On the AER Convolution Processors for FPGA

A. Linares-Barranco, R. Paz-Vicente, F. Gómez-Rodríguez, A. Jiménez, M. Rivas, G. Jiménez, A. Civit

Robotic and Technology of Computers group. University of Seville.
Av. Reina Mercedes s/n, 41012-Sevilla, SPAIN. alinares@atc.us.es

***Abstract*— Image convolution operations in digital computer systems are usually very expensive operations in terms of resource consumption (processor resources and processing time) for an efficient Real-Time application. In these scenarios the visual information is divided into frames and each one has to be completely processed before the next frame arrives in order to warranty the real-time. A spike-based philosophy for computing convolutions based on the neuro-inspired Address-Event-Representation (AER) is achieving high performances. In this paper we present two FPGA implementations of AER-based convolution processors for relatively small Xilinx FPGAs (Spartan-II 200 and Spartan-3 400), which process 64x64 images with 11x11 convolution kernels. The maximum equivalent operation rate that can be reached is 163.51 MOPS for 11x11 kernels, in a Xilinx Spartan 3 400 FPGA with a 50MHz clock. Formulations, hardware architecture, operation examples and performance comparison with frame-based convolution processors are presented and discussed.**

I. Introduction

Digital vision systems process sequences of frames from conventional video sources, like cameras. For performing complex object recognition algorithms, sequences of computational operations must be performed for each frame. The computational power and speed required makes it difficult to develop a real-time autonomous system. But brains perform powerful and fast vision processing using millions of small and slow cells working in parallel in a totally different way. Vision sensing and object recognition in brains is not processed frame by frame; it is processed in a continuous way, spike by spike, in the brain-cortex.

The visual cortex is composed by a set of layers ([1][2]), starting from the retina. The processing starts when the retina captures the information. In recent years significant progress has been made in the study of the processing implemented in the visual cortex. Many artificial systems that implement bio-inspired software models use biological-like processing that outperform more conventionally engineered machines [3][4]. However, these systems generally run at extremely low speeds because the models are implemented as software programs. For real-time solutions direct hardware implementations of these models are required. A growing number of research groups world-wide are implementing some of these computational principles onto real-time spiking hardware through the development and exploitation of the so-called AER (Address Event Representation) technology. AER was proposed by the Mead lab in 1991 [5] for communicating between neuromorphic chips with spikes. Each time a cell on a sender device generates a spike, it transmit a digital word representing a code or address for that pixel, using an external inter-chip digital bus (the AER bus). In the receiver the spikes are directed to the pixels whose code or address was on the bus. In this way, cells with the same address in the emitter and receiver chips are virtually connected by streams of spikes. Arbitration circuits ensure that cells do not access the bus simultaneously. Usually, these AER circuits are built using self-timed asynchronous logic [6]. Several works are already present in the literature regarding to visual processing filters. Serrano et al. presented chip-processor able to implement image convolution filters based on spikes that work at very high performance parameters compared to traditional digital frame-based convolution processors [15][16]. Another approach for solving frame-based convolutions with higher performances are the ConvNets [17][18], based on cellular neural networks, that are able to achieve a theoretical sustained 4 GOPS for 7x7 kernel sizes.

In this paper we present two FPGA implementations of neuro-cortical inspired convolution processors. These circuits have been developed studying a previous work for VLSI chip [12], but with several differences: (a) Instead of using the Integrate and Fire (IF) neuron, we have used RAM based integrators in one solution and any integrator in the second implementation; (b) instead of using arbitrated rate-code output we produce Poisson-like; (c) instead of 31x31 kernel sizes, we have limited to 11x11 because of FPGA resources limitations in one case and 3x3 in the second case; (d) instead of 32x32 expandable image sizes between chips, we have implemented 64x64 image sizes but no expandable.

