Visual Spike-based Convolution Processing with a Cellular Automata Architecture

M. Rivas-Pérez, A. Linares-Barranco, J. Cerdá, N. Ferrando, G. Jiménez, A. Civit

***Abstract*—this paper presents a first approach for implementations which fuse the Address-Event-Representation (AER) processing with the Cellular Automata using FPGA and AER-tools. This new strategy applies spike-based convolution filters inspired by Cellular Automata for AER vision processing. Spike-based systems are neuro-inspired circuits implementations traditionally used for sensory systems or sensor signal processing. AER is a neuromorphic communication protocol for transferring asynchronous events between VLSI spike-based chips. These neuro-inspired implementations allow developing complex, multilayer, multichip neuromorphic systems and have been used to design sensor chips, such as retinas and cochlea, processing chips, e.g. filters, and learning chips. Furthermore, Cellular Automata is a bio-inspired processing model for problem solving. This approach divides the processing synchronous cells which change their states at the same time in order to get the solution.**

I. INTRODUCTION

CELULLAR organization in biology has been an inspiration in several fields, such as the description and definition of Cellular Automata (CA). They are discrete models that consist of a regular grid of cells. Each cell has an internal state which changes into discrete steps and knows just one simple way to calculate the new internal state like a rudimentary automaton. Cellular activity is carried out simultaneously like it occurs in biology. Von Neumann refers to this system as a Cellular Space and is known currently as Cellular Automata [1].

The first self-reproducing CA, proposed by von Neumann consisted of a 2D grid of cells, and the self-reproducing structure was composed of several hundreds of elemental cells. Each cell presented 29 possible states [2]. The evolution rule was defined as a function of current state of the cell and its neighbours (up, down, right and left). Due to the high complexity of the model, von Neumann rule has never been implemented in hardware, but some partial implementations have been obtained [3].

Address-Event- Representation (AER) is a spike-based representation technique for communicating asynchronous spikes between layers of different chips. The spikes in AER are carried as addresses of neurons (called events) on a digital bus. This bio-inspired approach was proposed by the Mead lab in 1991 [4].

There is a world-wide community of AER protocol engineers and researchers for bio-inspired applications in vision and audition systems and robot control, as it is demonstrated by the success in the last years of the AER group at the Neuromorphic Engineering Workshop series [5]. The goal of this community is to build large multi-chip and multi-layer hierarchically structured systems capable of performing massively-parallel data-driven processing in real time [6].

One of the first processing layers in the cortex consists of applying different kinds of convolution filters with different orientations and kernel sizes. Complex filtering processing based on AER convolution chips have been already implemented, which are based on Integrate and Fire (IF) neurons [7]. When an event is received, a convolution kernel is copied in the neighbourhood of the targeted 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 convolution kernel matrix, X the input image and Y the convolved image.

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

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.

Digital frame-based convolution processors implemented in FPGA or CPUs usually measure their performance by calculating the number of operations per second (MOPS). A comparative study between frame-based and spike-based convolution processors was presented in [8]. A frame-based 3x3 kernel convolution processor in a Spartan-III FPGA yielded 139 MOPS, whereas spike-based one yielded 34.61 MOPS for the same kernel. Nevertheless, frame-based 11x11 kernel convolution processors decreased their performance to 23 MOPS, while the spike-based processors increased their performance to 163.51 MOPS. Therefore, spike-based convolution processors may achieve higher performances for the same hardware availability. This has to be thanked to the fully parallel processing allowed by AER or spike-based processing.

This work was supported by the Spanish grants SAMANTA II (TEC2006-11730-C03-02) and VULCANO (TEC2009-10639-C04-02), and by the Andalusia Council grants BrainSystems (P06-TIC-01417).

M. Rivas-Pérez, A. Linares-Barranco, G. Jiménez, and A. Civit are with the Dept. of Computer Architecture and Technology, University of Seville, Seville, SPAIN (e-mail: mrivas@atc.us.es).

J.Cerdá and N. Ferrando are with the Dept. of Electronic Engineering Tech University of Valencia, Valencia, SPAIN.

