FPGA Implementations Comparison of Neuro-cortical Inspired Convolution Processors for Spiking Systems

A. Linares-Barranco, R. Paz, F. Gómez-Rodríguez, A. Jiménez, M. Rivas, G. Jiménez, and 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 in frames and each one has to be completely processed before the next frame arrives. Recently a new method for computing convolutions based on the neuro-inspired philosophy of spiking systems (Address-Event-Representation systems, AER) is achieving high performances. In this paper we present two FPGA implementations of AER-based convolution processors that are able to work with 64x64 images and programmable kernels of up to 11x11 elements. The main difference is the use of RAM for integrators in one solution and the absence of integrators in the second solution that is based on mapping operations. The maximum equivalent operation rate 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.

1 Introduction

Digital vision systems process sequences of frames from conventional video sources, like cameras. For performing complex object recognition algorithms, sequences of computational operations are 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 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 beginning at the time the information is captured by the retina. Although cortex has feedback connections, it is known that a very fast and purely feed-forward recognition path exists in the visual cortex [1][3].

* This work was supported by Spanish grant TEC2006-11730-C03-02 (SAMANTA 2) and by the Andalussian Council project BrainSystem (P06-TIC-01417).

In recent years significant progress has been made towards the understanding of the computational principles exploited by visual cortex. Many artificial systems that implement bio-inspired software models use biological-like (convolution based) processing that outperform more conventionally engineered machines [4]. However, these systems generally run at extremely low speeds because the models are implemented as software programs on conventional computers. For real-time solutions direct hardware implementations of these models are required. However, hardware engineers face a large hurdle when trying to mimic the bio-inspired layered structure and the massive connectivity within and between layers. A growing number of research groups world-wide are mapping 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 (Fig. 1). Each time a cell on a sender device generates a spike, it communicates with the array periphery and a digital word representing a code or address for that pixel is placed on the external inter-chip digital bus (the AER bus). Additional handshaking lines (Acknowledge and Request) are used for completing the asynchronous communication. In the receiver chip 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].

There is a growing community of AER protocol users for bio-inspired applications in vision, audition systems and robot control, as demonstrated by the success in the last years of the AER group at the Neuromorphic Engineering Workshop series [7].

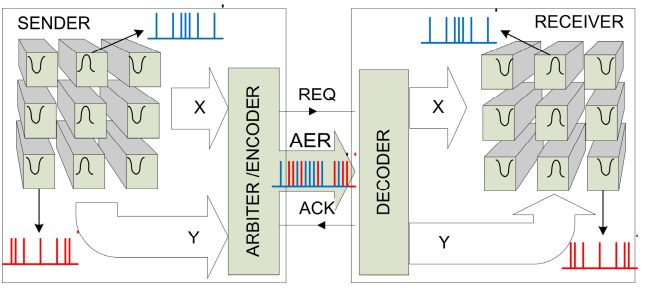

Fig. 1. Rate-coded AER inter-chip communication scheme

Table 1. AER Convolution processors comparison

Integrator based convolutions	**Mapper based convolutions**	**IF based convolutions [15]**
Poisson output	Poisson output	Arbitrated output
From 3x3 to 11x11 kernels	2x2 and 3x3 kernels	Up to 32x32 kernels
64x64 images	64x64 images	32x32 images per chip. Chips are scalables
50MHz clock	50MHz clock	200MHz clock
Fully digital 8-bit cells on RAM	No cells. Results on-the-fly by mappings	Analog integrators (capacitors)

In this paper we present two FPGA implementations of neuro-cortical inspired convolution processors. These circuits are similar from a previous work implemented in a VLSI chip [13], but with some differences, presented in table 1.

Section two explains how to convolve using spiking neurons. A VLSI chip [15] uses this mechanism. This paper presents an FPGA implementation with Poisson output distribution in section 3. In section 4 a new way of implementing convolutions not based on integrators is presented. Then in section 5 implementation details are presented. Section 6 discusses the performance of these spike-based convolution processors compared to frame-based digital convolution processors implemented on FPGA.

2 Convolutions with Spikes

a) Description. Complex filtering processing based on AER convolution chips already exists [13]. These chips are based on IF neurons. 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.

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

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.

For implementing convolutions using spikes let's suppose Y a matrix of integrators (capacitors for analog circuits or registers 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 $P \cdot G$ (proportional to the gray level of the pixel). Each event identifies the source pixel by the address on the AER bus. The value of the pixel is coded by the frequency of appearance of this address on the bus. 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:

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

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]. Since the AER Poisson generation method is accessing randomly *Y* matrix, this cannot be reset. A periodic and parameterized mechanism of forgetting is used instead of reseting cells when they spike. Periodically, the convolved matrix *Y* is accessed for subtracting a constant value to each element.

