Event-Driven Configurable Module with Refractory Mechanism for ConvNets on FPGA

L. A. Camuñas-Mesa, Y. Domínguez-Cordero, T. Serrano-Gotarredona and B. Linares-Barranco

Instituto de Microelectrónica de Sevilla (IMSE-CNM), CSIC y Universidad de Sevilla, Sevilla, Spain

***Abstract*—We have developed a fully configurable event-driven convolutional module with refractory period mechanism that can be used to implement arbitrary Convolutional Neural Networks (ConvNets) on FPGAs following a 2D array structure. Using this module, we have implemented in a Spartan6 FPGA a 4-layer ConvNet with 22 convolutional modules trained for poker card symbol recognition. It has been tested with a stimulus where 40 poker cards were observed by a Dynamic Vision Sensor (DVS) in 1s time. A traffic control mechanism is implemented to down-sample high speed input stimuli while keeping spatio-temporal correlation. For slow stimulus play back, a 96% recognition rate is achieved with a power consumption of 0.85mW. At maximum play back speed, the recognition rate is still above 63% when less than 20% of the input events are processed.**

***Index Terms*—ConvNets, Refractory Period, Event-driven processing, Reconfigurable Networks, AER vision**

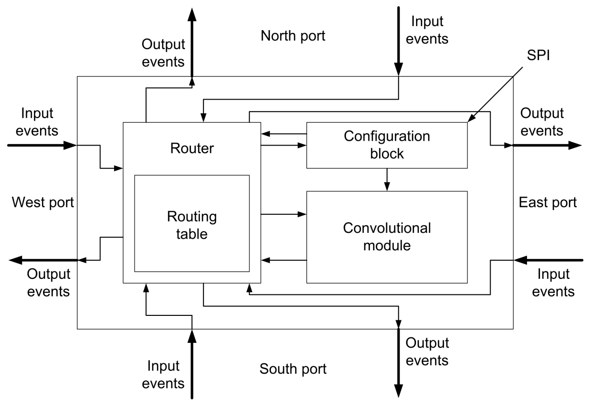

Fig. 1. Block diagram for the node designed to build 2D arrays.

I. Introduction

The development of bio-inspired event-driven neuromorphic Dynamic Vision Sensors (DVS) [1]–[3] provides a revolutionary way of capturing visual scenes by generating flows of events representing real-time visual information. Each pixel in a DVS operates autonomously and sends out an event (spike) whenever it senses a change of light greater than a preset threshold. Therefore, the DVS generates a continuous flow of events with a high temporal resolution (sub-microsecond) representing reality dynamically, without frames. Spiking Neural Networks (SNNs) [4] process flows of events using different neuronal and synaptic models, performing tasks like object tracking [5] or shape recognition [6].

The hardware implementation of complex SNNs is limited in general by the all-to-all connectivity between layers of neurons. This restriction is overcome by Convolutional Neural Networks (ConvNets) [7], where each neuron is connected only to a subset of neurons in the following layer, known as projective field. These projective fields are common for all neurons in each layer, which reduces the amount of connections in the network, facilitating its hardware implementation. Although the ConvNets were originally developed for frame-driven processing of static images [7], a previous work proposed a method to transform a frame-driven ConvNet into an event-driven one and implement it in software [8].

In this work, we present a configurable convolutional module described in VHDL for FPGAs with a programmable refractory period mechanism reproducing the behavior of biological neurons based on a leaky Integrate-And-Fire model. This module can build arbitrary large-scale ConvNets by following the 2D array structure proposed by Zamarreño et al. [9]. A 4-layers ConvNet with 22 convolutional blocks trained for poker card symbol recognition has been implemented in one Spartan6 FPGA. A traffic control mechanism has been proposed to discard input events when the event rate is too high to process all the information in real time.

II. Network node

The network architecture proposed in this work is based on a configurable node that can be assembled in large 2D arrays, following the block diagram in Fig. 1, with connections to 4 other neighboring nodes through North, South, East and West ports. All these ports carry bidirectional flows of events (input and output). Internally, all ports are connected to a router, which sends each incoming event to either the appropriate output port or to the local convolutional module, depending on both the event header and the previously programmed routing table, following the destination-driven protocol [9]. All programmable parameters both in the router or in the convolutional module are set by the configuration block, which receives commands through a Serial Peripheral Interface (SPI).