Another approach for solving frame-based convolutions with higher performances is the ConvNets [7] [9]. They are based on cellular neural networks and can achieve theoretical sustained 4 GOPS for 7x7 kernel sizes.

This paper presents two implementations of AER spike-based convolution processor for 3x3 kernel sizes using architectures inspired by Cellular Automata. These processors have been described into VHDL and implemented for a Spartan II 200 Xilinx FPGA with a 50 MHz clock. A performance analysis has been carried out by USB-AER tools [10]. This processor can yield up to 150 MOPS for 3x3 kernel sizes, which implies a performance of up to 2 GOPS for a possible 11x11 kernel implementation.

Next section introduces and describes CA, AER and how they can work together. Section III presents architectures, results and future works about AER image filtering implementations based on CA. Finally the conclusions are presented in section IV.

II. AER PROCESSING BASED ON CELLULAR AUTOMATA

A. *Cellular Automata*

A Cellular Automata consists of a regular 2D array of cells. The state of each cell is defined by a set of bits and varies longitudinally according to an evolution rule. This evolution rule should be the same for all the cells and it is a function of the current internal state of the cell and its neighbourhood [1], so it does not depend on external stimulus. These neighbours are a fixed set of cells adjacent to the specified cell. A new generation is created every time the rule is applied to the whole grid. A global clock signal sets when the state of the cell is updated.

B. *AER for spike-based systems*

Address-Event-Representation (AER) is a communication protocol for transferring asynchronous events between VLSI neuro-inspired chips, originally developed for bio-inspired processing systems [4]. Every time a neuron generates a pike, a digital word (called event), which identifies the neuron, is placed on an external AER bus. A receiver chip decodes the received event and sends a spike to the corresponding neuron. This way each neuron from a sender chip is virtually connected to a corresponding neuron in the receiver chip through a single time division multiplexed bus. These neurons carry out an internal processing for every arriving spike and can produce an output spike or stream of spikes in response. The most active neurons access the bus more frequently than less active ones. An arbitration circuit ensures that neurons do not access the bus simultaneously. This AER circuit is usually built using self-timed asynchronous logic as it is discussed in [11].

AER chips develop hierarchical systems composed by layers of neurons like a brain. Results of one layer represent the input of the next layer or a feedback of a previous one. Furthermore, like in a biological neural system, several AER devices such as visual sensors (retina [12]), audio sensors (cochlea), filters and learning chips have been developed, as well as a set of glue tools (AER tools) which facilitate developing and debugging of these spike-based multi-layer hierarchical systems. For example, a synthetic AER generator is a tool that reproduces any test bench stimulus for debugging purposes, thus it is able to transform a sequence of static frames into AER stream of spikes [13].

Transmitting event addresses through the AER bus allows performing additional operations on the events while they are travelling from one chip to another. For example, the output of an AER retina can be easily translated, scaled, or rotated by simple mapping operations on the emitted addresses. These mapping can either be lookup-based (using, e.g. an EEPROM) or algorithmic. Furthermore, the events that a chip transmits can be received by many receiver chips in parallel, by handling the asynchronous communication protocol properly.

The AER information transmitted by a visual AER sensor or a synthetic AER generator is usually coded in gray, i.e. the number of events (in other words, the frequency of spikes) transmitted by a pixel through the AER bus identifies the gray level of that pixel or the intensity of the pre-processed result.

C. *Cellular Automata for AER processing*

The philosophy of AER systems is lightly different from CA but also similar in a certain sense. A CA is a cooperative system, whose evolution only depends on the input and its neighbours, but an AER system usually does not implement evolution rules for producing new output spikes, i.e. it only produces spikes when a stimulus arrives. Nevertheless, they cooperate with the neighbourhood to produce a response. Spike-based convolution processor cells communicate with their neighbours somehow for the kernel processing, but the results are produced only when a stimulus arrives.

