Advanced Vision Processing Systems: Spike-Based Simulation and Processing

José-Antonio Pérez-Carrasco[1,2], Carmen Serrano-Gotarredona[2], Begoña Acha-Piñero[2], Teresa Serrano-Gotarredona[1], and Bernabe Linares-Barranco[1]

[1] Instituto de Microelectrónica de Sevilla (IMSE-CNM-CSIC)
Avenida Reina Mercedes, s/n. 41012, Seville, Spain
{jcarrasco,terese,bernabe}@imse.cnm.es
[2] Dpto. Teoría de la Señal, ETSIT, Universidad de Sevilla,
Avda de los descubrimientos, s/n. 41092, Seville, Spain
{cserrano,bacha}@us.es

Abstract. In this paper we briefly summarize the fundamental properties of spike events processing applied to artificial vision systems. This sensing and processing technology is capable of very high speed throughput, because it does not rely on sensing and processing sequences of frames, and because it allows for complex hierarchically structured neurocortical-like layers for sophisticated processing. The paper describes briefly cortex-like spike event vision processing principles, and the AER (Address Event Representation) technique used in hardware spiking systems. In this paper we present a simulation AER tool that we have developed entirely in Visual C++ 6.0. We have validated it using real AER stimulus and comparing the outputs with real outputs obtained from AER-based devices. With this tool we can predict the eventual performance of AER-based systems, before the technology becomes mature enough to allow such large systems.

1 Introduction

Artificial man-made machine vision systems operate in a quite different way from biological brains. Machine vision systems usually capture and process sequences of frames. This frame-based processing is slow, especially when many convolutions need to be computed in sequence for each input image or frame, such as in wavelet based processing. Biological brains do not operate on a frame by frame basis. In the retina, each pixel sends spikes (also called events) to the cortex when its activity level reaches a threshold. Very active pixels will send more spikes than less active pixels. All these spikes are transmitted as they are being produced, and do not wait for an artificial "frame time" before sending them to the next processing layer. Besides this frame-less nature, brains are structured hierarchically in cortical layers [1]. Neurons (pixels) in one layer connect to a projection-field of neurons (pixels) in the next layer. This processing based on projection-fields is equivalent to convolution-based processing [2]. This

fact has been exploited by many researchers to propose powerful convolution based image processing algorithms [3][4]. However, convolutions are computationally expensive and it seems unlikely that software programs running on the fastest of today´s computers could emulate the high number of convolutions that the brain might perform when fast vision processing is considered. A solution to this could be Address-Event-Representation (AER). AER is a promising emergent hardware technology that shows potential to provide the computing requirements of large frame-less projection-field based multi-layer systems. It is an event-based representation hardware technique for communicating events between layers of neurons in different chips. AER was first proposed in 1991 in one of the Caltech research labs [5][6], and a wide community of neuromorphic hardware engineers has used it since then. AER has also been used for auditory systems, competition and winner-takes-all networks, and even for systems distributed over wireless networks [7][8][9]. However, the high potential of AER has become even more apparent since the availability of AER convolution chips [10][11][12]. These chips, which can perform large arbitrary kernel convolutions (32x32 in [11]) at speeds of about 3×10^9 *connections/sec/chip*, can be used as building blocks for larger cortical-like multi-layer hierarchical structures, because of the modular and scalable nature of AER based systems. At present, only a small number of such chips have been used simultaneously [10], but it is expected that hundreds or thousands of such modular AER chips could be integrated in a compact volume. This would eventually allow the assembly of large cortical-like projection-field and event-based frame-less vision processing systems operating at very high speeds. The objective of this paper is to illustrate and to introduce the processing power of AER-based systems, based on the performance characteristics of already available AER hardware modules (chips), but through behavioral simulations. In spite of existing a lot of work developed in the study and simulation of the different layers in which the brain is structured [13][14], this work is not focused on simulating the biological aspects of the brain, but on simulating existing or possible hardware AER-based devices that are biologically inspired. To do this, we have developed an open AER simulator entirely in Visual C++ 6.0. With this tool we will be able to describe any AER module (including timing and non-ideal characteristics), and assemble large netlists of many different modules. This makes it possible to obtain a good estimation of the delays and processing power of the simulated systems. Furthermore, the AER behavioral simulator can be used to test new AER processing modules within large systems, and thus orient hardware developers on what kind of AER hardware modules may be useful and what performance characteristics they should possess. To validate the tool, we have used real AER stimulus obtained with an electronic motion-sensing retina [9][10] and we have simulated two real AER-based hardware implementations [10][11]. The outputs have been compared to those obtained with the real systems. Finally, in the last section we test the proposed tool with a single layer neural network based on AER for recognizing handwritten digits. In particular, we use the MNIST database [15] consisting of 70000 handwritten digits and provide a 91% of correct classification.