II. Convolutions with spikes.

A. Description

Complex filtering processing based on AER convolution chips already exists. These chips are based on Integrate and Fire neurons [12]. Each time an event is received, a kernel of convolution is copied in the neighbourhood of the targeted IF neuron. When a neuron reaches its threshold, a spike is produced and the neuron is reset. Bi-dimensional image convolution is defined mathematically by the following equation, being K an nxm kernel matrix of the convolution, X the input image and Y the convolved image.

$$\underset{i,j}{\forall} \rightarrow Y(i,j) = \sum_{a=-n/2}^{n/2} \sum_{b=-m/2}^{m/2} K(a,b) \cdot X(a+i, b+j)$$

This work has been supported by Spanish government grants SAMANTA II (TEC2006-11730-C03-02) and VULCANO (TEC2009-10639-C04-02), and Andalussian regional government grant BrainSystems (P06-TIC-01417) .

Each convolved image pixel *Y(i,j)* is defined by the corresponding input pixel *X(i,j)* and weighted adjacent pixels, scaled by *K* coefficients. Therefore an input pixel *X(i,j)* contributes to the value of the output pixel *Y(i,j)* and their neighbours, multiplied by the corresponding kernel coefficients *K*.

For implementing convolutions using spikes let's suppose *Y* a matrix of integrators (capacitors for analog circuits or registers or RAM cells for digital circuits) to store the result of applying a kernel of convolution to an input image *X* that is coded into a stream of events through the AER protocol. Each pixel *X(i,j)* represents a gray value *G* in the original image. Let's suppose that in the AER bus will be represented, for a fixed period of time, an amount of events *P·G,* proportional to the gray level of the pixel. For each event coming from the continuous visual source (e.g. an AER retina or a synthetic AER generator), the neighbourhood of the corresponding pixel address in *Y* is modified by adding the convolution kernel *K*, stored in a RAM memory and previously configured. Thus, each element of *Y* is modified when an event with the address *(i,j)* arrives with the following equation:

$$Y(i+a, j+b) = Y(i+a, j+b) + K(a,b),\ \forall a,b$$

$$a \in \left[-\frac{N}{2}, \frac{N}{2}\right], b \in \left[-\frac{M}{2}, \frac{M}{2}\right], N, M = \dim(K)$$

Once all the events of the 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)* $\forall a,b$, (the gray values) times the value of the kernel, so the multiplication operation has been reached. The output of the convolution operation, at this point is stored in the matrix of integrators *Y*. This result can be sent out in several ways: (a) scanning all the integrators values and sending out an analog or digital signal. Each integrator or register is reset after reading. The system is converted into a frame-based output, so the neuro-inspired nature is lost. (b) Based on IF neuron, when an integrator reaches a threshold a spike is produced and the corresponding AER event is produced. The system is totally spike-based, but the output cannot follow a Poisson distribution due to the IF neuron [8]. Every time a spike is produced by a neuron, this is reset. This solution is used by analog convolution chips [13]. (c) Generate synthetically a stream of spikes. Having the result in the Y matrix, a method for synthetic AER generation can be used to warranty the Poisson distribution of spikes [10].

In [12] the convolution chip is able to receive positive and negative events to process signed kernels and therefore, it is also able to produce signed output events. This is done by duplicating the number of integrators, having a positive and a negative integrator per cell. If the positive integrator reaches the zero and additional negative values arrive to the cell, the negative integrator starts to work. The output event produced will be signed depending on the integrator used to produce that event.