A spike-based convolution processor inspired in CA is suggested to filter a visual stimulus for edge detection (detecting vertical and/or horizontal contrast changes) using a non-evolutionary process. The output of this edge detection produced by a set of cells may be communicated in an evolution way for detecting predefined objects. For example, in a spike-base convolution processor of 3x3 kernels configured to detect edges and stimulated by spikes from a diamond image, only the edge cells of the diamond may produce events. This represents the convolution output. But if each output spike is sent to the neighbours, there will be cells receiving spikes that represent the edge detection from their neighbours at the same time. If these spikes are transmitted through the neighbourhood, at the end the pixel in the centre of an object will receive edge detection information from several sides at the same time, what allow identifying the centre of a diamond or a square or a circle, for example. There will be a junction cell for these spikes

that represents the centre of the object. In this case, the 2D array of cell may be implementing an edge detector using the 3x3 kernel convolution processor, but it may also have the same cells implementing a next processing layer, like an object centre detector.

This paper discusses a start point of this theory by implementing convolutions using a 2D array of cells that communicates with a predefined neighbourhood.

D. AER Image filtering strategy

When an event arrives from an emitter chip, its address is decoded and a spike is sent to the corresponding cell. For each spike received, several operations are arranged to transform the target cell and the neighbourhood. These are summarized below:

- The cell which received a spike resends it to its neighbours and updates its state with an increment of the kernel centre.
- The state of each neighbour is incremented by the corresponding kernel element, which depends on where the spike comes from, i.e., each spike received brings on several increment operations in the kernel size neighbourhood of the target cell. When the specific cell and its neighbours are updated, an acknowledge signal is sent to the AER emitter.
- When the state of a cell achieves the threshold, this cell generates an event and resets its state. The threshold value is constant in the whole grid. This behaviour corresponds to the Integrate and Fire neuron model.

The number of events that a cell receives depends on the gray level of the corresponding pixel, so the specific cell and its neighbours are incremented as many times as the gray level, implementing the multiplication of the convolution operation.

III. IMPLEMENTATION ON FPGA

A. USB-AER board

The USB-AER board includes a relatively large FPGA (Spartan-II 200) that can be loaded from MMC/SD or USB (through the C8051F320 microcontroller), a large SRAM bank (512Kx32 12ns) and two AER ports [10]. An input AER bus and an output AER bus connected directly to the FPGA allows implementing any hardware for manipulating or processing AER information.

The USB-AER tool has several functionalities according to the module that is loaded in the FPGA (through MMC/SD or USB). For example, it may act as a sequencer, monitor, mapping, event processor, datalogger, etc. Most of such functionalities can be performed in a standalone manner. This standalone operating mode requires to load the FPGA and the mapping RAM from some type of non-volatile storage so that it can be easily modified by the users, e.g. MMC/SD cards. USB input is also provided for development stages. Due to the bandwidth limitations of full speed USB (12Mbit/s), a based-event to frame conversion is essential in this board for high or even moderate event rates [10].

The present work increases these functionalities by including two versions of filters based on CA. The first version comes from the CA concept: hardware implements independent cells that evolve and communicate with their neighbours. The second version optimizes resources by exploiting a quirk of the AER-CA: only the targeted cell and its neighbours work when a spike arrives, i.e., this spike only affects that neighbourhood. The neighbourhood size is the same as the convolution kernel size. Therefore, it may be possible to keep the results of the whole 2D array in memory and to implement only a shared set of cells that perform the