2 AER-Based Convolution

To illustrate how AER convolution is performed event by event (without frames) consider the example in Fig. 1. Fig. 1(a) corresponds to a conventional frame based convolution, where a 5x5 input image f is convolved with a 3x3 kernel h, producing a 5x5 output image g. Mathematically, this corresponds to Eq. 1.

$$g(i,j) = \sum_m \sum_n h(m,n) f(i-m, j-n). \tag{1}$$

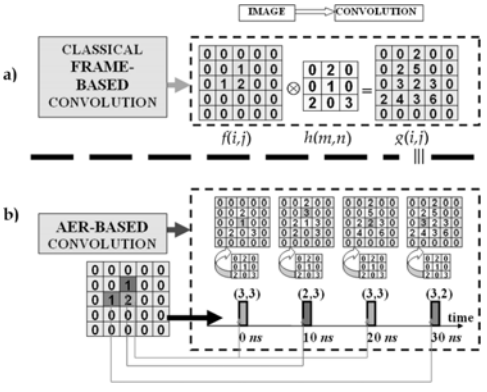


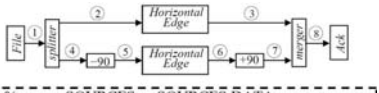
Fig. 1. *(a)*classical frame based and *(b)* AER-Based convolution processing

In an AER system, shown in Fig. 1, an intensity retina sensing the same visual stimulus would produce events for 3 pixels only (those sensing a non-zero light intensity). The pixel at coordinate (3,3) senses twice as much intensity as pixels (2,3) and (3,2). The event frequency of address (3,3) will therefore be twice that of pixels (2,3) and (3,2). In this particular case, the retina can send a new event every 10ns. Thus, for this particular stimulus, after four events we already have a valid representation of the stimulus and only 40ns are required to transmit it. In a practical situation the two events of pixel (3,3) would be separated by more than 20ns. Every time the convolution chip receives an event from the retina chip, the kernel is added to the array of pixels (which operate as adders and accumulators) around the pixel having the same event coordinate. Note that this is actually a projection-field operation. In this way, after the four retina events have been received and processed, the result accumulated in the array of pixels in Fig. 1(b) is equal to that in Fig. 1(a). Additionally, in an AER convolution chip, a fixed threshold level is defined for all pixels. Whenever a pixel exceeds that level, it will generate an output event, and the pixel will be reset. Consequently, events are generated, transmitted, and processed immediately, without waiting for any frame timing constraints. In a more realistic situation, the retina pixel values are higher and more events are sent per pixel. However, note that more intense pixels have higher frequencies, and consequently their

events will start to come out earlier, and will be processed first. In general, more intense pixels are more information-relevant pixels (especially in contrast or motion retinae). In AER systems, since events are processed by a multi-layer cortical-like structure as they are produced by the sensor, it is possible to achieve successful recognition after a relatively small fraction of the total number of events are processed [16].

