus [21][22][23], which receives events from an event-based sensor. A diagram of the whole design is shown in Fig 7.

	Utilization	Available	Percentage
LUT	249660	277400	90%
BRAM	514	755	68%
LUTRAM	36788	108200	34%
FLIP-FLOPS	122056	554800	22%

TABLE I: Resources Table for Kintex 7

The design is configurable through the AXI bus using a DMA with one channel that reads from the memory. In this work, both interfaces, AER and AXI, have been simulated using a C/C++ program which allows to manipulate the signals of the design.

IV. Test Scenario

The simulation scenario to test this engine was done in Mentor Graphic Modelsim using System Verilog DPI-C interface. This characteristic of System Verilog allows writing test benches using C/C++, which facilitates the manipulation of input/output signals.

As was mentioned before, the design has two different buses: an AXI-Slave and an AER Input. The C++ code acts as the master of both buses, writing configuration parameters and kernel values for the AXI bus, and writing events to the AER interface. For this experiment, a club card image from the Poker-DVS database [24] was selected to check the results after applying convolution kernels. This kind of files are written in AEDAT[25] format, which represents events with their corresponding timestamps. These files can be visualized using programs such as jAER [26].

The output from the system will be saved in a file and visualized in jAER to verify the behavior of each filter as shown in Fig. 6.

This test is divided into two parts: in the first part, events are processed with different kernel sizes, obtaining the processing time for a convolution engine. However, in CNN odd kernel sizes are commonly used. With the aim of determining the correct behavior of the system on a more stressful situation, a second test using 64 convolution engines was conducted. The kernels in this test were configured with different Sobel filters

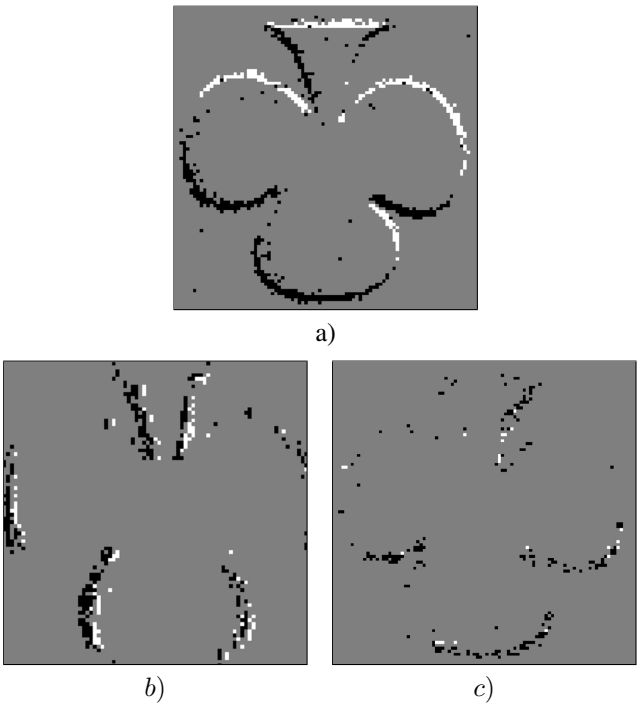

Fig. 6: Source Image(a) and convolved image with K1(b) and K2(c).

to detect rows and columns, as shown in (3). The kernel sizes in this experiment were 3x3, 5x5 and 7x7.

$$\text{K1} = \begin{pmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{pmatrix} \text{K2} = \begin{pmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{pmatrix} \quad (3)$$

For a more realistic scenario inside a simulation, the design was tested at the post implementation level. At that level, each component is mapped into a physical device with their corresponding delay, thus the time obtained is the closest one to a real implementation on a chip.

V. Results

For the first scenario, identity kernels were configured in order to measure the time since an event arrives at the engine until the system had finished processing this event. The results obtained for one convolution engine are $0.1\,\mu s$ to process a 1x1 kernel and $0.7\,\mu s$ for a 7x7 kernel (Fig. 8). Convolving one kernel row requires 10 cycles (memory access, pipeline stages...), for a clock of $10\,ns$; processing a row takes $0.1\,\mu s$. Therefore, for a size L kernel, the processing time for an event is $0.1\,\mu s$*L.