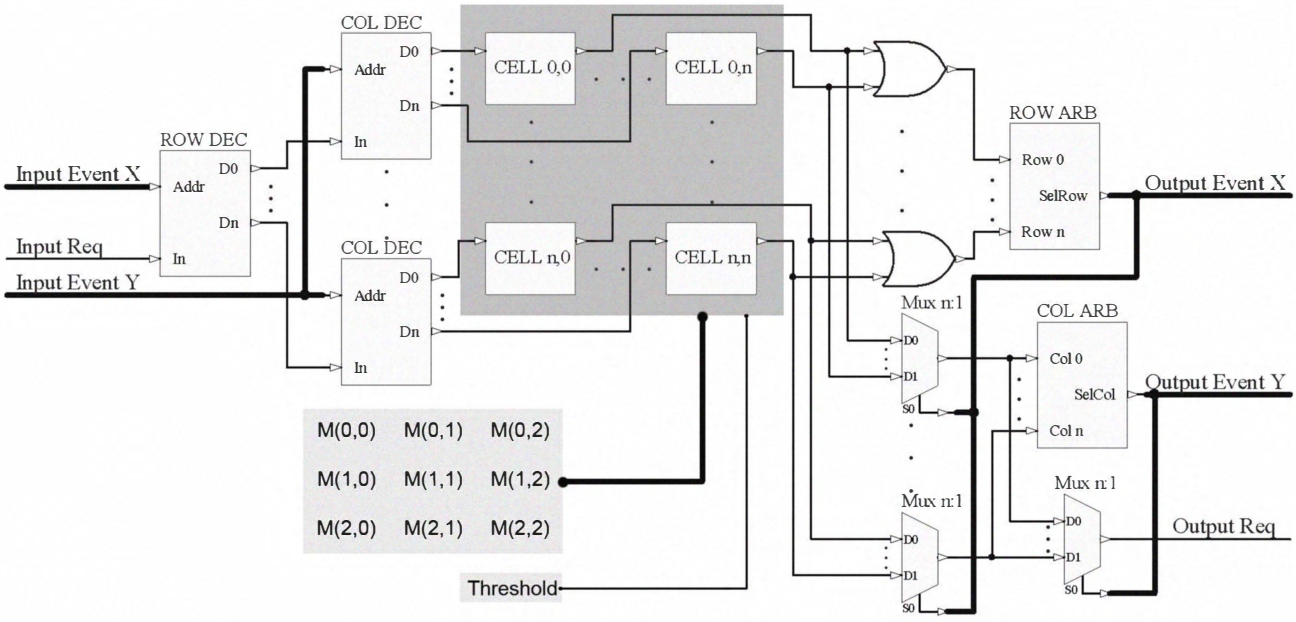

Fig. 1. Block diagram of CA through distributed computational units

operations needed to calculate the result of the convolution when a new spike arrives. Both versions are described below.

Next sub-section presents the implementation of the whole 2D array, and sub-section C describes a resource sharing implementation using distributed memory to store the 2D processing result.

B. Implementation based on distributed computational units

A 3x3 kernel implementation has been described into VHDL for Spartan-II 200. Larger kernel sizes require FPGA with more resources than Spartan-II. Each cell in a 3x3 kernel implementation is connected to eight neighbours.

Figure 1 illustrates the main diagram of this implementation. AER input is connected to the grid through two-level decoders. When an event comes, e.g. from a vision sensor, its address is divided into row and column. The first level decodes the row by *Input Event X* and the second level decodes the column by *Input Event Y* in order to select the targeted cell. *Input Req* signal notifies when the new event arrived.

Each cell is also connected to a two-level arbiter. The first level consists of a row arbiter and OR gates to encode the output event row address (*Output Event X*). The second level is composed of a column arbiter and an array of multiplexors. This array selects a row from the grid and it is controlled by the first level. The column arbiter encodes output event column address (*Output Event Y*) from the array of multiplexer. The array of multiplexors and a multiplexor connects the cell to the output. 3x3 kernel elements and threshold are available for all cells of the grid as figure 1 shows. Each cell is connected to its eight neighbours through a single wire, only request signal, to minimize the number of connections.

Figure 2 shows the digital logic of each cell in the grid. A cell consists basically of two multiplexers to select the kernel coefficient to add (*Mux8:1* and *Mux2:1*), an adder *ADD8* to update the cell internal state, a register *FD8CE* to save the state, a comparator *COMP* to calculate when the cell must fire, D Flip-Flop *FDSR* to communicate the cell with the arbiters by handshaking process and some logic gates to control the overflow. The lower overflow is also controlled because some kernel coefficients may be negative.