III. Convolutional module

The convolutional module designed inside the network node shown in Fig. 1 computes the 2D convolution of the input events $ev_{in}(t,x,y,p)$ with a kernel $w_k(x,y)$, generating the output events $ev_{out}(t,x,y,p)$, where t is time, x and y are the spatial coordinates, p is the polarity of the event, and k is the kernel id, as multi-kernel processing is allowed (applying a different kernel k to each incoming event, depending on where it comes from). The state of the convolution (the values of all pixels or neurons) is stored in the Neuron Memory, while the Kernel Memory stores all the kernel values and their corresponding parameters (x- and y-size, and center shift). Another memory is used to implement the refractory period mechanism described later. The convolutional module receives configuration data through an SPI interface, which is used to write the kernels (with their parameters) and the parameters of the controller (threshold, leakage, refractory period).

The convolutional module is fully configurable, including some parameters which have to be adjusted before the hardware implementation (address space of input and output events, size of the different memories and range of the refractory period) and some other parameters which can be modified after implementation using the SPI interface (integration threshold, leakage parameters, refractory period, and the kernel values with their parameters). The behavior of the module can be described by the main convolutional operation and other mechanisms which take place simultaneously: global leackage, refractory period and traffic control.

A. Convolutional operation

Each time a new incoming event arrives at the convolutional module, it is stored in the input FIFO and follows these steps:

1) Read the event from the input FIFO, with the address (x_{in}^i, y_{in}^i), the polarity p_{in}^i, and the kernel id k^i.
2) Using k^i, read the size and center shift of the kernel. This information, with the event address, gives the coordinates of the pixels which have to be updated.
3) Calculate the positions of the pixels in the neuron memory, and the kernel weights in the kernel memory.
4) One by one, read pixel value and kernel weight, and add them. If the event is negative, invert the kernel weight.
5) If the result is larger than the threshold, check the refractory period T_R. If firing event is allowed, go to step 6. If not, update the pixel.
6) Generate an output event with address (x_{out}^j, y_{out}^j) and polarity p_{out}^j, and write it in the output FIFO, reset the pixel and update it in its memory position.

B. Global leakage

In parallel with the convolutional operation, a global leakage process runs continuously. A global 32-bit counter is increased with every clock cycle, until it reaches the previously programmed value T_{leak}. This process has the highest priority, and every time it reaches T_{leak} it decreases all neuron values by N_{leak} if they are positive, and increases them if they are negative, never crossing the reset value.

C. Refractory period

The main novelty of this work is the hardware implementation of a mechanism that emulates the refractory period property of biological neurons. This property guarantees a minimum separation in time (given by T_R) between two consecutive spikes generated by a single neuron.

Fig. 2(a) illustrates how the neuron state is increased every time a new input event arrives until it reaches the threshold. At $t = t_0$, an output event is sent and the neuron state is reset, so the module reads the present time t_0 in the $32-bit$ global counter (see Fig. 2(b)) and calculates the future time when it will allow another output spike t_{lim}. This future time is given by $t_{lim} = t_0 + T_R - \Delta t$, where t_0 is the present time, T_R the refractory time, and Δt is a small correction applied to compensate for frequency deviations. At $t = t_1$, the neuron reaches the threshold (Fig. 2(a)), but it is not allowed to send an output event, as $t_1 < t_{lim}$. Therefore, the neuron keeps the threshold value until t_2, when a new input event is received and an output event is finally sent (because $t_2 > t_{lim}$).

The resolution of the global counter is 32 bits, so that is also the size of t_{lim}. However, we reduced the needed memory resources by storing only 8 bits per pixel (t_{lim}^{8b} in Fig. 2(b)), from b_{TR} to $b_{TR} - 7$. The value of b_{TR} is a parameter that must be specified before implementation ($31 \geq b_{TR} \geq 7$), and it corresponds to the MSB (Most Significant Bit) of the refractory period T_R. Therefore, the possible values of T_R that can be programmed after implementation will be limited between $T_R^{min} = 2^{b_{TR}}$ and $T_R^{max} = 2^{b_{TR}+1} - 1$.