Despite these values, it occurs sporadically that the global counter overflows during a convolution. This implies that the leakage memories need to be updated. As was mentioned before, if a convolution engine is in write state, it cannot request memory access until the memory is updated, thus it has to wait. This case represents the worst situation in terms of latency. Updating a row of leakage memory requires two

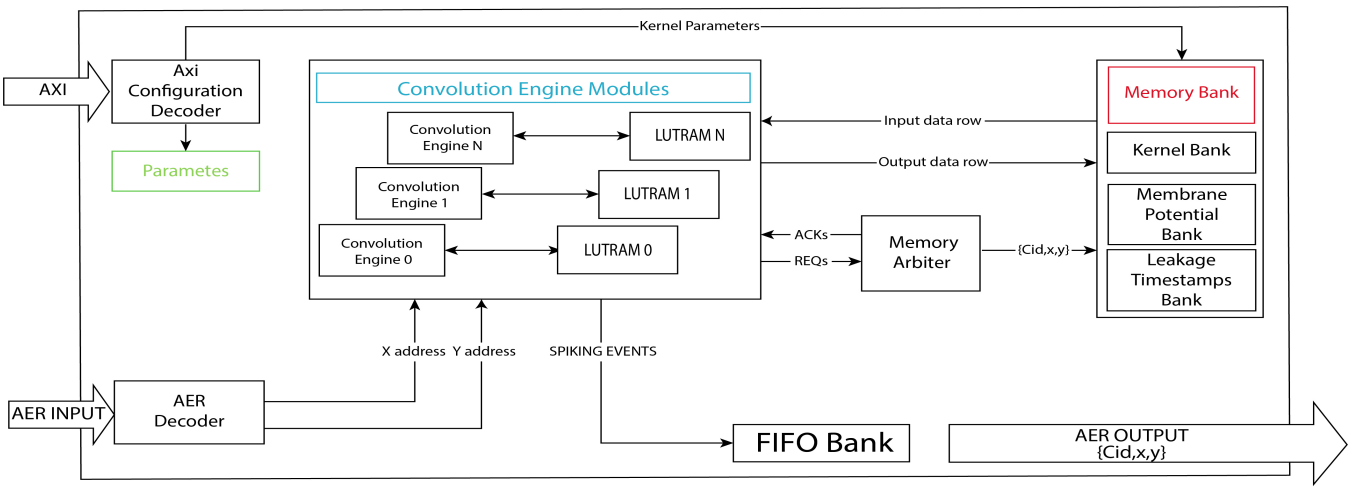

Fig. 7: Block diagram of the system architecture

clock cycles. Since there are 128 rows, a time of 2,56μs is needed.

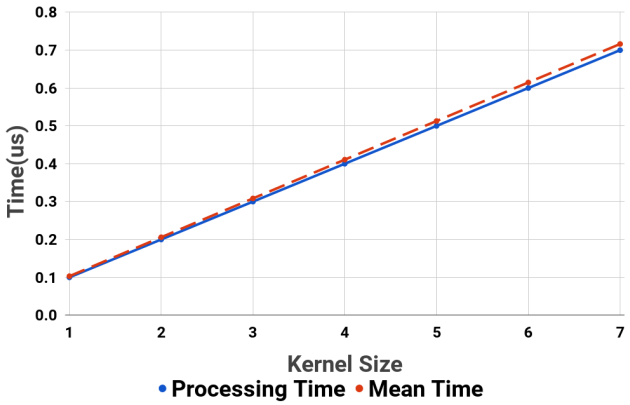

Fig. 8: Processing time for different kernel sizes. The blue line represents the ideal processing time of an event. The red line is the average processing time after the computation of the club card image.

The probability of overflow depends on the configuration of the global counter and of the kernel size. Even for large kernels, if the global counter is properly configured according to the throughput, the probability of an overflow during a convolution is very small due to the fact that the memory could be updating while the system is not working, or when a convolution engine is waiting, the memory arbiter answers. Although the probability is low, the mean time to process each of the events from the club image was calculated, giving results not so far from the best case (0.103μs for 1x1 and 0.71μs for 7x7, shown in Fig 8), demonstrating that the updating time of leakage memories is insignificant.

In current SCNNs, odd kernels are commonly used and each convolution layer has more than one feature map. With the aim of testing a more realistic approach, a second scenario was prepared. In this scenario, the system was configured with 64 convolution engines with different odd kernel sizes in order to measure the time to process an event.