In [13] 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 cero 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 2 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 interface between the microcontroller and the FPGA uses a Serial Peripheral Interface (SPI). The circuit in the FPGA can be divided into four parallel blocks:

- *AER event reception and processing.* The input traffic AER protocol 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 RAM in the FPGA. Each position of the RAM is 8-bit length. The kernel is stored in an 11x11 matrix with 8-bit resolution. Each kernel value is in the range of -127 to 127. When the kernel is added the result stored in the matrix is limited between 0 and 255, because no signed events are implemented.
- *Forgetting mechanism.* For high AER input bandwidth and / or weighted kernels, maximum cell value could be reached quickly and thus, next events are lost. Let's call this situation saturation effect. For this reason, to avoid errors, a configurable forgetting circuitry is present in the architecture. The forgetting is based on a programmable counter that accesses the *Y* matrix periodically in order to decrease its values by a constant. Forgetting period and quantity are configured through USB. Since the saturation effect depends on the AER input bandwidth and the kernel values (weight), the forgetting parameters depend on these two.

- *Poisson like AER output. Y* 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 [9][10].
- *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 FPGA. The testing scenario consists on a USB-AER working as Uniform AER generator [9], the output is connected to an AER-Robot working as the AER convolution processor; and its output is connected to a second USB-AER working as an AER datalogger or framegrabber. A laptop is connected to the three boards, to manage all of them from MATLAB, through USB.

Figure 2 (left, top row) shows a source image and its negative version. These images 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 with the same kernel to the source bitmap used by AER generator.

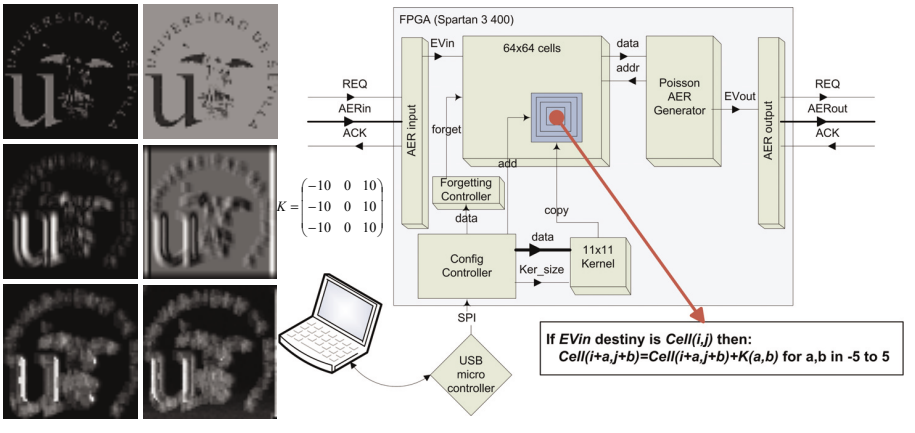

Fig. 2. Right: block diagram of the FPGA AER-based convolution processor. Left: source images (top), MATLAB simulation images result (center) and FPGA AER convolution output histograms (bottom).

3 Spike-Based Convolutions with a Probabilistic Mapper

a) Description. Let's explain how to implement this using a probabilistic multi-event mapper [15]. This mapper is able to send *M* different events per input event. 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*) to be sent or not. For each input event (In_i), up to *dim(K)* output events are generated.

$$Out_{a,b} = R_{a,b} \cdot P_{a,b} \cdot In_i \quad \forall a,b \in \dim(K)$$

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 probability and repetition factor of the input events in the neighbourhood. This neighbourhood is defined by the kernel size. Therefore, the number of events for a fixed output address follows the expression:

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

So, 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 K_i=*1.2*, repetition factor is R_i=*2* and probability is P_i=*0.6*.

b) Implementation II: Probabilistic Mapper. The identification or address of each emitter neuron that is present in the AER bus, is used to index a mapping table in the AER mapper. Then, a list of mapped addresses is sent replacing the original one. Each mapped address 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 and repeating each mapped event according to *R* (see figure 3, left).