B. FPGA Implementation I: Integrators on RAM

Figure 1 shows the block diagram of the first architecture. It is composed basically by two devices: a FPGA and a microcontroller. The FPGA is in charge of the AER-based convolution processor and the microcontroller is responsible of the PC interface for configuring all the parameters (Kernel matrix, kernel size, forgetting period and forgetting quantity). The circuit in the FPGA can be divided into four parallel blocks:

- ***AER event reception and processing***. The AER input traffic is managed by the "AER input" block. Each received event *(i,j)* is used to address the *Y* matrix (64x64 cells). Centered on *(i,j)*, the kernel is copied in the *Y* matrix. The neighborhood of the cell *(i,j)* will modify their values by adding the corresponding kernel value. Therefore, *Y* matrix is always up-to-date with the convolution result. *Y* matrix is implemented using a block of dual-port 8-bit RAM in the FPGA. The 11x11kernel is stored in another 8-bit block of one-port RAM. Each kernel value is in the range of -127 to 127. When the kernel is added to *Y* the result is limited between 0 and 255, so no signed events are implemented.
- ***Forgetting mechanism***. For high AER input bandwidth and / or weighted kernels, maximum cell value (255) could be reached quickly and thus, next events are not processed. Let's call this situation saturation effect. For this reason, to avoid errors, a configurable forgetting circuitry is in the architecture. The forgetting is based on a programmable counter that accesses the *Y* matrix periodically (let's call forgetting period) in order to decrease its values by a constant (let's call forgetting quantity). Forgetting period and quantity are configured through USB.

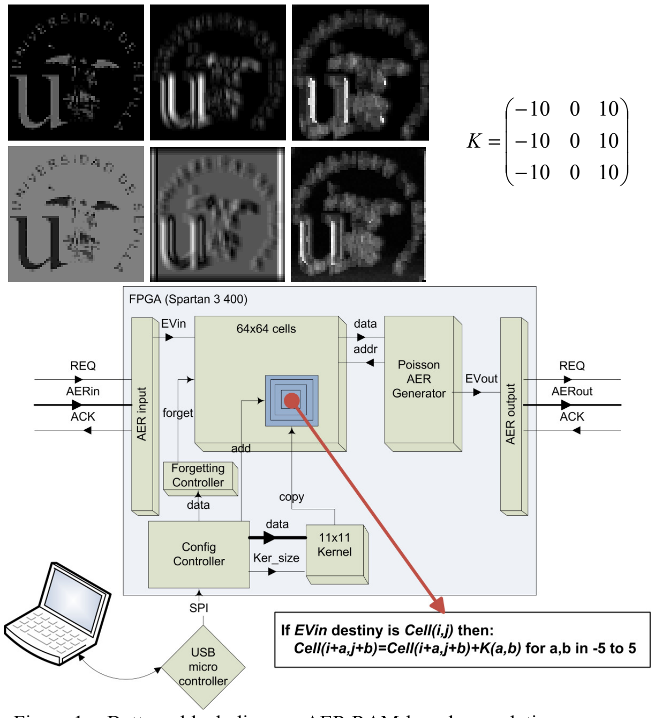

Figure 1. Bottom: block diagram AER RAM-based convolution processor. Top: source images (left column), MATLAB simulation results (center column) and FPGA AER convolution output histograms (rigth column).

- ***Poisson like AER output***. *Y* matrix always has the result of the convolution. Thanks to the forgetting mechanism, this matrix can be captured at any time with a valid convolved result. To warranty the Poisson distribution of output events, the *Y* matrix is accessed by a Random AER synthetic generator [8][9].

- ***Configuration***. The controller is in charge of receiving kernel size and values, forgetting period and amount to forget. An SPI interface connects a USB microcontroller and the FPGA. A computer with MATLAB controls the system.

The system has been developed in hardware in a Spartan 3 400 FPGA. The testing scenario consists on a USB-AER working as Uniform AER generator [9], which output is connected to an AER-Robot configured as the AER convolution processor; and its output is connected to a second USB-AER working as an AER datalogger.

Figure 1 (top, left column) shows an example bitmap and its negative version. These bitmaps are converted into AER using an AER Uniform generator to feed the convolution processor. When applying a kernel for edge detection (ON to OFF, see figure 2, matrix K), opposite responses are expected. Bottom row shows the reconstructed normalized histograms for 512Kevents obtained with the AER datalogger [11], from convolution processor output. The middle column images are the result of applying a MATLAB *conv2()* function.

