High-Speed Character Recognition System based on a complex hierarchical AER architecture

J. A. Pérez-Carrasco, T. Serrano-Gotarredona, C. Serrano-Gotarredona, B. Acha, and B. Linares-Barranco

Inst. Microelectrónica de Sevilla (IMSE-CNM-CSIC)

Dpto. Teoría de la Señal, ETSIT, Universidad de Sevilla

Abstract

In this paper we briefly summarize the fundamental properties of spikes 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 cortical-like layers for sophisticated processing. The paper describes briefly cortex-like spiking vision processing principles, and the AER (Address Event Representation) technique used in hardware spiking systems. Afterwards an example application is described, which is a simplification of Fukushima's Neocognitron. Realistic behavioral simulations based on existing AER hardware characteristics, reveal that the simplified neocognitron, although it processes 52 large kernel convolutions, is capable of performing recognition in less than 10μs.

I. Introduction

Artificial man-made machine vision systems operate in a quite different way than biological brains. Machine vision systems usually operate by capturing and processing sequences of frames, which are then processed frame by frame to extract, enhance and combine features, and perform operations in feature spaces, until a desired recognition is achieved. 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. This activity level may respond to different image properties like intensity, contrast, motion, etc. - which have been pre-computed within the retina before generating the spikes to be sent to the visual cortex. Very active pixels will send more spikes than less active pixels. When the retina responds to a stimulus, those pixels sensing the profile will elicit a simultaneous collection of spikes which are strongly space-time correlated. The visual cortex receiving these spikes is sensitive to the space location where the spikes were originated and to the relative timing between them. This way, it can recognize and follow this moving profile. 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. This way, in biological brains, strong features are propagated and processed through projection-fields from layer to layer as soon as they are produced, without waiting to finish collecting and processing data of whole image frames.

In general, projection-fields in biological neuro-cortical layers perform feature extraction operations, which are dependent on the "shape" (weights) of the projection-field connections. Note that projection-field processing is equivalent to convolution processing, where the kernel of the convolution is the projection-field shape [13]. In biological neuro-cortical structures there are several (8-10) sequential projection-field layers that extract features [1]-[3], group them, extract more elaborated features, and so on, until in the end they perform complicated recognition tasks, such as handwritten character recognition [4]-[5] or face recognition [6]-[7].

A very interesting and powerful property of the spike-based projection-field processing is its high speed. The spikes from one layer are sent simultaneously to the following layer through projection-fields. Consequently, this spike based projection-field processing approach is structurally much faster than a conventional frame-based processing approach. In a frame-based approach all pixels in a retina (or camera) are sensed and transmitted to the next layer (or processing stage), where all pixels of the frame are processed, usually with convolution operations, and so on. This frame convolution processing is slow, specially if several convolutions need to be computed in sequence for each input image frame.

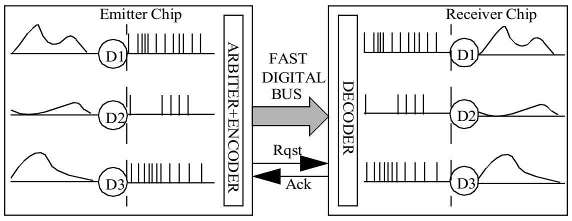


Fig. 1. Point-to-point AER link

Artificial spike based processing hardware systems usually exploit the AER (Address-Event-Representation) protocol.

II. The AER Protocol

Fig. 1 illustrates the communication in a traditional point-to-point AER link [10]. The neurons of an emitter chip communicate their state to the corresponding neurons in a receiver chip. The continuous-time state of the emitter neurons is transformed into a sequence of fast digital pulses. The pulses generated by the emitter neurons are time-multiplexed in a common output digital bus. Each neuron is identified with an address. Each time a neuron emits a pulse that neuron address appears in the output digital bus, together with standard four-phase handshake signals for request (Rqst) and acknowledge (Ack). This is called an "Address Event".

The receiver chip reads and decodes the addresses of the incoming events and sends pulses to the corresponding receiving neurons. The receiving neurons integrate those pulses and are able to reproduce the state of the corresponding emitter neurons.