Kernel Size	Time(μs)
3x3	3.86
5x5	6.4
7x7	8.98

TABLE II: Processing time of different odd kernel sizes for 64 Convolution engines

The DVS retina has a latency around 15-20 µs. Comparing the sensor latency with one input event latency to be processed by 64 convolution engines (see Table II), we demonstrated that for a number of 64 Convolution engines the system is able to process events in real time using any kernel size.

Although the parameters are properly configured, the number of spiking events at the output of the system is quite high for SCNNs. A possible solution to reduce the firing rate of events is to implement a refractory period for each pixel, so an event will not fire again until that period is met.

VI. Conclusions

In this paper, the authors presented an event-based convolution system that is able to compute a maximum of 64 convolutions with different kernel sizes, from 1x1 to 7x7. The design presented improves other SCNN FPGA engines, such as Zamarreño et. al[27] , in terms of processing time and image size. Focusing on processing time, as was shown in Table III, for a kernel size of 11x11, Zamarreño et. al. [27] it takes about 3 µs, which is the same time that the implementation presented in this work takes to convolve a layer of 64 feature map filters of 3x3 at lower latency. It is also important to highlight the number of adders used for both systems, because of the fact that it is an important factor in power consumption. In the design presented in this paper, the convolution engines

cannot work in parallel due to the fact that memory resources are being shared. Therefore, the maximum number of adders match the maximum kernel size row to convolve, which is 7. However, in Zamarreño's systems, each convolution module works in parallel, meaning that for any kernel size, the system needs a number of 64 adders for 64 parallel modules. This implies a high impact in computational resources and power consumption in comparison with the system presented.

	Zamarreño et. al.	This work
Image Size	64x64	128x128
Kernel max size	11x11	7x7
Convolution Modules	64	64
Frequency (MHz)	120	100
Latency per event(μs)	3(11x11)	0.7(7x7)
Adders	64	7

TABLE III: Comparison table of Spiking Convolution Engines

The work presented is able to process higher images with 64 convolution modules faster than other spiking convolution engines, being able to process events from neuromorphic sensors in real time. As future work, the design will be deployed in an FPGA chip implementing the refractory period of LIF neurons to reduce the firing rate at the output and connecting it to a DVS retina.

VII. Acknowledgment

This work is supported by the excellence project from the Spanish government grant (with support from the European Regional Development Fund) COFNET (TEC2016-77785-P) and by the NPP project funded by SAIT (2015-2018). The work of R. Tapiador has been supported by a Formación de Personal Investigador Scholarship from the University of Seville. The work of J.P. Dominguez-Morales was supported by a Formación de Personal Universitario Scholarship from the Spanish Ministry of Education, Culture and Sport.

References

[1] M. T. Hagan and M. B. Menhaj, "Training feedforward networks with the Marquardt algorithm," *IEEE transactions on Neural Networks*, vol. 5, no. 6, pp. 989–993, 1994.

[2] J. P. Dominguez-Morales, A. F. Jimenez-Fernandez, M. J. Dominguez-Morales, and G. Jimenez-Moreno, "Deep neural networks for the recognition and classification of heart murmurs using neuromorphic auditory sensors," *IEEE Transactions on Biomedical Circuits and Systems*, vol. 12, no. 1, pp. 24–34, Feb 2018.

[3] M. Szarvas, A. Yoshizawa, M. Yamamoto, and J. Ogata, "Pedestrian detection with convolutional neural networks," in *IEEE Proceedings. Intelligent Vehicles Symposium, 2005.*, 2005, pp. 224–229.

[4] L. Cavigelli and L. Benini, "Origami: A 803-GOp/s/W Convolutional Network Accelerator," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 27, no. 11, pp. 2461–2475, nov 2017.

[5] Y. H. Chen, T. Krishna, J. S. Emer, and V. Sze, "Eyeriss: An Energy-Efficient Reconfigurable Accelerator for Deep Convolutional Neural Networks," *IEEE Journal of Solid-State Circuits*, vol. 52, no. 1, pp. 127–138, 2017.

[6] K. Simonyan and A. Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition," *CoRR*, vol. abs/1409.1, 2014.

[7] A. Aimar, H. Mostafa, E. Calabrese, A. Rios-Navarro, R. Tapiador-Morales, I.-A. Lungu, M. B. Milde, F. Corradi, A. Linares-Barranco, S.-C. Liu, and T. Delbrück, "NullHop: A Flexible Convolutional Neural Network Accelerator Based on Sparse Representations of Feature Maps, volume = abs/1706.0, year = 2017," *CoRR*.

