On Real-Time AER 2-D Convolutions Hardware for Neuromorphic Spike-Based Cortical Processing

Rafael Serrano-Gotarredona, Teresa Serrano-Gotarredona, *Member, IEEE*, Antonio Acosta-Jiménez, Clara Serrano-Gotarredona, José A. Pérez-Carrasco, Bernabé Linares-Barranco, Alejandro Linares-Barranco, Gabriel Jiménez-Moreno, and Antón Civit-Ballcels

***Abstract*—In this paper, a chip that performs real-time image convolutions with programmable kernels of arbitrary shape is presented. The chip is a first experimental prototype of reduced size to validate the implemented circuits and system level techniques. The convolution processing is based on the address–event-representation (AER) technique, which is a spike-based biologically inspired image and video representation technique that favors communication bandwidth for pixels with more information. As a first test prototype, a pixel array of 16×16 has been implemented with programmable kernel size of up to 16×16. The chip has been fabricated in a standard 0.35-μm complimentary metal–oxide–semiconductor (CMOS) process. The technique also allows to process larger size images by assembling 2-D arrays of such chips. Pixel operation exploits low-power mixed analog–digital circuit techniques. Because of the low currents involved (down to *nanoamperes* or even *picoamperes*), an important amount of pixel area is devoted to mismatch calibration. The rest of the chip uses digital circuit techniques, both synchronous and asynchronous. The fabricated chip has been thoroughly tested, both at the pixel level and at the system level. Specific computer interfaces have been developed for generating AER streams from conventional computers and feeding them as inputs to the convolution chip, and for grabbing AER streams coming out of the convolution chip and storing and analyzing them on computers. Extensive experimental results are provided. At the end of this paper, we provide discussions and results on scaling up the approach for larger pixel arrays and multilayer cortical AER systems.**

***Index Terms*—Address–event representation (AER), analog circuits, asynchronous circuits, bioinspired systems, cortical layer processing, image convolutions, image processing, low power circuits, mixed-signal circuits, spike-based processing.**

This work was supported by the European Commission under Grants IST-2001-34124 (CAVIAR) and 216777 (NABAB), the Spanish Ministry of Education and Science under Grants TIC-2000-0406-P4 (VICTOR), TIC-2003-08164-C03-01 (SAMANTA), and TEC2006-11730-C03-01 (SAMANTA2), and the Junta de Andalucia under Grant TIC-1417 (Brain System). The work of R. Serrano-Gotarredona was supported by the Spanish Ministry of Education and Science under the FPU scholarships. The work of J. A. Pérez-Carrasco was supported by a scholarship from Junta de Andalucia.

R. Serrano-Gotarredona was with the CSIC Instituto de Microelectronica Sevilla (IMSE–CNM), 41012 Sevilla, Spain. He is now with the R&D Department, NXP, Graz, Austria.

T. Serrano-Gotarredona, A. Acosta-Jiménez, C. Serrano-Gotarredona, J. A. Pérez-Carrasco, and B. Linares-Barranco are with the CSIC Instituto de Microelectronica Sevilla (IMSE–CNM), 41012 Sevilla, Spain (e-mail: bernabe@imse.cnm.es).

A. Linares-Barranco, G. Jiménez-Moreno, and A. Civit-Ballcels are with the Departamento de Arquitectura y Tecnología de Computadores, ETSI Informática, Universidad de Sevilla, 41012 Sevilla, Spain.