With this mapper, sophisticated operations can be performed in the AER traffic during the transmission time, like small kernel convolutions. As the probability can be modified per each mapped event, it is possible to implement a *one-to-dim(K)*

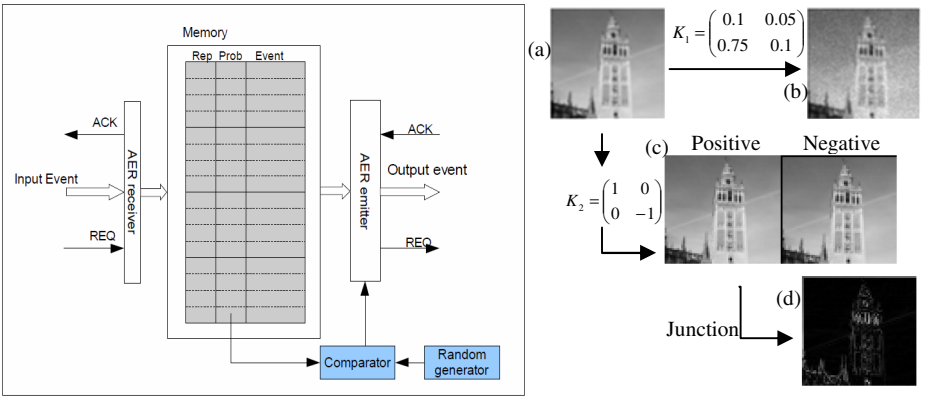

Fig. 3. Left: Probabilistic Multi-event mapper block diagram. Right: (a) source image, (b) K1 convolution histogram, (c) K2 postive output traffic and negative traffic, (d) abs(positve+negative) K2 histogram.

mapping, where each of these *dim(K)* events can be modulated with *P* and *R*, as expressed in previous section. Thus, this architecture is able to send weighted traffic to a neighbourhood per each input event. Furthermore, 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.

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 [11][12].

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) [11]. 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 3 right shows two examples. The same source image (64x64) in AER format, generated by the Uniform AER generator [9], has been used (a). The second image is the normalized histogram of collecting AER traffic for 100ms, after applying a simple 2x2 kernel convolution K1 (b). 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. The last image (d) shows the absolute value of the combination of positive and negative traffic, using an up-down counter per address.

4 Comparison with Digital Frame-Based Convolution Processors

Digital frame-based convolution processors implemented in FPGA, GPUs or CPUs usually measure their performance by calculating the number of operations per second (MOPS). In this way, a frame-based convolution processor has to calculate 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 [16][17] a performance study was presented for different kernel sizes and platforms (see table 2).

A spike-based convolution processor with integrators cells is performing a number of ADD operations per each input event, but no MULTs are done because these are replaced by an explosion in the number of ADD operations due to the spike-based representation of the information. In this case, the number of ADD per input event is equal to the kernel size, since the kernel is added to the neighborhood of the pixel address of the incoming event. Measuring AER protocol time in the input, it is feasible to calculate the MOPS.

There exist previous works related to convolution solving based on AER, like [13], where a 32x32 pixels convolution chip is based on analog Integrate and Fire neurons. This chip can process images with a 31x31 kernel. Each input spike needs 330 ns to be processed. Therefore, this system yields 31x31/330=2910MOPS.

FPGA implementation explained in section 3 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. For example, for 3x3 kernel, 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 3x3x2 cycles are needed per each input event, apart from those needed for the handshake. The architecture can be easily improved by having a number of adders equivalent to the kernel row size, and allowing the access to the RAM by blocks. Table 2 shows the real performance in MOPS for a one ADD architecture, and the simulated performance for the N ADD architecture of the NxN kernel convolution processor.

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^2 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 one possible equivalent number of operations can be the number of mappings per each input event. Table 2 shows the results.

Table 2. Mega Operations per Second for frame and spike-based convolution processors

MOPS	Digital frame based 2D convolutions [16][17]				Spike-based 2D Convolutions			
					By mapping	By RAM integrators		By analog caps [13]
Kernel size (NxN)	Pentium 4 3GHz CPU	6800 ultra GPU	Spartan 3	Virtex II-Pro	Spartan II (100MHz)	Spartan 3 (real) 1 sum (50MHz)	Spartan 3 (sim) N sum (50 MHz)	VLSI 0,35um (200MHz)
2x2	14	1070	190	221	18,2	-	-	-
3x3	9,7	278	139	202	22,5	20,45	34,61	-
5x5	5,1	54	112	162	-	23,14	65,78	-
7x7	2,6	22	90	110	-	24,01	98,00	-
9x9	1,6	9	73	61	-	24,39	130,00	-
11x11	1,2	4,7	23	48	-	24,59	163,51	-
31x31	-	-	-	-	-	-	495,00	2910