This is the simplest AER-based inter-chip communication. However, this point-to-point communication can be extended to a multi-receiver scheme [9]. Also, multiple emitters can merge their outputs into a smaller set or receiver chips [13]. Moreover, AER visual information can be easily translated or rotated by remapping the addresses during the inter-chip transmission [8],[11].

III. Illustration of Multi-Chip Multi-Layer Convolution Processing

To illustrate the potential of AER modules based spiking systems, and how AER chips and modules can be used in a multi-chip multi-layer AER structure, we will show in this Section behavioral simulations of a small system made of many AER convolution chips like the one reported in [12],[13]. At present we don't have yet the hardware infrastructure to illustrate the behavior of such kind of multi-layer systems experimentally, but we do have a Matlab toolbox to emulate behaviorally the operation and timing of such systems. We have used this toolbox to emulate the behavior of the system shown in Fig. 2, using delays similar to those of our AER experimental chips. This system is loosely based on the neocognitron architecture used for handwritten character recognition [14], although it is a very simplistic version, which we have adapted here so that it can distinguish between characters 'A', 'B', 'C', 'H', 'L', 'M', and 'T'. As shown in Fig. 2, it receives an input visual stimulus (of 16×16 pixels), which can be one of the previous letters, and it can tolerate slight deformations. Each active pixel of the 16x16 input stimulus will fire ten events, and the rest of pixels will not fire. Input events will be separated

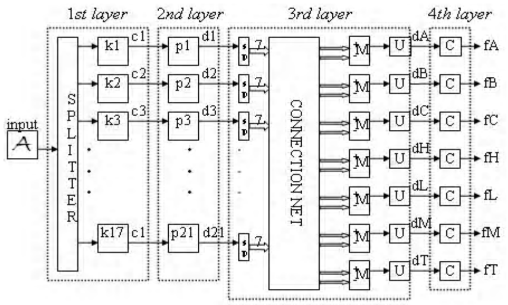


Fig. 2: Illustration of a multi-chip multi-layer AER convolution processing system to distinguish between handwritten characters 'A', 'B', 'C', 'H', 'L', 'M' and 'T'.

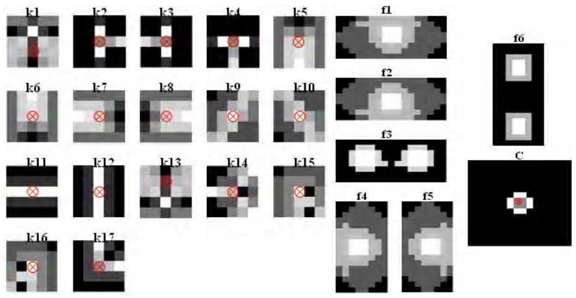


Fig. 3: Kernels used for the different convolutions in Fig. 2.

50*ns*. This way, the complete input stimulus which has around 30 active pixels, will be transmitted in about 15μ*s*.

The first processing layer performs 17 convolutions in parallel, of kernels *ki* (*i* = 1 to 17). These are shown on the left of Fig. 3 normalized from '0' to '1'. The 'x' in the kernels shows the event coordinate around which the kernel is copied in the pixel array. In the system of Fig. 2, each convolution chip is configured to not send out any negative event[1]. Only positive events will be transmitted. Consequently, each convolution chip will compute a half wave rectification, besides the programmed convolution.

Kernel *k*1 is intended to detect the presence and position of the upper peak in letter 'A'. Kernel *k*2 detects a horizontal segment ending on the left and touching a vertical segment. Kernel *k*3 the same, but ending on the right. Kernel *k*4 detects a vertical segment ending on the top and touching a horizontal segment. Kernel *k5* detects the bottom end of a vertical segment, kernel *k6* the same but for the top end. Kernel *k7* detects the left end of a horizontal segment, kernel *k8* the same but for the right end. Kernel *k9* is intended to detect the upper curvature of letter 'C' and kernel *k10* the same but for the lower one. Kernel *k11* detects a horizontal segment and kernel *k12* the same but for a vertical one. Kernel *k13* is intended to detect the central crossing point between the two inclined segments of letter 'M', kernel *k14* detects the crossing point between the two right curves in letter 'B'. Kernel *k15* the upper left peak in letter 'M', kernel *k16* the same, but on the right and finally, kernel *k17* detects the crossing point between the horizontal and vertical segments in letter 'L'. Consequently, the first layer of convolutions is intended to detect a set of 17 geometrical features which can be used to detect and discriminate between the different letters. Letter 'A' should produce activity at outputs {c1, c2, c3, c5, c11}, letter 'B' at {*c2, c11, c12, c14, c17*}, letter 'C' at {*c8, c9, c10, c11, c12*}, letter 'H' at {*c2, c3, c5, c6, c11, c12*}, letter 'L' at {*c6, c8, c11, c12, c17*}, letter 'M' at {*c5, c12, c13, c15, c16*} and letter 'T' at {*c4, c5, c7, c8, c11, c12*}.