According to this strategy, 8 bits from t_{lim} (t_{lim}^{8b}) are stored at the refractory period memory for that pixel. After that, the next time this pixel reaches the threshold, the module reads the time t_1 in the $32-bit$ global counter, extracts 8 bits from it (t_1^{8b}, from b_{TR} to $b_{TR} - 7$, see Fig. 2(c)) and compares it with t_{lim}^{8b}. If $t_1^{8b} > t_{lim}^{8b}$, it sends out the event; otherwise, it stores the threshold in the neuron memory and waits for the next incoming event. However, this mechanism can cause wrong decisions, as the bits more significant than b_{TR} are not compared (it can happen that $t_1^{8b} < t_{lim}^{8b}$ while $t_1 > t_{lim}$). To avoid this, a refresh mechanism is implemented by generating a global refresh pulse every time the global counter reaches the value $2^{b_{TR}+1} - 1$ (all bits from b_{TR} to b_0 set to 1), indicating overflow. Every time a global refresh pulse is generated, all overflow flags f_{of} with 1 value are set to 0, while flags with 0 value set their corresponding t_{lim} to 0, making sure that the pixel will be allowed to fire.

Finally, the Δt correction applied to the calculation of t_{lim} is also illustrated in Fig. 2(a). When the first output event is generated at t_0, we assume no previous activity and $\Delta t = 0$. However, the second output event occurs at t_2, although it should have been generated at t_{lim}, being delayed by the refractory period mechanism. In this particular example, after sending out the event at t_2, the next t_{lim} would be calculated as $t_{lim} = t_2 + T_R - \Delta t_2$.

D. Traffic control mechanism

The convolutional module generates a $full_{FIFO}$ signal when the output FIFO is full, and this signal is used to implement a traffic control mechanism in an arbitrary network. Considering that the network receives events from a DVS using AER protocol, we implemented a mechanism which drops input events whenever a module in the network has the $full_{FIFO}$ signal active. Instead of holding the acknowledge and introducing artificial delays, this mechanism reduces dynamically the amount of input events while keeping the spatio-temporal correlation between them.

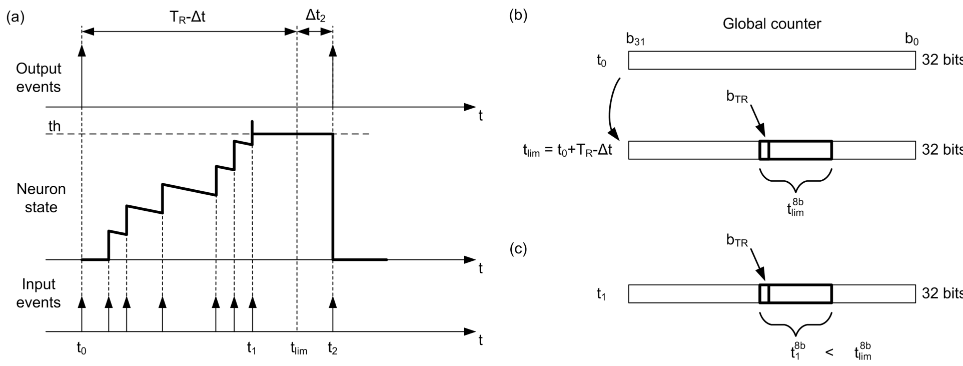

Fig. 2. Refractory period mechanism. (a) A pixel fires an event at $t = t_0$, so no output events are allowed until $t_{lim} = t_0 + T_R - \Delta t$. The pixel reaches the threshold at t_1 ($< t_{lim}$), so it keeps that state until t_2. (b) At $t = t_0$, future time t_{lim} is calculated using the value in the $32-bit$ global counter. Only 8 bits of t_{lim} are stored (t_{lim}^{8b}). (c) It reaches the threshold again at $t = t_1$. As $t_1 < t_{lim}$, no output event is allowed. For that, t_1^{8b} and t_{lim}^{8b} are compared.