When receiving a spike, this cell increment its internal state according to kernel centre by *ADD8* and sends a spike to its eight neighbours simultaneously. Each one of these neighbours adds a kernel coefficient to its internal state depending on where the spike arrives, e.g. when a spike comes from the bottom right cell, it adds the bottom right coefficient of the kernel to its internal state. That is, the cell that receives the spike and its eight neighbours (nine cells in total) modify their states by the corresponding kernel element. This implies a restriction in the system that makes the kernel sizes to be small for a Spartan II 200 FPGA.

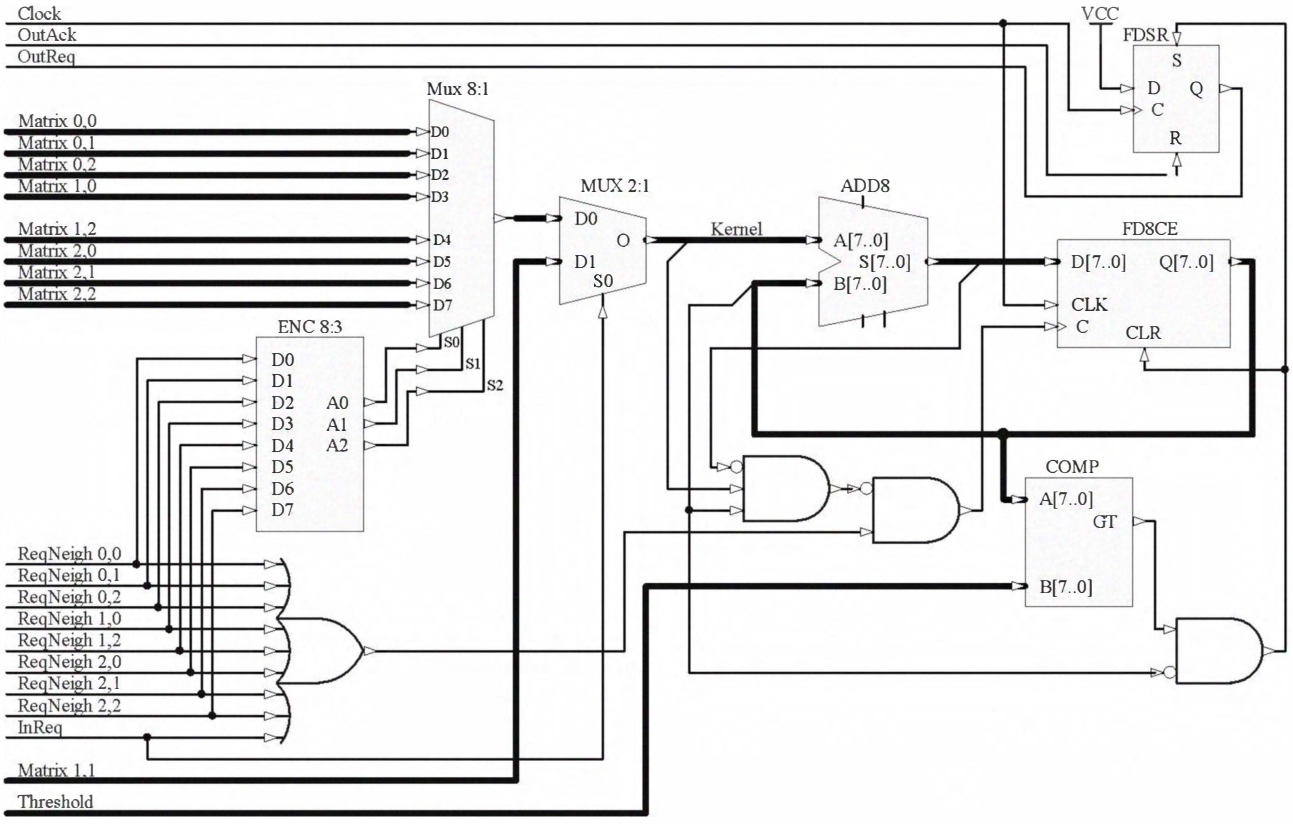

Fig. 2. Block diagram of a cell

When the state of a cell reaches the threshold, the comparator *COMP* in the figure 2 resets its state and saves a request in the *FDSR*. The output arbiter processes the fired cells by a fixed priority. This arbiter generates the output event address according to the cell attended. When this output event is acknowledged, the arbiter clears the request stored in the *FDSR* of the cell.