III. Spike-based convolutions by Complex Mappings.

A. Description.

In this section we explain how to implement convolutions using a probabilistic multi-event mapper [10][14]. This mapper is able to send several different output events per each input event and each output event can be repeated several times. Let's suppose that *M* is the number of elements of a convolution kernel *K*. Each of these mapped events has associated a repetition factor *(R)* and a probability *(P)* of being sent or not. For each input event *(Ini)*, up to *dim(K)* output events are generated. Thus a projective field of events is generated for each input event. From the receiver point of view, the number of events for the same address depends on the probability and repetition factor of the input events in the neighbourhood. This neighbourhood is as big as the kernel size. Therefore, the number of events for a fixed output address follows the expression:

$$Nev\_Out_i = \sum_{a,b} In_{a,b} \cdot R_{a,b} \cdot P_{a,b} = \sum_{a,b} In_{a,b} \cdot K_{a,b} \Rightarrow$$
$$K_{a,b} = R_{a,b} \cdot P_{a,b} \quad \forall a,b \in \dim(K)$$

Therefore, to calculate the repetition and probability, the following equation can be applied:

$$R_i = \lceil K_i \rceil \; ; and \; P_i = {K_i}/{R_i}$$

For example, for a desired kernel coefficient Ki=1.2, repetition factor is *Ri=2* and probability is *Pi=0.6*.

B. Implementation II: Probabilistic Mapper.

An AER mapper uses the identification of each emitter neuron that is present in the AER bus to address its mapping table. Then, a list of mapped addresses is sent replacing the original event in the AER communication. Each mapped event is stored in the mapping table with two parameters: a repetitious factor *(R)* and an output probability *(P)*. A FSM is in charge of sending the list of events if the probability function allows it. This FSM will repeat each mapped event according to *R* (see figure 2, top). An internal LFSR (Left Feedback Shift Register) is used to generate pseudorandom numbers which are compared with the probability in order to decide if the mapped event is sent. In this way a Poisson output distribution is obtained [8][9].

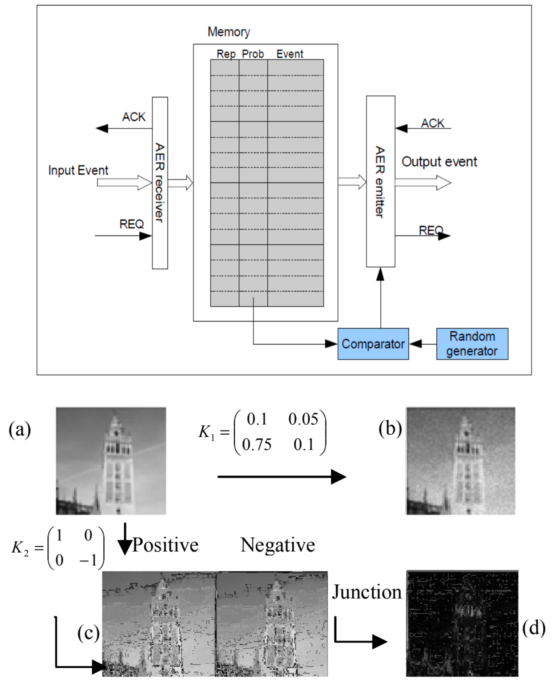

Figure 2. Top: Probabilistic Multi-event mapper block diagram. Right: (a) source image, (b) K1 convolution histogram, (c) K2 postive output traffic and negative traffic, (d) abs(positve+negative) K2 histogram.

With this mapper, sophisticated operations can be performed to events during transmission time, like small kernel convolutions, for example. As the probability can be modified per each mapped event, it is possible to implement a *one-to-dim(K)* mapping, where each of these *dim(K)* events can be modulated with *P* and *R* along time, as expressed in the previous section. Thus, this architecture is able to increase or reduce the frequency of events of a neighbourhood per each input event according to a kernel of convolutions. In the other hand, mapped events can be positive and negative, so signed events appears in the output AER bus. A receiver must take the average between positive and negative events per each address for completing the convolution operation.