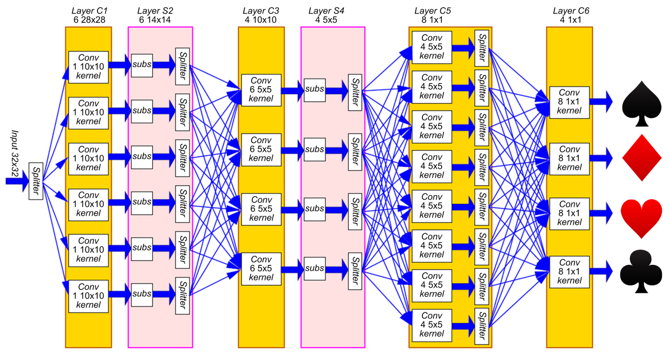

Fig. 3. Schematic block diagram of the Convolutional Neural Network used for poker card symbol recognition [8].

IV. ConvNet for recognition tasks

As described in Section II, the network node has been designed to assemble large 2D arrays in order to implement Convolutional Neural Networks (ConvNets). As an example, we implemented on FPGA the ConvNet described by Pérez-Carrasco et al. [8] for poker card symbol recognition (Fig. 3). It consists on 4 convolutional layers (C1, C3, C5 and C6 in the figure) and 2 subsampling layers (S2 and S4). We need 22 convolutional modules to implement the network: 6 modules with 28×28 pixels in layer C1, 4 modules with 10×10 pixels in layer C3, 8 modules with 1 pixel in layer C5 and 4 modules with 1 pixel in layer C6. In the present work, these modules are placed in an array of 6×4, programming the internal routers in each module to reproduce the network connectivity.

The network parameters were mapped from those given by Pérez-Carrasco [8] following a two-stage procedure. First, amplitude parameters (weights and thresholds) and time parameters (refractory periods and leakage rates) were adapted (scaled and rounded) to the hardware implementation, and second, they were tuned to compensate for hardware nonidealities by using a simulated annealing optimization algorithm.

V. Experimental results

The convolutional board and the whole network were tested using three boards: an AER data player [10] which receives a list of AER events through a USB port, a nodeboard with a Spartan6 FPGA where the network is implemented, and another AER board which receives the output AER events and sends them to a PC [10]. A micro-controller in the nodeboard receives the configuration parameters from a PC and sends them to the FPGA through an SPI interface.

A. Characterization of convolutional module

A single convolutional module was implemented on FPGA and different values of T_R were programmed covering the whole desired working range ($51.2ms$, $3.2ms$, $200\mu s$, $50\mu s$). For each case, a 1×1 kernel with value 1 and $th = 10$ were configured. The input stimulus was a train of events with fixed address and inter-spike interval following a normal distribution with mean $1/f_{in}$ and standard deviation $std_{in} = 10\%$ of the mean. Therefore, with no refractory limitation, the output average frequency would be $f_{out} = f_{in}/10$, with $std_{out} = std_{in}$. Fig. 4 shows f_{out} vs f_{in}, where each subplot corresponds to a different value of T_R. The error bars represent the standard deviation of the measured output frequencies. Having a closer look at Fig. 4(a), there is a saturation frequency $f_{sat} = 1/T_R = 19.53Hz$, so for values of $f_{in} < th\times f_{sat} = 195.3Hz$ there is a linear relationship, while larger input frequencies produce saturation. As T_R decreases in Fig. 4(b)-(d), the different subplots reproduce the same behavior, until the inter-spike interval becomes comparable with the global refresh pulse applied by the refractory period mechanism.

B. ConvNet characterization

The ConvNet described in Section IV was implemented and tested on FPGA. This network consists of 22 convolutional blocks distributed in 4 layers, with a total number of 5 116 neurons and 531 232 synapses, and consumed 93% of the available slices on the Spartan6 FPGA (21 465 out of 23 038). In order to characterize this network, a sequence of events was reproduced by an AER data player and sent to the FPGA. These events were previously recorded using a Dynamic Vision Sensor (DVS) [3] which was observing a deck of 40 poker cards running in 1 second. The recorded events were pre-processed to track the symbols and extract a 32×32 pixels window of the whole visual field showing only the centered