When all fired cells are attended, an acknowledge signal is sent to the emitter AER chip and therefore processing of the incoming event concludes. In this way, no new input events can arrive before the current event has finished. This constrain simplifies this design because:

- An Input FIFO is not needed to store all new events that arrive while the current event is being processed.
- Output arbiter may be fixed priority because there is no risk that lower priority cells starve and a fixed priority arbiter is the simplest implementation.
- Each cell never has more than one pending request, so a single FDSR is needed and then less resource are required.

C. *Implementation based on distributed memory*

A design based on distributed computational unit wastes many resources (see results bellow) because each cell is implemented by a different computational unit; but only nine cells of the whole grid work simultaneously, i.e. the selected cell and its eight neighbours, so only a few resources are used at the same time.

This section describes a new implementation based on nine shareable computational units. Each unit always works with the same kernel element but different cells. In the previous design each computational unit stores a cell state, but in this new implementation all states are saved in RAM memory because there is not one unit per cell. RAM memory is divided into nine banks so that nine computational units can access one memory bank at the same time. These states are distributed to nine RAM memory banks in a way that each computational unit accesses a different RAM bank simultaneously every time an event arrives.

Each computational unit always uses the same kernel element and modifies the state of same neighbour relative to the target cell, for example, the upper left unit always works with the upper left kernel element and the upper left neighbour of the targeted cell. The left side of the figure 3 shows an example about how it works. Each location in the grid indentifies a cell and it is labelled by *Bx* where *x* is the number of bank associated to that cell. When the cell *C(3, 2)* receives an spike, a computational unit uses the kernel element *K(1,1)* and the bank *B2* to modify the state of cell *C(3, 2)*, another one uses the kernel element *K(0,0)* and the bank *B7* to modify the state of neighbour (2,1), and so on up to nine units, but each unit accesses a different bank.

When each cell stores its state, the memory address and the memory bank are calculated as the following:

- Event address is divided into row and column coordinates.
- Each coordinate is incremented and decremented in order to obtain the address of nine neighbours as a couple of coordinates. For example, cell address *(3,8)* generates eight addresses *(2,7), (2,8), (2,9), (3,7), (3,9), (4,7), (4,8)* and *(4,9)*.
- Every coordinate (original, incremented and decremented coordinate for row and column) is divided in 3. In the above example, row coordinates are 2, 3 and 4; and column coordinates are 7, 8 and 9.
- The quotient of division denotes the memory address. Six calculated quotients are combined in pairs, quotient from a row and column to get memory address of the nine cells.
- The reminder of division indicates the memory bank. The calculation process of memory banks is the same as memory addresses but a reminder is used.

A separate router connects automatically each computational unit to the corresponding cell state by using addresses and banks calculated so that units do not have to control RAM memory directly.

As in distributed computational unit implementation, every computational unit calculates the new state adding the corresponding kernel coefficient to current state. When cell state achieves the threshold, resets its state and saves a request in its *FDSR* (figure 2) until the output fixed priority arbiter processes it. When the arbiter dispatches all requests, an input acknowledge signal is sent to sender and the system