This architecture has been tested in a USB-AER board that is composed by a Spartan II FPGA and 2Mb of SRAM memory (mapping table) [10]. It is able to process a one-to-one mapping in *120ns*, and a one-to-M effective mapping in *(60+M·60)ns*. If a mapped event is not sent the time consumption decreases in *20ns*.

Figure 2 bottom shows two examples. A source bitmap (64x64) (a) is converted to AER format by the Uniform method [9]. (b) is the normalized histogram of collecting AER traffic for 100ms, after applying a simple 2x2 kernel convolution K1. The resulting image should be the initial one with soft-edges. An added random noise, due to Poisson distribution of mapped events, is introduced.

Images (c) and (d) belong to a second example, where an edge detection kernel (K2) is applied. The mapper sends a positive and a negative event per each input event with different addresses. Positive and negative (c) normalized histograms are shown. (d) image shows the absolute value of the combination of positive and negative traffic. We have used an up-down counter per each address.

IV. Comparison with digital frame-based convolution processors.

Digital frame-based convolution processors implemented in FPGA, GPUs or CPUs measure their performance by calculating the number of operations per second (MOPS). In this way, a frame-based convolution processor has to calculate the number of ADD and MULT operations done to apply a kernel of convolution to an input frame. Once all the operations are executed for the whole frame, a result frame is obtained and sent to another stage. In [15][16] a performance study was presented for different kernel sizes and platforms.

The FPGA implementation explained in section II is accessing sequentially all the kernel elements and adding their values to the corresponding cells. As cells are stored in RAM memory and this is read cell by cell, time required per each input event grows linearly with the kernel size. In this case the hardware implements only one ADD unit. The system consumes one cycle for reading the cell value and the corresponding kernel position, a second cycle is used for the add operation and for writing the result in the internal RAM. Thus for a 3x3 kernel we need 3x3x2 cycles per each input event, apart from those needed for the handshake. The architecture can be easily improved by having a number of ADD units equivalent to the kernel row size, and allowing the access to the RAM by blocks. Table I shows the real performance in MOPS for one ADD, and the simulated performance for the N ADD in parallel of the NxN kernel convolution processor.

TABLE I. MEGA OPERATIONS PER SECOND FOR FRAME AND SPIKE-BASED CONVOLUTION PROCESSORS.

MOPS	Digital frame based 2D convolutions [15][16]				Spike-based 2D Convolutions			
					By mapping	By RAM integrators		Analog [12]
Kernel size (NxN)	Pent 4 3GHz CPU	6800 ultra GPU	Spartan 3	Virtex II-Pro	Spartan II 100 MHz	Spartan 3 (real) 1 sum 50MHz	Spartan 3 (sim) N sum 50 MHz	VLSI 0,35um 200 MHz
2x2	14	1070	190	221	18,2	-	-	-
3x3	9,7	278	139	202	22,5	20,5	34,61	-
5x5	5,1	54	112	162	-	23,1	65,78	-
7x7	2,6	22	90	110	-	24,0	98,00	-
9x9	1,6	9	73	61	-	24,4	130,0	-
11x11	1,2	4,7	23	48	-	24,6	163,5	-
31x31								2910

A spike-based convolution processor based on the probabilistic mapper is not calculating neither MULT nor ADD operations. It works by projecting the input traffic following a 1 to N mapping in a first stage, and then by cancelling negative and positive events of the same address in a short period of time. In this way the equivalent number of operations is the number of mappings per each input event.