3 AER Simulator Tool

In the simulator proposed a generic AER system is described by a netlist that uses only two types of elements: instances and channels. An instance is a block that generates and/or produces AER streams. We have implemented a basic library of instances, and any user can easily modify them or add new ones. For example, a retina chip would be a source that provides an input AER stream to the AER system [9]. A convolution chip [10], [11], [12] would be an AER processing instance with an input AER stream and an output AER stream. For every input event at coordinate (x, y) a convolution map of a size specified by the user is added in the pixel array stored in the convolution module around the input event coordinate (Fig. 1). In a realistic situation, each pixel should be implemented as an integrate-and-fire neuron with a threshold and physical delays should be modeled. A splitter [10] would be an instance which replicates the events from one input AER stream onto several output AER streams. Similarly, a merger [10] is another instance which would receive as input several AER streams and merge them into a single output AER stream. Other developed instances are a winner-take-all module and a multiplier. Both instances will be described in the next sections. AER streams constitute the nodes of the netlist in an AER system, and are called channels. Channels represent point-to-point connections. Fig. 2 shows an example system. The system contains 7 instances and 8 channels. The netlist description is provided to the simulator through a text file. The ASCII file netlist corresponding to the example system is shown at the bottom in Fig. 2. Channel 1 is a source channel. Each source channel needs a line in the netlist file, starting with the key word sources, followed by the channel



%	SOURCES	SOURCES DATA		
sources	{1}	{data1}		
%MODULES	IN	OUT	PARAMS	STATES
splitter	{1}	{2, 4}	{params1}	{states1}
conv1	{2}	{3}	{params2}	{states2}
rotate90	{4}	{5}	{params3}	{states3}
conv1	{5}	{6}	{params4}	{states4}
rotate-90	{6}	{7}	{params5}	{states5}
merger	{3, 7}	{8}	{params6}	{states6}
ack_only	{8}	{8}	{params7}	{states7}

Fig. 2. Example AER-based system emulated by our simulation tool and its netlist ASCII file

number and the file containing its events. The following lines describe each of the instances, one line per instance in the network. The first field in the line is the instance name, followed by its input channels, output channels, name of a text file containing its parameters, and name of a text file containing its initial state. Each instance is described by a C++ function whose name is the name of the instance. The simulator imposes no restriction on the format of the parameters and state structures. This is left open to the user writing the code of the function of each instance. The simulator only needs to know the name of the files where these structures are stored. Channels are described by a list of events. Each element in the list has two components. The first one corresponds to information of one event and the second one is a pointer to the following event to be processed in the list. Each event has six components: 'x' and 'y', that represent the coordinates or addresses of the event, 'sign' represents its sign, 'tprereq' represents the time at which the emitter instance creates the event, 'treqef ' represents the time at which the receiver instance processes the event, and 'tack' represents the time at which the receiver instance finally acknowledges the event. We distinguish between a pre-Request time and an effective Request time. The first one is only dependent on the emitter instance, while the second one requires that the receiver instance is ready to process an event request. This way, we can provide as source a full list of events which are described only by their addresses, sign, and times. Once an event is processed by the simulator, its final effective request and acknowledge times are established. Before starting the simulator, the events of the source channels have to be provided in an input text file (we have also implemented some tools to create these text files from real aer streams or from images). During the simulation, events in the channels are established. After the simulation, the user can visualize the computed flow of events in the different channels. The execution of the simulator is as follows. Initially the netlist file is read as well as all parameters and states files of all instances. Each instance is initialized according to the initial state of each instance. Then, the program enters a continuous loop that performs the following steps:

1. All channels are examined. The simulator selects the channel with the earliest unprocessed event (earliest pre-Request time). An event is unprocessed when only its pre-Request time has been established, but not its final Request time nor its acknowledge time.
2. Once a channel is selected for processing, its earliest unprocessed event is provided as input to the instance the channel is connected to. The instance updates its internal state. In case this event triggers new output events from this instance on its output channels, a list of new unprocessed events is provided as output. These output events are included by the simulator in the correct position of the respective channels, which at a later time should be processed by their respective destination instances. Finally, the simulator stores the new state for the instance under execution and goes back to 1.

4 Results for AER-Based Implementations

To validate the tool described above, we have implemented two simulations of two AER systems that had been previously built in hardware [10][11]. All the parameters describing the modules such us thresholds, forgetting ratio, kernels values, delays, array sizes, etc. have been chosen according to the specifications of AER devices [10][11] and adjusted to produce an output frequency in the same order as the input frequency of activity, as done in the physical implementations. Finally, we propose and simulate also an AER-based single layer neural network to detect hand-written digits from the MNIST database [15].