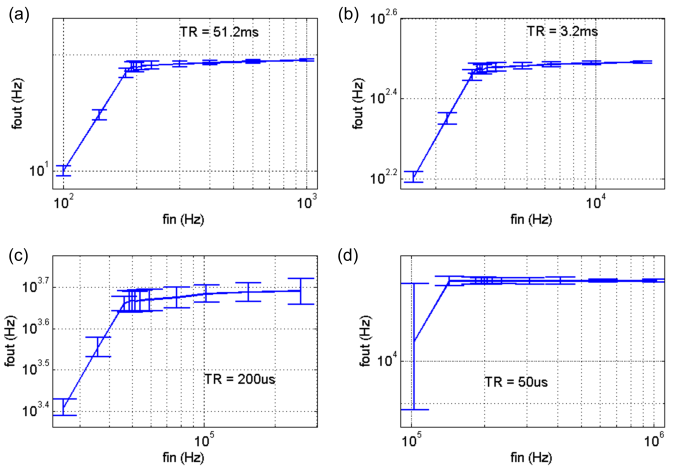

Fig. 4. Characterization of the refractory period for one single convolution pixel with $kernel = 1$ and $th = 10$. Each subplot corresponds to a different value of T_R: (a) $51.2ms$, (b) $3.2ms$, (c) $200\mu s$ and (d) $50\mu s$.

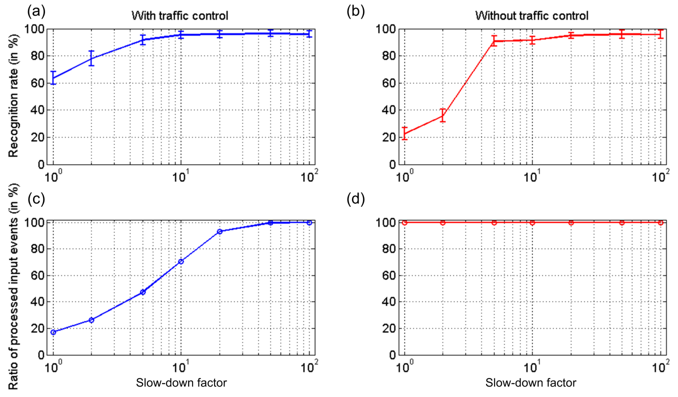

Fig. 5. Characterization of the network for poker card symbol recognition with different values of the slow-down factor. (a),(b) Recognition rate (in %), where blue trace includes traffic control mechanism, while the red trace does not. (c),(d) Proportion of processed input events (in %).

40 symbols. This stimulus consists of 174 644 events with an exact duration of $950ms$, which corresponds to an average event rate of $184Keps$ (events per second). When this stimulus is processed by this ConvNet, the total traffic registered inside the network is formed by 3 172 361 events, which corresponds to an event rate of $3.34Meps$. This event rate is higher than the capabilities of the network. In order to test the behavior of the network when processing this stimulus, different slow-down factors were applied to the input events (100, 50, 20, 10, 5, 2 and 1) to reduce the speed of the data. For each factor, the time constants of the network were also scaled proportionally. Fig. 5 shows the behavior of the network for poker card symbol recognition with each slow-down factor, illustrating the effect of the proposed traffic control mechanism.

For each time interval associated to one of the 40 symbols, the output events generated by the four neurons in the last layer were observed. Positive events indicate a symbol recognition, so we counted the positive events generated by each of these output neurons, obtaining n_s, n_h, n_d and n_c (number of positive events associated to spades, hearts, diamonds and clubs, respectively). For example, if $n_s > n_h, n_d, n_c$, we consider that the network recognized a spade. Following this criterion, we measured the recognition rate for each trial as the number of symbols recognized correctly over the total number of presentations. Fig. 5(a)-(b) shows a comparison between the recognition rates obtained for the implemented network with the proposed traffic control mechanism (blue trace) and those obtained for the network without traffic control (red trace). For slow-down factors larger or equal than 5, the recognition rates obtained for both networks are almost identical (above 90%), while larger event rates (factors 1 or 2) demonstrate the advantages of the proposed method, with recognition rates around 65% and 22%, respectively, when processing the real time recording. Fig. 5(c)-(d) shows the proportion of input events actually processed in each case. When there is no traffic control, all events are processed by the network (although they are delayed by the handshake protocol between different network modules, altering the spatio-temporal correlation of the events), as represented by the red trace. However, the proposed mechanism discards input events when any convolutional block is saturated, producing a reduction of the number of processed events as the event rate increases (blue trace). In the most conservative case (factor 100), the recognition rate is larger than 96%, with 100% of the input events actually processed. When the factor is 5, the proportion of processed events drops dramatically to around 45% while the recognition rate is still larger than 90%. Even when the recording is processed at real time, a recognition rate of around 65% is obtained with less than 20% of the input events.