[8] C. Farabet, R. Paz, J. Pérez-Carrasco, C. Zamarreño, A. Linares-Barranco, Y. LeCun, E. Culurciello, T. Serrano-Gotarredona, and B. Linares-Barranco, "Comparison between frame-constrained fix-pixel-value and frame-free spiking-dynamic-pixel convnets for visual processing," *Frontiers in neuroscience*, vol. 6, p. 32, 2012.

[9] T. Serrano-Gotarredona and B. Linares-Barranco, "A 128$,times$ 128 1.5% Contrast Sensitivity 0.9% FPN 3 #x00B5;s Latency 4 mW Asynchronous Frame-Free Dynamic Vision Sensor Using Transimpedance Preamplifiers," *IEEE Journal of Solid-State Circuits*, vol. 48, no. 3, pp. 827–838, 2013.

[10] 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 Journal of Solid-State Circuits*, vol. 46, no. 1, pp. 259–275, 2011.

[11] P. Lichtsteiner, C. Posch, and T. Delbruck, "A 128 times; 128 120 dB 15 #956;s Latency Asynchronous Temporal Contrast Vision Sensor," *IEEE Journal of Solid-State Circuits*, vol. 43, no. 2, pp. 566–576, feb 2008.

[12] 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 transactions on pattern analysis and machine intelligence*, vol. 39, no. 7, pp. 1346–1359, 2017.

[13] X. Clady, J.-M. Maro, S. Barré, and R. B. Benosman, "A motion-based feature for event-based pattern recognition," *Frontiers in neuroscience*, vol. 10, p. 594, 2017.

[14] S. B. Furber, D. R. Lester, L. A. Plana, J. D. Garside, E. Painkras, S. Temple, and A. D. Brown, "Overview of the spinnaker system architecture," *IEEE Transactions on Computers*, vol. 62, no. 12, pp. 2454–2467, 2013.

[15] Y. Cao, Y. Chen, and D. Khosla, "Spiking Deep Convolutional Neural Networks for Energy-Efficient Object Recognition," *International Journal of Computer Vision*, vol. 113, no. 1, pp. 54–66, may 2015. [Online]. Available: https://doi.org/10.1007/s11263-014-0788-3

[16] J. H. Lee, T. Delbruck, and M. Pfeiffer, "Training Deep Spiking Neural Networks Using Backpropagation," *Frontiers in Neuroscience*, vol. 10, p. 508, 2016. [Online]. Available: https://www.frontiersin.org/article/10.3389/fnins.2016.00508

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

[18] A. Linares-Barranco, R. Paz-Vicente, F. Gmez-Rodriguez, A. Jimnez, M. Rivas, G. Jimnez, and A. Civit, "On the aer convolution processors for fpga," in *Proceedings of 2010 IEEE International Symposium on Circuits and Systems*, May 2010, pp. 4237–4240.

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

[20] A. R. M. AMBA, "AXI4-Stream Protocol Specification," 4.

[21] R. Berner, T. Delbruck, A. Civit-Balcells, and A. Linares-Barranco, "A 5 Meps $100 USB2.0 address-event monitor-sequencer interface, year = 2007," in *2007 IEEE International Symposium on Circuits and Systems*. IEEE, pp. 2451–2454.

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

[23] T. Iakymchuk, A. Rosado, T. Serrano-Gotarredona, B. Linares-Barranco, A. Jimenez-Fernandez, A. Linares-Barranco, and G. Jimenez-Moreno, "An AER handshake-less modular infrastructure PCB with x8 2.5 Gbps LVDS serial links, year = 2014," in *Circuits and Systems (ISCAS), 2014 IEEE International Symposium on*. IEEE, pp. 1556–1559.

[24] T. Serrano-Gotarredona and B. Linares-Barranco, "Poker-DVS and MNIST-DVS. Their History, How They Were Made, and Other Details," *Frontiers in Neuroscience*, vol. 9, p. 481, 2015. [Online]. Available: https://www.frontiersin.org/article/10.3389/fnins.2015.00481

[25] AEDAT file formats. [Online]. Available: http://inilabs.com/support/software/fileformat/

[26] T. Delbruck, "jAER open source project (2007)."

[27] C. Zamarreno-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 Transactions on Biomedical Circuits and Systems*, vol. 7, no. 1, pp. 82–102, feb 2013.