The second layer of convolution processing performs 21 convolutions in parallel *pi* (i=1 to 21). There are only six different kernels in this layer, *fi* (i=1 to 6), which are shown on the right of Fig. 3 normalized from '0' to '1'. Each convolution

		COORDINATE				COORDINATE	
FILTER	KERNEL	X-COORD	Y-COORD	FILTER	KERNEL	X-COORD	Y-COORD
p1	f1	10	9	p12	f1	3	9
p2	f4	8	7	p13	f1	9	9
p3	f5	8	2	p14	f1	4	12
p4	f1	9	9	p15	f1	2	9
p5	f3	-3	9	p16	f1	2	5
p6	f3	10	8	p17	f1	5	9
p7	f5	10	7	p18	f1	3	7
p8	f5	10	0	p19	f2	10	13
p9	f1	7	11	p20	f2	10	5
p10	f1	0	11	p21	f1	-2	12
p11	f1	-2	8				

Table I. Kernels used in layer '2' and 'x' coordinate around which kernel is copied in the pixel array.

pi on layer '2' makes use of one of these six kernels *fi* (i=1 to 6). The kernel that each filter *pi* uses and the event coordinate around which the kernel is copied in the pixel array are shown in Table I. This layer is intended to evaluate whether the spatial distribution of features detected in the first layer is meaningful for the character to be detected. For example, for letter 'A', the top peak (detected by *k*1) should be in the upper part above all other features. Consequently, kernel *p*1 will produce a positive contribution in the region below, because this would be the place in output *d1* where the center of letter 'A' would be if all its features are detected simultaneously. In a similar manner, if there is output activity at *c2*, the center of 'A' could be to the right. Therefore, kernel *p*2 will add contribution to the pixels in *d2* which are to the right of those who fired in *c2*. The output at *c3* has to be treated symmetrically than the one for *c2*. Kernel *k5* places events at *c5* if a bottom end of vertical segment is detected. This means that the center of letter 'A' is somewhere above, either to the right or to the left. This spatial weighting is performed by kernel *p5*. Finally, if there is output at *c11*, the center of 'A' should be in the same position. Therefore, kernel *p11* will add contribution to the pixels in *d11* which are at the same position of those who fired in *d11*. In this way, when the input is letter 'A' the activity at {*c1, c2, c3, c5, c11*} will be on different pixels. However, the activity at {*d1, d2, d3, d5, d11*} would be around the center of letter 'A'. Something similar will occur with the rest of letters: their center will be identified also with the respective outputs obtained from kernels in layer '2', which will perform the spatial weighting needed for each of the different features extracted in the first layer.

The purpose of the third layer is to add with positive or negative weight the outputs of the second layer. For example, for letter 'A' outputs {*d1, d2, d3, d5, d11*} should contribute positively, while outputs {*d4, d6, d7, d8, d9, d10, d12, d13, d14, d15, d16, d17, d18, d19, d21*} should inhibit. Similarly will occur with the rest of letters. Consequently, all outputs *d1-d21* are split (blocks 'Sp' in Fig. 2) into seven separate pathways with seven independent 21-input merger blocks, each one to detect one of the letters. Only positive events come out at outputs *d*1-*d*21. However, the sign bits are hardwired at the inputs of the merger blocks, with positive sign if the events contribute positively or negative sign if the events contribute negatively. The merger blocks simply sequence the events coming from their 21 input channels, and feed them to a convolution chip programmed with a 1x1 kernel with weight 1. The convolution chips parameters are set so that 3 input events will be necessary to produce an output event for one pixel.