The power consumed by the whole network was measured while processing the input sequence for different slow-down factors, obtaining $7.7mW$ at real time, and even lower consumptions for slower processing: $5.25mW$ when it was 10 times slower, and $0.85mW$ for a factor of 100.

VI. Conclusion

A new configurable event-based convolutional module with refractory period mechanism has been designed for hardware implementation of ConvNets on FPGA. This module was designed to assemble large 2D arrays, implementing a traffic control mechanism to discard input events when the network is busy, keeping spatio-temporal correlation and avoiding artificial delays. A 4-layer ConvNet has been implemented on a Spartan6 FPGA with more than $5K$ neurons and $500K$ synapses. Recognition rates around 96% were obtained for slow stimuli, 63% was obtained at high speed while less than 20% of the events were processed, demonstrating the robustness of the method even when the input stimulus is barely visible by a human observer. Arbitrary ConvNets can be easily implemented using the proposed module and methodology.

Acknowledgment

This work was partly funded by the EU Horizon 2020 Programme under grant no. 644096 (ECOMODE) and grant no. 687299 (NEURAM3), by Spanish research grant TEC2015-63884-C2-1-P (COGNET) (with support from the European Regional Development Fund), and by Andalusian research grant TIC-6091 (NANONEURO). L. A. Camuñas was funded by Spanish research fellowship “Juan de la Cierva”.

References

[1] P. Lichtsteiner, C. Posch, and T. Delbrück, “A 128x128 120dB 15μs latency asynchronous temporal contrast vision sensor,” IEEE Journal on Solid-State Circuits, vol. 43, no. 2, pp. 566-576, 2008.

[2] C. Posch, D. Matolin, D. 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 on Solid-State Circuits, vol. 46, no. 1, pp. 259-275, 2011.

[3] T. Serrano-Gotarredona and B. Linares-Barranco, “A 128x128 1.5% contrast sensitivity 0.9% FPN 3μs latency 4mW asynchronous frame-free dynamic vision sensor using transimpedance amplifiers,” IEEE Journal on Solid-State Circuits, vol. 48, no. 3, pp. 827-838, 2013.

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

[5] T. Delbrück, M. Lang, “Robotic goalie with 3ms reaction time at 4% CPU load using event-based dynamic vision sensor,” Frontiers in Neuroscience, 7:223, 2013.

[6] B. Zhao, R. Ding, S. Chen, B. Linares-Barranco and H. Tang, “Feedforward categorization on AER motion events using cortex-like features in a spiking neural network,” IEEE Transactions on Neural Networks and Learning Systems, vol. 26, no. 9, pp. 1963-1978, 2015.

[7] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard and L. D. Jackel, “Backpropagation applied to handwritten zip code recognition,” Neural Computation, vol. 1, no. 4, pp. 541-551, 1989.

[8] J. A. Pérez-Carrasco, B. Zhao, C. Serrano, B. Acha, T. Serrano-Gotarredona, S. Chen and B. Linares-Barranco, “Mapping from frame-driven to frame-free event-driven vision systems by low-rate rate-coding and coincidence processing. Application to feed-forward ConvNets,” IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 35, no. 11, pp. 2706-2719, November 2013.

[9] 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 Transactions on Biomedical Circuits and Systems, vol. 7, no. 1, pp. 82-102, February 2013.

[10] R. Serrano-Gotarredona, M. Oster, P. Lichtsteiner, A. Linares-Barranco, R. Paz-Vicente, F. Gómez-Rodríguez, L. Camuñas-Mesa, R. Berner, M. Rivas, T. Delbrück, S. C. Liu, R. Douglas, P. Häfliger, G. Jiménez-Moreno, A. Civit, T. Serrano-Gotarredona, A. Acosta-Jiménez, B. Linares-Barranco, “CAVIAR: A 45k-Neuron, 5M-Synapse, 12G-connects/sec AER Hardware Sensory-Processing-Learning-Actuating System for High Speed Visual Object Recognition and Tracking,” IEEE Transactions on Neural Networks, vol. 20, No. 9, pp. 1417-1438, September 2009.