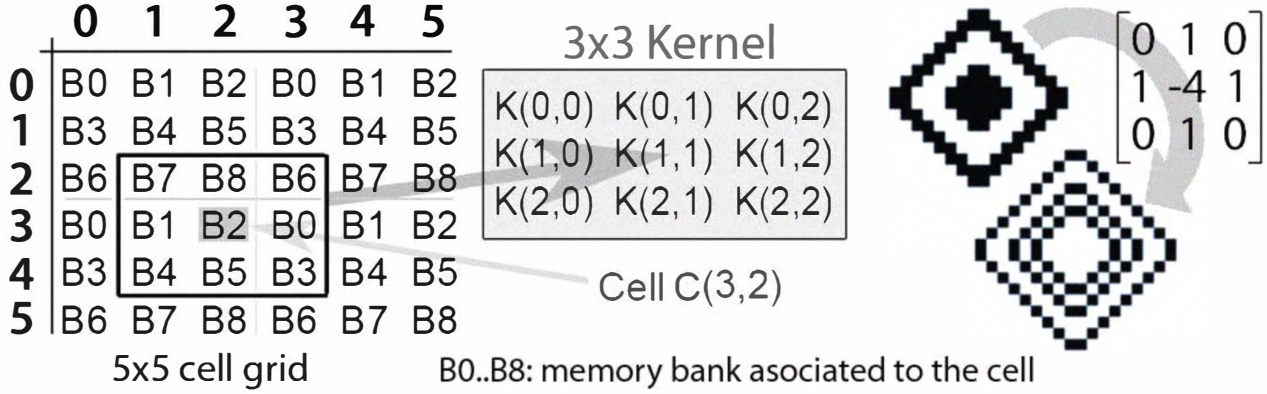

Fig. 3. Example of how memory banks are assigned

waits for a new event. Figure 3 at the right shows snapshots of how these implementations work when an edge detection kernel is used.

D. Testing Scenario

Three USB-AER boards have been connected in line in order to test and measure the performance of these processors.

A sequencer firmware is loaded in the first board to send a stream of events to the second one. This stream depends on the image previously loaded in the FPGA memory. A diamond image as figure 3 shows is uploaded in memory to test firmware.

The convolution processor to be tested is loaded in the second USB-AER board. The board receives spikes from the sequencer and carries out the filtering operation according to 3x3 kernel loaded. Filter parameters, such as kernel coefficients and threshold, are controlled from a computer application via USB. The right side of the figure 3 illustrates the original and resultant image when a 3x3 kernel is applied for extracting edges.

The third USB-AER board receives events from the convolution processor. A datalogger or framegrabber firmware can be loaded in this board. The datalogger firmware is able to store, in real-time, not only the address of the spikes produced by the AER-CA, but also the time when they are produced. Therefore, the spikes captured can be processed off-line by Matlab to test the processor and measure performance. The framegrabber firmware rebuilds the resultant image from the spikes received and sends this frame to a computer via USB to be displayed.

A windows application called Multi-LoadFPGA or Matlab controls and configures every USB-AER via USB without reloading the FPGA.

E. Result

The event rate achieved is determined not only by convolution processor delay but also by input and output event rate. The input event rate depends on the image loaded into the sequencer and the output event rate is related to the kernel coefficients and threshold.

AER-CA based on distributed computational units requires three clock cycles for every event received: a synchronization cycle, a cycle for edge detection and another one to calculate cell states. Furthermore, this version requires two cycles for every new event generated: one cycle to send an event and another one to wait for the acknowledgement. This implementation yields up to 16.6 mega-events per second when no event is generated. In addition, nine ADD operations are computed every three cycles, thus the system yields up to 150 MOPS when a 50 MHz clock is used.

This implementation requires many resources to implement the cell grid. The largest grid that may be loaded in a Spartan-II 200 is an 8x8 grid. This grid spends 84% of all resources and allows clock frequencies of up to 52 MHz. A 16x16 grid requires a FPGA three times larger.

AER-CA based on distributed memory requires six clock cycles per received event. Unlike the previous version, this firmware requires four cycles to calculate cell states: A cycle to calculate memory addresses and banks, a cycle to connect each unit to the appropriate bank, another one to read the current state and the last one to write the new state in memory. As the former implementation, this version also requires two cycles per event. It performs up to 8.3 mega-events per second and 75 MOPS.

This version saves more resources in comparison to the former one. This version allows grids of up to 32x32 in a Spartan-II. This grid spends just 28% of the internal logic and uses 9 of 14 memory banks. A larger grid requires adding four memory banks more because the number of banks must be multiple of nine at least.