Finally, the fourth layer consists of one single convolution chip for each character path, programmed with kernel *C*, which will detect whether the events coming from the previous layer are more or less clustered together, rather than spread over the pixel array. If they are clustered, it means the character has been detected. This kernel is shown on the bottom right of Fig. 3 normalized from '0' to '1'. The 'x' in the kernel *C* of Fig. 3 shows the event coordinate around which the kernel is copied in the pixel array.

The system shown in Fig. 2 has been tested using three slightly modified versions of each one of the seven proposed letters. The twenty-one characters are shown together in Fig. 4. The results obtained after testing the system are shown in Table II. The output events generated at the different convolution outputs {*c1, c2, c3, c5, c11, d1, d2, d3, d5, d11, dA, dB, dC, dH, dL, dM, dT, fA, fB, fC, fH, fL, fM, fT*} are shown in Fig. 5 for the

1. If a pixel produces a negative output event, the pixel is reset, but it does not transmit the event out of the chip.

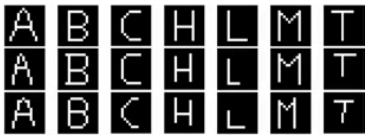


Fig. 4: Three different versions of the seven letters under analysis.

					OUTPUT ACTIVITY (NUMBER OF EVENTS)						
INPUT LETTER	FRAME TIME(μs)	RETRIEVAL ACCURACY	TIME FIRST OUTPUT (μs)	TIME LAST OUTPUT (μs)	fA	fB	fC	fH	fL	fM	fT
A1	15	100	7,45	33,23	69	0	11	0	0	0	0
A2	12,95	100	11,3	27,15	28	0	6	1	0	1	0
A3	14,45	100	7,97	32,02	72	0	4	0	0	0	0
B1	17,95	100	14,64	37,08	3	43	0	12	0	8	0
B2	19,95	100	16,58	37,91	5	21	0	9	0	0	0
B3	17,45	100	17,15	40,07	0	36	9	1	1	8	0
C1	9,95	100	7,74	25,9	0	8	72	9	12	0	0
C2	10,95	100	7,65	21,18	0	19	76	16	34	0	7
C3	7,95	100	7,71	28,44	1	0	33	0	0	0	0
H1	14,95	100	5,97	48,62	0	0	0	49	8	0	0
H2	12,95	100	14,76	29	1	0	0	16	12	4	0
H3	12,45	100	16,23	45,32	0	0	0	22	9	1	0
L1	9,45	100	5,44	23,88	0	12	0	8	47	0	0
L2	6,95	100	6,72	23,62	0	8	0	0	32	0	0
L3	6,45	100	7,65	25,26	0	0	12	7	31	0	0
M1	15,45	100	5,41	27,83	0	1	0	9	0	83	0
M2	14,95	100	9,19	27,53	0	0	0	11	1	33	0
M3	13,95	100	5,74	38,68	0	0	0	0	6	50	0
T1	9,95	100	7,11	11,04	6	1	0	5	0	0	23
T2	8,45	100	7,21	14,19	7	10	0	7	0	0	18
T3	7,95	100	5,93	20,26	0	1	0	4	0	0	23
AVERAGE	12,40	100,00	9,31	29,44							

Table II. Timing and number of events obtained after the system simulation.

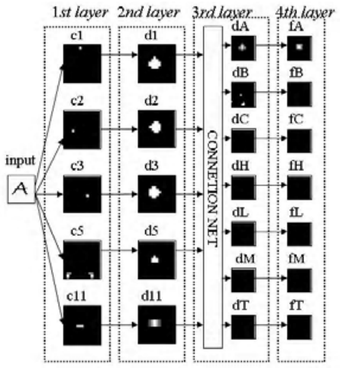


Fig. 5. Output events generated at the different convolution outputs {c1, c2, c3, c5, c11, d1, d2, d3, d5, d11, dA, dB, dC, dH, dL, dM, dT, fA, fB, fC, fH, fL, fM, fT} for the case of input stimulus 'A'.