4.1 Detection and Tracking of Moving Circles with Different Radius

The first developed implementation was built to simulate the demostration system described in [10]. This system could simultaneously track two objects of different size. A block diagram of the complete system is shown in Fig. 3. The complete chain consisted of 17 pieces (chips and PCBs), and was intended to detect two solid circles of different radiuses on a rotating disc. The detection was implemented with two convolution chips programmed to detect circumferences of radius 4 pixels and 9 pixels, respectively. The AER scheme that we have used to emulate the detection system is that described in Fig. 4. The system receives as input the events captured by the electronic retina, the movement of a rotating disc with two solid circles of different radiuses. The captured data, available in data files, were previously converted to a valid format for our simulation tool. These events reach a convolution module programmed a kernel tuned to detect a circumference of a certain radius (convolution chip in Fig. 4). The output events describing the center of the circumference we want to detect are sent to a winner-take-all module. The output activity of this module responds only to the incoming addresses having the highest activity. We can implement the winner-take-all module using a convolution chip with a kernel that is positive in the center and negative in the rest of values and using the output activity from this chip as feedback to the input of the chip. The input frequency in each pixel belonging to each circumference was 266Hz, that is, 104312 events were produced in 4.5s when the kernel is tuned to detect the small circumference. We have obtained 1911 events at the output in total, all of them due to the pixel detecting the center of the circumference, which implies that this pixel produces events with a frequency of 266 Hz approximately. Note that AER streams are not represented by sequences of frames (as in conventional video). However, to show the results graphically, we have collected input and output events during time intervals of 33ms and show 2-D images. In Fig. 5 the four images in the left represent the images reconstructed with the hardware implementation (images were obtained with a java tool [10][11]). The gray values correspond to non-activity. Black values correspond to changes in intensity due to the motion sensing retina at the input and white levels at the bottom figures correspond to the pixels detecting the center of the moving ball at the output). The four images on the right correspond to the images obtained using our C++

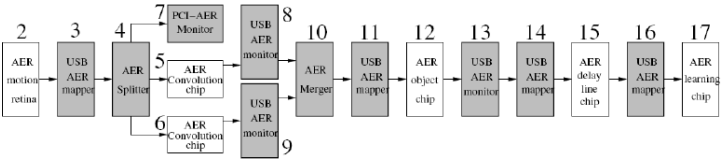


Fig. 3. Block diagram of the hardware implementation AER-based system to detect objects of different shape and size

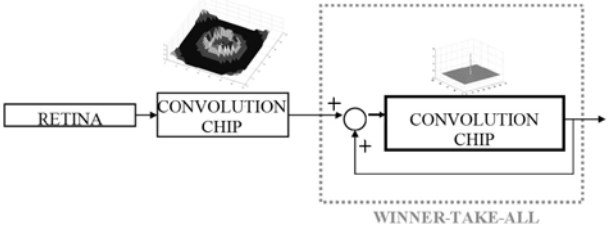


Fig. 4. Block diagram of the AER system developed to simulate the hardware implementation

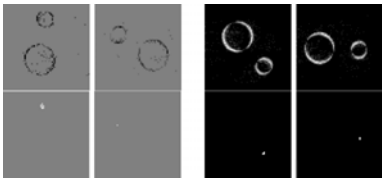


Fig. 5. *On the left*, input and output obtained with the hardware implementation. *On the right*, input and output obtained with the simulated implementation.

simulation tool. Black levels correspond to non-activity and white levels correspond to pixels producing activity.