F. Future Improvements

Both proposed versions require saving the state of every cell to calculate new states from the previous one, but it wastes many resources. A probabilistic version may be developed, which cells fire depending on a probabilistic matrix, to avoid storing cell states. Under this new vision, a random number and the corresponding probability coefficient determine when cells fire instead of adding kernel coefficients until threshold is achieved. This new version may increase processing speed and may work with larger grids in the Spartan-II.

IV. Conclusions

This paper proposes AER neuro-inspired filters for vision processing by a Cellular Automata approach. These filters can implement two layers, one for the input processing and the second one thanks to the evolution rule.

Two AER filters based on 3x3 kernel convolutions have been implemented for FPGA using a Cellular Automata approach. An AER filter based on distributed computational units has been implemented in which each cell is assigned to a different computational unit. This implementation performs up to 150 MOPS for a 3x3 kernel and yields up to 16.6 Mega-events per second in a Spartan-II.

An improved version called AER Cellular Automata in distributed memory has also been implemented to save resources by reducing a number of computational units. A performance of 75 MOPS has been measured for 3x3 kernels. This performance may be easily improved for higher kernel size, allowing up to 2 GOPS for 11x11 kernels, which is feasible for the distributed memory implementation.

A real scenario has been used to prove these implementations consisting of an AER sequencer, the AER-CA convolution processor to test, and an AER monitor or datalogger.

Acknowledgment

The authors would like to thank the Technical University of Valencia for hosting first author during April 2009 for

developing this work. The authors also thank the Spanish grants SAMANTA II (TEC2006-11730-C03-02), VULCANO (TEC2009-10639-C04-02) and BrainSystems (P06-TIC-01417) for supporting this work.

REFERENCES

[1] J. von Neumann, "The Theory of Self-reproducing Automata", A. Burks, ed., Univ. of Illinois Press, Urbana, IL, 1966.

[2] A. Burks (ed). Essays on Cellular Automata. Univ. Illinois Press. 1970.

[3] U. Pesavento. An implementation of von Neumann's self-reproducing machine. Artificial Life, Vol. 2, pp. 337-354, 1995.

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

[5] A. Cohen et al., Report to the National Science Foundation: Workshop on Neuromorphic Engineering, Telluride, Colorado, USA, June-July 2006. [www.ine-web.org]

[6] M. Mahowald. "VLSI Analogs of Neuronal Visual Processing: A Synthesis of Form and Function". Ph.D. Thesis. California Institute of Technology, Pasadena, California 1992.

[7] 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.

[8] A. Linares-Barranco, R. Paz, F. Gómez-Rodríguez, A. Jiménez, M. Rivas, G. Jiménez and A. Civit "FPGA Implementations comparison of Neuro-Cortical inspired Convolution Processors for Spiking Systems". Lecture Notes in Computer Science Vol. 5517, pp.97-105, 2009.

[9] N. Farrig, 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

[10] F. Gomez-Rodriguez, R. Paz, A. Linares-Barranco, M. Rivas, L. Miró, G. Jimenez, A. Civit. "AER tools for Communications and Debugging". Proceedings of the IEEE ISCAS 2006, Kos, Greece. May 2006.

[11] K. A. Boahen. "Communicating Neuronal Ensembles between Neuromorphic Chips". Neuromorphic Systems. Kluwer Academic Publishers, Boston 1998.

[12] Lichtsteiner, P.; Posch, C.; Delbruck, T. "A 128× 128 120 dB 15 μs Latency Asynchronous Temporal Contrast Vision Sensor". IEEE Journal of Solid-State Circuits, IEEE Journal, Vol 43, Issue 2, pp. 566-576, Feb. 2008.

[13] A. Linares-Barranco, G. Jimenez-Moreno, A. Civit-Ballcels, and B. Linares-Barranco. "On Algorithmic Rate-Coded AER Generation". IEEE Transaction on Neural Network, Vol. 17, No. 3, pp. 771-788, May, 2006.