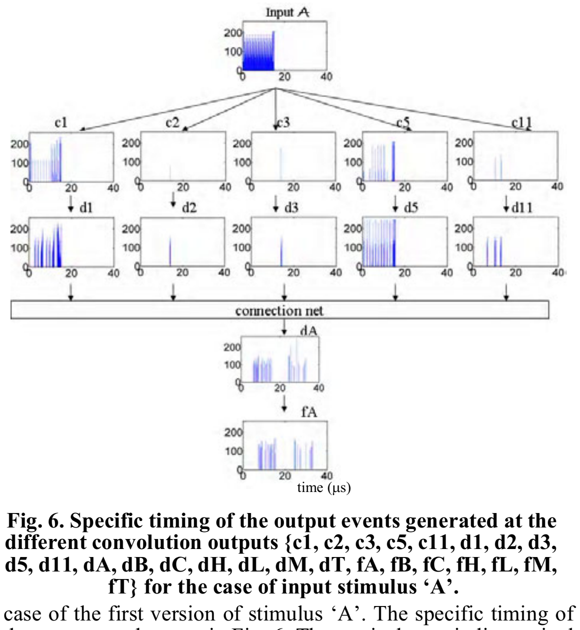
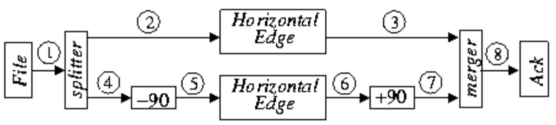


Fig. 6. Specific timing of the output events generated at the different convolution outputs {c1, c2, c3, c5, c11, d1, d2, d3, d5, d11, dA, dB, dC, dH, dL, dM, dT, fA, fB, fC, fH, fL, fM, fT} for the case of input stimulus 'A'.

case of the first version of stimulus 'A'. The specific timing of the events can be seen in Fig. 6. The vertical axes indicate pixel numbers (from 0 to 255) in the 16x16 pixel arrays, while the horizontal axes represent time in μ*s* (from 0 to 40μ*s*). As can be seen, the system is capable of detecting which letter is present, in about 9,3μ*s* after the first input stimulus event is received.



```
%First, we declare sources to the system
%                SOURCES           SOURCES DATA
sources            {1}                  {s1}

%Next, we declare priorities
priorities     {0.9 0.8 0.7 0.6 0.5 0.4 0.3 0.2}

%Next, we declare blocks
%NAME                IN-CHANNELS  OUT-CHANNELSPARAMS      STATES
splitter             {1}          {2,4}    {params1}      {state1}
h_sobel              {2}          {3}      {params2}      {state2}
imrotate90           {4}          {5}      {params3}      {state3}
h_sobel              {5}          {6}      {params4}      {state4}
imrotate-90          {6}          {7}      {params5}      {state5}
merger               {3,7}        {8}      {params6}      {state6}
ack_only             {8}          {8}      {params7}      {state7}
```

Fig. 7: (top) Example block diagram of AER system. (bottom) Netlist file describing it.

IV. Simulator Description

In this simulator any generic AER system is described by a netlist that uses only two types of elements: *instances* and *channels*. An *instance* is a block that processes and/or produces AER streams. A convolution chip [12] would be an AER processing instance with an input AER stream and an output AER stream. A splitter [13] would be an instance which replicates the events from one input AER stream onto several output AER streams. Similarly, a merger [13] is another instance which would receive as input several AER streams and merge them into a single output AER stream. AER streams constitute the nodes of the netlist in an AER system, and are called *channels*. The simulator imposes the restriction that a *channel* connects a single AER output from an instance to a single AER input of another (or the same) instance. This way, channels represent point-to-point connections. For splitting and/or merging channels splitters and/or merger instances must be included in the netlist.