4.2 Recognition of a Rotating Propeller Rotating at High Speed

The second experiment demonstrates the high-speed processing capabilities of AER based systems. It is the recognition and tracking of a highspeed S-shaped rotating propeller at 5000 rev/sec [11] and moving across the screen. At this speed, a human observer would not be able to discriminate the propeller shape and he would only see a moving ball across the screen. The propeller has a diameter of 16 pixels. The AER simulated system is again that shown in Fig. 4. This time, the convolution chip was programmed with a kernel to detect the center of the S-shaped propeller when it is in the horizontal position. In Fig. 6(a), the kernel is shown. In Fig. 6(b) and (c) we show the 2-D input (propeller) and output reconstructed images when we consider a 50μs interval of time collecting events

(1/4 of a rotating movement). In Fig. 6(d) and (e) we show the 2-D input and output images when we consider a 200ms interval of time collecting events (corresponding to one complete back-and-forth screen crossing). As it can be seen, only those pixels that detect the center of the propeller produce activity. The propeller is properly detected and tracked at every moment in real time. Note that using conventional frame-based image processing methods to discriminate the propeller is a complicated task, which requires a high computational load. First, images must be acquired with an exposure time of at least 100μs and all this must be performed in real time (the propeller is rotating at 5000 rev/sec). As done previously, all the parameters describing the modules were adjusted to produce an output frequency in the same order as the input frequency of activity (137 KHz), as done in the physical implementations [11].

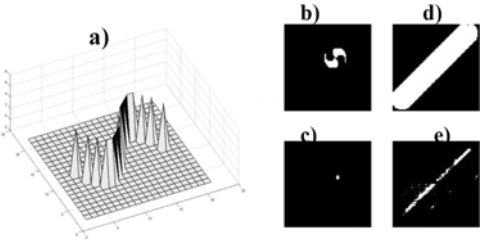


Fig. 6. *a)* Kernel used to detect the propeller, *b)* and *c)* input and output when we collect events during 50μs, *d)* and *e)* input and output when we collect events during 200ms

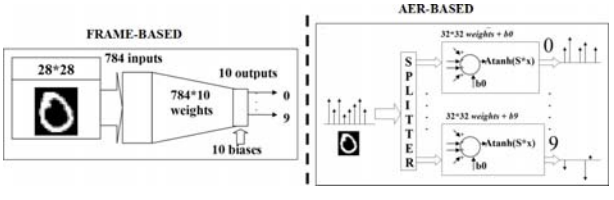


Fig. 7. Frame-based (on the left) and AER-based (on the right) implementation of a single layer neural network to detect digits of 28x28

4.3 Single Layer Neural Network

The last system that we have simulated is a single layer neural network trained with back-propagation [17]. When designing a pattern recognition system, one of the main problems is that the recognition accuracy is largely determined by the ability of the designer to come up with an appropriate set of features. This turns out to be a daunting task which, unfortunately, must be redone for each new problem. A possible solution to these problems is the use of neural networks [17]. Aimed to this, we have developed an AER-based system that emulates directly

a frame-based a fully connected single layer neural network trained with back-propagation [17] and which is called Net-1. Net-1 is shown on the left of Fig. 7 with 10 sigmoid output units (7850 weights including the biases). The database MNIST [15] consisting of 70000 28x28 images of hand-written digits has been employed. 60000 are used for training and 10000 are used for testing purposes. Each pixel from one image constitutes an input to the network. Each output unit in the network computes a dot product between its input vector and its weight vector. This weighted sum, denoted xj for unit j, is then passed through a sigmoid squashing function (a scaled hyperbolic tangent) to produce the state of unit j, denoted by yj (Eq. 2):

$$yj = A \tanh(S * xj). \tag{2}$$

Where A is the amplitude of the function and S determines its slope at the origin. In the training phase, each weight is updated using backpropagation:

$$w_{ij} = w_{ij} - \epsilon \frac{\partial E}{\partial w_{ij}}. \tag{3}$$

Where w_{ij} is the weight that connects pixel i to output neuron j and E is the error at the output computed as:

$$E = \sum_{j} \frac{1}{2}(yj - dj)^2. \tag{4}$$

Where dj is the desired output for each input in unit j when using the training set. The same rule is used for the biases. After training the net, only the output neuron sensitive to the stimulus will produce positive activity at the output. A classification rate of 91% was obtained when we tested the net with the test images. On the left of Fig. 7 we show the frame-based implementation of the single layer neural network. In the figure, a digit corresponding to one training digit belonging to the MNIST database is supplied to the network and the output neuron sensitive to that input digit will reach a state of value '1'. The rest will have a value of '-1'. On the right part of Fig. 7 we show our AER-based scheme. This time, we do not have real AER input stimulus. However, our simulator proposed allows us to convert 2-D images into events. In this way, we have coded all the images in the MNIST database into events separated each other 10ns. Each flow of events corresponding to one digit is used as input to the system. When an event belonging to the input stimulus reaches a splitter module, it is replicated in each one of the ten output ports. Events travelling through one of this output ports come to a multiplier module. A multiplier module consists of an array with 784 weights and one bias value. When an event arrives to this module, its address is decoded and the weight specified with that address in the pixel array, which is stored in the module, is added to the state of the single neuron inside the module. If the state of the neuron reaches a certain positive or negative threshold, it produces a new output event positive or negative respectively. This event will be sent automatically to the output port. Then, the neuron resets

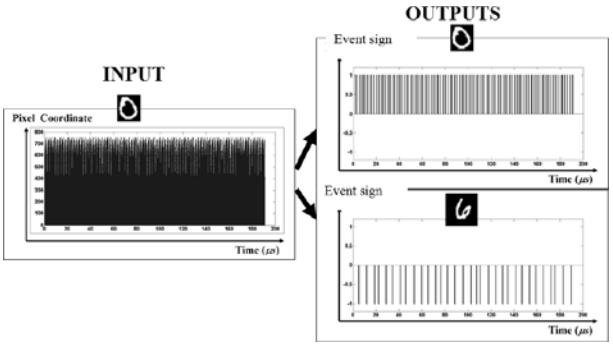


Fig. 8. Input and output events for neurons '0' and '6'

itself and it is initialized with the bias value specified inside each multiplier. All the weights and biases used in the implementation were computed with backpropagation using a frame-based scheme (this is not difficult because only one layer is involved). As an example, in Fig. 8 we show the input events when we used a version of digit '0'. For this input, in the same figure we show also the output events obtained for the neurons sensitive to input '0' and input '6' (the activity for the rest of neurons is quite similar). As it can be seen, output events are obtained automatically, without the need of waiting for the entire frame-time. In less that 3μs (note that duration of the input stimulus is almost 200μs) since the first input event reached the system, we have output events indicating the correct detection of digit '0' (positive events). The rest of output neurons produce negative events, indicating that they are not sensitive to that stimulus. When we used the entire set of 10000 test images and converted them to events, we got a recognition rate of 91%. This rate is the same as the rate obtained in the net-1 frame-based implementation. It is obvious that AER allows for fast processing providing the good results that we had using classical frame-based methods but now in real time. In a real hardware implementation using AER modules we would be able to process input events at speeds going from 33 Mevents/sec to 3 Mevents/sec [11] when we use the maximum size for kernels allowed in convolution or multiplier chips. This implies that the real time recognition here simulated could be easily achieved by a real system and with aproximately the times here described. In the future, neural networks with more layers will be implemented in AER in order to achieve higher classification rates as those obtained (almost 100%) in recent frame-based schemes (LeNet-5 [18]).

5 Conclusions

In this paper, we have described a simulation C++ tool for AER-based systems. The tool is able to process around 20Kevents/sec. Hardware AER modules are able to process input events at speeds going from 33 Mevents/sec to

3 Mevents/sec [11]. However, in spite of the tool being slower than a physical hardware implementation, it will allow us to simulate complex and hierachically-structured systems before the available hardware technology allows it and without hardware cost. The AER-based simulation tool can be used to test new AER processing modules within large systems, and thus orient hardware developers on what kind of AER hardware modules may be useful and what performance characteristics they should possess. We have presented three implementations to validate our tool and the results show clearly the high speed and possibility of implementing complex processing systems that AER provides. With the three AER-based implementations we try to demonstrate the feasibility of AER technology when it is applied to real-time image processing. AER is able to process input stimulus in real time and to transmit the resulting activity in each layer to the following layers even without having finished collecting all the input events. It is also feasible to assemble multiple chips working in parallel so that complex processing and multiple convolutions can be done. The first two applications were intented to detect and track objects with different shape and size in real time. It can be observed that in the two implementations the processing has been always in real time and that the outputs were equal to those obtained with the hardware devices. The third application is an example of a single layer neural network to recognize hand-written digits in the MNIST database. Available AER devices allow us to implement simple but real-time architectures like the single layer neural network and with the times computed. In future implementations we want to develop more sophisticated and cortical-like multi-layer systems where the link between hardware AER implementations, bio-inspired processing and frame-based applications will become more apparent.