V. Conclusions

Two neuro-cortical layers of 64x64 cells with NxN convolution kernel hardware implementation on FPGA have been presented. They can work with kernels of up to 11x11 (~500Ksynapses), with 8-bit integrator cells, or with 3x3 kernels (~37Ksynapses), with no integrator cells. The AER output of both follows a Poisson distribution. Both are compared with a VLSI analog implementation and tested for image edge detection filtering. Poisson output distribution of events and low cost are the main differences and strongest contributions of this approach. Digital frame-based architectures are limited by the number of parallel ADD units and MULT units so their performances decrease while their kernel sizes increase. Digital RAM-cell spike-based architecture is limited by RAM access. When parallelizing the number of ADD units, the performance increases with the kernel size. Spike-based mapping avoids ADD units since the operations consist in projecting inputs events with weighted and signed events and then averaging them in short periods of time. Thus, its performance depends on SRAM access time. Kernel sizes are limited by the mapping table capacity.

References

[1] T. Serre. “Learning a Dictionary of Shape-Components in Visual Cortex:Comparison with Neurons, Humans and Machines”, PhD dissertation, MIT. Comp. Sci. & AI Lab Technical Report, 2006.

[2] G. Shepherd, “The Synaptic Organization of the Brain”. Oxford University Press, 3rd Edition, 1990.

[3] S. Thorpe et al, “Speed of processing in the human visual system”. Nature, 381, 520 - 522, June 1996.

[4] C. Neubauer. “Evaluation of Convolution Neural Networks for Visual Recognition,” IEEE Trans. on Neural Networks, vol. 9, No. 4, pp. 685-696, July 1998.

[5] M. Sivilotti. “Wiring Considerations in analog VLSI Systems with Application to Field-Programmable Networks”, Ph.D. Thesis, California Institute of Technology, Pasadena CA, 1991.

[6] K. Boahen. “Communicating Neuronal Ensembles between Neuromorphic Chips”. Neuromorphic Systems. Kluwer Academic Publishers, Boston 1998.

[7] W.R. Softky, The highly irregular firing of cortical cells is inconsistent with temporal integration of random EPSPs, J Neurosci. 13 (1) (1993)

[8] A. Linares-Barranco, et al, “On Algorithmic Rate-Coded AER Generation,” IEEE Trans. On Neural Networks, vol. 17, 2006.

[9] A. Linares-Barranco, et al, “Inter-Spike-Intervals analysis of AER-Poisson-like generator hardware,” Neurocomputing, vol. 70, 2007.

[10] R. Paz et al. “Test Infrastructure for Address-Event-Representation Communications”. IWANN 2005. LNCS 3512. Springer Verlag.

[11] A. Linares-Barranco, et al. “An AER-based Actuator Interface for Controlling an Antrophomorphic Robotic Hand”. IWINAC-2007.

[12] R. Serrano-Gotarredona et al. “A Neuromorphic Cortical-Layer Microchip for Spike-Based Event Processing Vision Systems”. IEEE Trans. on Circuits and Systems I. Vol 53, No 12, Dec. 2006.

[13] T. Serrano-Gotarredona , et al. “AER image filtering architecture for vision processing systems”, IEEE Trans.Circuits and Systems (Part II): Analog and Digital Signal Processing, vol. 46, 1999.

[14] A. Linares-Barranco et al. “Implementation of a time-warping AER mapper”. ISCAS 2009. Taiwan.

[15] Ben Cope et al. “Implementation of 2D Convolution on FPGA, GPU and CPU”. Imperial College Report.

[16] B. Cope, et al. “Have GPUs made FPGAs redundant in the field of video processing?”.FPT 2005.

[17] C. Farabet, C. Poulet, J. Y. Han, Y. LeCun. “CNP:: An FPGA-based Processor for Convolutional Networks”. International Conference on Field Programmable Logic and Applications, 2009. FPL 2009.

[18] N. Farriga, F. Mamalet, S. Roux, F. Yang, M. Paindavoine. “Design of a Real-Time Face Detection Parallel Architecture Using High-Level Synthesis”. Hindawi Publishing Corporation. EURASIP Journal on Embedded Systems. Vol. 2008, id 938256, doi:10.1155/2008/938256