Fig. 7 shows an example netlist and its ASCII file netlist description. The netlist contains 7 instances and 8 channels. The netlist description is provided to the simulator through a text file, which is shown in the bottom of Fig. 7. Channel 1 is a source channel. All its events are available a priori as an input file to the simulator. There can be any arbitrary number of source channels in the system. Each source channel needs a line in the netlist file, starting with key word *sources*, followed by the channel number and the file containing its events. A line starting with key word *priorities* provides a priority number for each channel. Its use will be explained later. 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 structure containing its parameters, and name of structure containing its state. Each instance is described by a MATLAB function whose name is the name of the instance. The simulator imposes no restriction on the format of the parameters and states 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 MATLAB two dimensional matrixes. Each row in the matrix corresponds to one event. Each row has six components

$$[x, y, \text{sign}, T_{preRqst}, T_{Rqst}, T_{Ack}] \qquad (1)$$

'x' and 'y' represent the coordinates or addresses of the event and 'sign' its sign. '$T_{preRqst}$' represents the time at which the event is created at the emitter instance, 'T_{Rqst}' represents the time at which the event is processed by the receiver instance, and 'T_{Ack}' represents the time at which the event is finally acknowledged by the receiver instance. We distinguish between a pre-Request time $T_{preRqst}$ and an effective Request time T_{Rqst}. 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 $T_{preRqst}$ times. Once the events are processed by the simulator, their 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 file. 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. An event is unprocessed when only its pre-Request time $T_{preRqst}$ has been established, but not its final Request time T_{Rqst} nor its acknowledge time T_{Ack}. In case several channels have events with the same pre-Request time, they will be processed according to the channels priority information given in the netlist file (see Fig. 7). The channel with the highest priority number will be processed first.
2. Once a channel is selected, its earliest unprocessed event information is provided as input to the instance this channel is connected to. The instance updates its internal state. In case this event triggers new output events from this instance, a list of new unprocessed events is provided as output. This list of new events provides the events information and their pre-Request time $T_{preRqst}$ only. Then the simulator updates all channels, stores the new state for this instance and goes back to 1.

The operation of an instance is described by an independent MATLAB function using the instance name in the netlist file. A user can add and write new instances as desired. The only restriction is to respect the calling format of the function

$$\begin{aligned}&[\text{Event\_In, Events\_Out, State\_New, Time\_Out}] = \\&= \mathit{function}(\text{Event\_In, Params, State, Time\_In})\end{aligned} \quad (2)$$

'Event_In' corresponds to the present event being processed by the channel (see eq. (1)). The 'Event_In' passed to the function as input parameter contains the *'x'* and *'y'* coordinates of the event being processed and its pre-Request time $T_{preRqst}$. The updated 'Event_In' returned by the function contains also the established Request T_{Rqst} and Acknowledge times T_{Ack}. 'State' and 'State_New' represent the instance state before and after processing the event. 'Time_In' and 'Time_Out' are the global system times before and after the event processing. And 'Events_Out' is a list of output events produced by the instance at its different output channels. These unprocessed output events are included by the simulator in their respective channel matrixes, which at a later time should be processed by their respective destination instances.

V. Conclusions

In this paper, we have demonstrated how to build complex cortex-like artificial spiking vision processing systems based on the AER protocol. As an example, a simplified multi-chip (52 chips) multi-layer (4 layers) character recognition system has been simulated to illustrate how one can interconnect and program AER convolution chips and what timing can be expected from the system. Other characters could be analysed adding new convolutions chips or simply doing different merging of their outputs. Besides the convolution chips, one also requires splitters and/or merger blocks, and eventually some extra mappers. In the future, as AER processors become more sophisticated, we expect to be able to fit in a single chip several convolution arrays together with splitters/mergers and mappers. In the example of Fig. 2, all convolution chips are identical and 16x16 sized. The system could be conceived to include learning, so that an external supervisor trains it and updates the weights dynamically, optimizing final performance [14]. In any case, the recognition performance rate for this system (keep in mind that it tolerates a certain degree of letter deformation and scaling) is unprecedented (9,31μs is equivalent to over 100000 images per second). We believe that this technique for bio-inspired vision processing is very promising.

One limitation that can be argued for this vision processing technique is that for real size images much more events need to be processed. This is certainly true. However, our experience is that if one uses appropriate input sensors, like retinae that directly sense motion [15] or contrast [16]-[18] instead of image intensity, the flow of events is kept at reasonable rates (below 1*Mev/sec* for arrays of 128x128 pixels [15]).

VI. Acknowledgements

This work was partially supported by Spanish grants TIC-2003-08164-C03-01 (Samanta), TEC-2006-11730-C03-01 (Samanta2) and EU grant IST-2001-34124 (Caviar). JAPC was supported by the Andalusian government grant P06-TIC-01417 (Brain System).