5 Conclusions

Two neuro-cortical layers of 64x64 cells with NxN convolution kernel hardware implementation on FPGA have been presented. One can work with kernels of up to N=11 (~500Ksynapses), with 8-bit integrator cells, and the other with N=3 (~37Ksynapses), with no integrator cells. The AER output of both follows a Poisson distribution since LFSR are used. Both are compared with 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. A performance study is presented comparing frame-based digital convolution processors and VLSI analog-integrators spike-based convolution processors. Digital frame-based architectures are limited by the number of parallel ADDs and MULTs so their performance decreases while the kernel size increases. Digital RAM-cell spike-based architecture is limited by RAM access. When parallelizing the number of ADDs, the performance increases with the kernel size. Spike-based Mapper architecture avoids ADD operations since the operations consist in projecting inputs events with weighted and signed events and then averaging them in short periods of time. The performance of these systems depends on SRAM access time. Kernel sizes are limited by the mapping table capacity.

References

1. Serre, T.: Learning a Dictionary of Shape-Components in Visual Cortex:Comparison with Neurons, Humans and Machines, PhD dissertation, MIT. Comp. Sci. & AI Lab Technical Report, MIT-CSAIL-TR-2006-028 CBCL-260 (April 2006)
2. Shepherd, G.: The Synaptic Organization of the Brain, 3rd edn. Oxford University Press, Oxford (1990)
3. Thorpe, S., et al.: Speed of processing in the human visual system. Nature 381, 520–522 (1996)
4. Neubauer, C.: Evaluation of Convolution Neural Networks for Visual Recognition. IEEE Trans. on Neural Networks 9(4), 685–696 (1998)
5. Sivilotti, M.: Wiring Considerations in analog VLSI Systems with Application to Field-Programmable Networks. Ph.D. Thesis, California Institute of Technology, Pasadena CA (1991)
6. Boahen, K.: Communicating Neuronal Ensembles between Neuromorphic Chips. Neuromorphic Systems. Kluwer Academic Publishers, Boston (1998)
7. Cohen, A., et al.: Report to the National Science Foundation: Workshop on Neuromorphic Engineering, Telluride, Colorado, USA (June-July 2004), http://www.ini.unizh.ch/telluride
8. Softky, W.R., Koch, C.: The highly irregular firing of cortical cells is inconsistent with temporal integration of random EPSPs. J. Neurosci. 13(1), 334–350 (1993)
9. Linares-Barranco, A., Jimenez-Moreno, G., Linares-Barranco, B., Civit-Ballcels, A.: On Algorithmic Rate-Coded AER Generation. IEEE Trans. On Neural Networks 17(3) (May 2006)
10. Linares-Barranco, A., Oster, M., Cascado, D., Jimenez, G., Civit-Balcells, A., Linares-Barranco, B.: Inter-Spike-Intervals analysis of AER-Poisson-like generator hardware. Neurocomputing 70(16-18), 2692–2700 (2007)
11. Paz, R., et al.: Test Infrastructure for Address-Event-Representation Communications. In: Cabestany, J., Prieto, A.G., Sandoval, F. (eds.) IWANN 2005. LNCS, vol. 3512, pp. 518–526. Springer, Heidelberg (2005)
12. Linares-Barranco, A., Paz, R., Jiménez, A., Varona, S., Jiménez, G.: An AER-based Actuator Interface for Controlling an Antrophomorphic Robotic Hand. In: Mira, J., Álvarez, J.R. (eds.) IWINAC 2007. LNCS, vol. 4528, pp. 479–489. Springer, Heidelberg (2007)
13. Serrano-Gotarredona, R., et al.: A Neuromorphic Cortical-Layer Microchip for Spike-Based Event Processing Vision Systems. IEEE Trans. on Circuits and Systems I. 53(12) (December 2006)
14. Serrano-Gotarredona, T., Andreou, A.G., Linares-Barranco, B.: AER image filtering architecture for vision processing systems. IEEE Trans.Circuits and Systems (Part II): Analog and Digital Signal Processing 46(9), 1064–1071 (1999)
15. Linares-Barranco, A., et al.: Implementation of a time-warping AER mapper. In: ISCAS 2009, Taiwan (2009)
16. Cope, B., et al.: Implementation of 2D Convolution on FPGA, GPU and CPU. Imperial College Report, http://cas.ee.ic.ac.uk/people/btc00/index_files/Convolution_filter.pdf
17. Cope, B., Cheung, P.Y.K., Luk, W., Witt, S.: Have GPUs made FPGAs redundant in the field of video processing? In: Proc. IEEE International Conference on Field-Programmable Technology 2005, pp. 111–118 (December 2005)