Acknowledgement

This work was supported in part by grant TEC-2006-11730-C03-01 (Samanta2) from the Spanish Ministry of Education and Science and grant P06-TIC-01417 (Brain System) from the Andalusian regional government. JAPC was supported by a doctoral scholarship as part of research project Brain System.

References

1. Shepherd, G.M.: The Synaptic Organization of the Brain, 3rd edn. Oxford University Press, Oxford (1990)
2. Rolls, E.T., Deco, G.: Computational Neuroscience of Vision. Oxford University Press, Oxford (2002)
3. LeCun, Y., Bengio, Y.: Convolutional Networks for Images, Speech, and Time Series. In: Arbib, M. (ed.) The Handbook of Brain Science and Neural Networks, pp. 255–258. MIT Press, Cambridge (1995)
4. Fasel, B.: Robust Face Analysis using Convolution Neural Networks. In: Proc. of the Int. Conf. on Pattern Recognition (ICPR 2002), Quebec, Canada (2002)
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. Mahowald, M.: VLSI Analogs of Neural Visual Processing: A Synthesis of Form and Function, Ph.D. Thesis, California Institute of Technology, Pasadena CA (1992)
7. Cauwenberghs, G., Kumar, N., Himmelbauer, W., Andreou, A.G.: An analog VLSI Chip with Asynchronous Interface for Auditory Feature Extraction. IEEE Trans. Circ. Syst. Part-II 45, 600–606 (1998)
8. Oster, M., Liu, S.-C.: Spiking Inputs to a Spiking Winner-Take-All Circuit. In: Weiss, Y., Schölkopf, B., Platt, J. (eds.) Advances in Neural Information Processing Systems (NIPS 2006), vol. 18, pp. 1051–1058. MIT Press, Cambridge (2006)
9. Lichtsteiner, P., Delbrück, T.: 64x64 AER Logarithmic Temporal D rivative Silicon Retina. Research in Microelectronics and Electronics 2, 202–205 (2005)
10. Serrano-Gotarredona, R., et al.: AER Building Blocks for Multi-Layers Multi-Chips Neu-romorphic Vision Systems. In: Weiss, Y., Schölkopf, B., Platt, J. (eds.) Advances in Neural Information Processing Systems (NIPS 2006), vol. 18, pp. 1217–1224. MIT Press, Cambridge (2006)
11. Serrano-Gotarredona, R., Serrano-Gotarredona, T., Acosta-Jiménez, A., Linares-Barranco, B.: A Neuromorphic Cortical Layer Microchip for Spike Based Event Processing Vi-sion Systems. IEEE Trans. on Circuits and Systems, Part-I 53(12), 2548–2566 (2006)
12. Serrano-Gotarredona, R., et al.: On Real-Time AER 2D Convolutions Hardware for Neu-romorphic Spike Based Cortical Processing. IEEE Trans. on Neural Networks 19(7), 1196–1219 (2008)
13. Serre, T., Wolf, L., Bileschi, S., Riesenhuber, M., Poggio, T.: Object recognition with cortex-like mechanisms. IEEE Transactions on Pattern Analysis and Machine Intelligence 29(3), 411–426 (2007)
14. Masquelier, T., Thorpe, S.J.: Unsupervised Learning of Visual Features through Spike Timing Dependent Plasticity. PLoS Comput. Biol. 3(2), e31 (2007)
15. The MNIST database, `http://yann.lecun.com/exdb/mnist/index.html`
16. 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), 771–788 (2006)
17. Le Cun, Y., et al.: Backpropagation applied to handwritten zip code recognition. Neural Computation 1, 541–551 (1989)
18. Le Cun, Y., et al.: Gradient-based learning applied to document recognition. Proceedings of the IEEE 86(11), 2278–2324 (1998)