VII. References

[1] H. Fujii, H. Ito, K. Aihara, N. Ichinose, and M. Tsukada, "Dynamical Cell Assembly Hypothesis - Theoretical Possibility of Spatio-Temporal Coding in the Cortex," *Neural Networks*, vol. 9, pp. 1303-1350, 1996.

[2] G. A. Orban, *Neural Operations in the Visual Cortex*, Springer-Verlag, Berlin, 1984.

[3] G. M. Shepherd, *The Synaptic Organization of the Brain*, Oxford University Press, 3rd Edition, 1990.

[4] K. Fukushima: ''Visual feature extraction by a multilayered network of analog threshold elements,'' *IEEE Transactions on Systems Science and Cybernetics*, SSC-5 (4), pp. 322-333, Oct. 1969.

[5] K. Fukushima, "Analysis of the process of visual pattern recognition by the neocognitron," *Neural Networks,* vol. 2, pp. 413-420, 1989.

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

[7] M. Browne and S. S. Ghidary, "Convolutional Neural Networks for Image Processing: An Application in Robot Vision," *Advances in Artificial Intelligence: 16th Australian Conf. on AI*, pp. 641-652, November 2003.

[8] T. Serrano-Gotarredona, A. G. Andreou, B. Linares-Barranco, "AER Image Filtering Architecture for Vision-Processing Systems," *IEEE Transactions on Circuits and Systems, Part I*, vol. 46, No. 9, pp. 1064-1071, September 1999.

[9] J. Lazzaro, J. Wawrzynek, M. Mahowald, M. Sivilotti, and D. Gillespie, "Silicon Auditory Processors as Computer Peripherals," *IEEE Transactions on Neural Networks,* vol.4, No 3, pp. 523-528, May, 1993.

[10]K. Boahen, "Point-to-Point Connectivity Between Neuromorphic Chips Using Address Events," *IEEE Trans. on Circuits and Systems Part-II*, vol. 47, No. 5, pp. 416-434, May 2000.

[11]D. H. Goldberg, G. Cauwenberghs, and A. G. Andreou, "Probabilistic Synaptic Weighting in a Reconfigurable Network of VLSI Integrate-and-Fire Neurons," *Neural Networks,* vol. 14, pp. 781-793, 2001.

[12]R. Serrano-Gotarredona, T. Serrano-Gotarredona, A. Acosta-Jiménez, and B. Linares-Barranco, "A Neuromorphic Cortical Layer Microchip for Spike Based Event Processing Vision Systems," *IEEE Trans. on Circuits and Systems, Part 1*, vol. 53, No. 12, pp. 2548-2566, December 2006.

[13]R. Serrano-Gotarredona, *et al*., "AER Building Blocks for Multi-Layers Multi-Chips Neuromorphic Vision Systems" *Advances in Neural Information Processing Systems*, vol. 18, Y. Weiss and B. Schölkopf and J. Platt (Eds.), (NIPS'06), MIT Press, Cambridge, MA, pp. 1217-1224, 2006.

[14]K. Fukushima and N. Wake, "Handwritten Alphanumeric Character Recognition by the Neocognitron," *IEEE Trans. Neural Networks*, vol. 2, No. 3, pp. 355-365, May 1991.

[15]P. Lichtsteiner, C. Posch, and T. Delbruck, "A 128x128 120dB 30mW Asynchronous Vision Sensor that Responds to Relative Intensity Change," *2006 IEEE ISSCC Digest of Technical Papers*, pp. 508-509, San Francisco, 2006.

[16]K. A. Zaghloul and K. Boahen, "Optic nerve signals in a neuromorphic chip: Parts 1," *IEEE Trans.Biomed Eng.*, vol. 51, pp. 657-666, 2004.

[17]K. A. Zaghloul and K. Boahen, "Optic nerve signals in a neuromorphic chip: Part 2," *IEEE Trans.Biomed Eng.*, vol. 51, pp. 667-675, 2004.

[18]J. Costas-Santos *et al*., "A Contrast Retina with On-chip Calibration for Neuromorphic Spike-Based AER Vision Systems," *IEEE Trans. Circ. and Syst., Part-I,* to be published in 2007.