I. Introduction

STANDARD image transmission and processing is done using frame-based techniques. Since the invention of cinema, television, and even for modern computer-based motion pictures, a video stream is formed by a sequence of still frames. If some kind of image processing is required for enhancing the image, or extracting recognition features out of it, complex processing and recognition algorithms are performed on individual frames. If all the required processing can be performed within the frame rate timing of the video stream (usually 30–40 ms per frame for commercial video), then we can say vision processing is performed in real time. However, many of today's known algorithms capable of performing complex image processing tasks for vision require tremendous amount of computing power, impossible to be achieved in real time with today's most sophisticated computers. For example, consider the boundary contour system–feature contour system (BCS–FCS) image segmentation software algorithm [1] (see, also, [30, Fig. 1]). A captured image is processed by a sequence of eight convolution layers. These convolutions have different kernels. Each layer consists of several sublayers in parallel, because from one layer to the next the same kernel is applied but with different spatial characteristics (for example, rotated for different angles). The structure includes a feedback path: the output of the last layer is fed back to an intermediate layer. The whole structure consists of different identical structures that run in parallel. The only difference between them is the spatial scale of the kernels in the convolution operations. If the complete BCS–FCS algorithm is to be performed on a computer, then for each input frame, all convolution operations need to be iterated until the feedback settles to a steady-state solution at all layers. Consequently, implementing such an algorithm in real time with frame-based convolution processing is completely unfeasible with today's computer technology. The creators of the BCS–FCS software algorithm talk about processing times of several hours for a single image. On the other hand, the spike-based hardware convolution technique presented in this paper makes it realistically plausible. In a hardware spike-based processing system, pixels at the input image sensor send spikes (events) as soon as they detect activity (intensity, contrast, motion, etc.). These spikes are very short in time (typically, tens of nanoseconds). Consequently, the most relevant pixels send a spatio–temporal wavefront of spikes in a very short time (microseconds), containing the most salient information of an input visual stimulus. This is consistent with discoveries of fast human visual processing [2]. As soon as they are produced, these spikes can travel very quickly (nano-

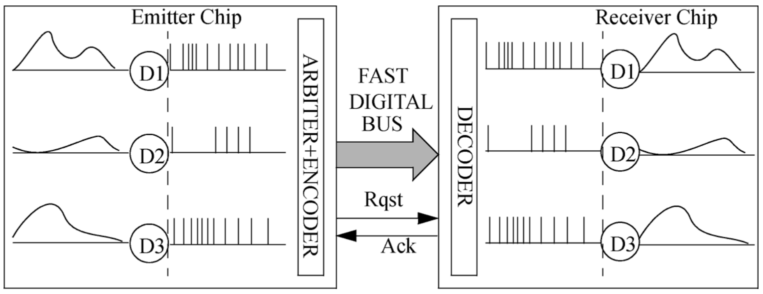


Fig. 1. Illustration of AER point-to-point communication.

to microseconds) through a complex hardware structure of processing layers performing very fast detection and recognition of the most salient features of an input visual stimulus.

Other examples of complicated software vision processing systems based on convolutions are the so-called convolution neural networks [6]–[10]. First realization of such a system was already proposed by Fukushima in 1980 [4]. Software convolution neural networks have been successfully applied for character recognition [4]–[8], object detection [7], and face recognition [8]–[10]. Reported kernel sizes go up to 11×11 for 64×64 pixel images.

This is for software algorithms running on conventional computers. For more dedicated hardware, today's hardware solutions for real-time convolution processing rely entirely on individual frame-by-frame computations. In the literature, one may find solutions based on digital signal processors (DSP) or specialized parallel architectures like simple instruction multiple data (SIMD) processors. The vast majority of examples usually perform convolutions of small-size kernels (3×3), which require computation power in the order of 30 Gops/s (gigaoperations per second)[1] to achieve rates of 30 frames/s [11]. A clever DSP architecture specialized for large kernel convolutions was presented in 1999 by Wall *et al.* [12]. It handles kernels of up to 15×15 and can compute the convolution of one single 256×256 image in about 55 ms. The processing speed grows quadratically with image dimension, but linearly with kernel size. For DSP type of implementations, algorithmic tricks can be used to optimize speed (or power) [13]. On the other hand, SIMD processors are array-based microchips, where each pixel is prepared to perform a certain amount of operations among local data available on each pixel. All pixels perform the same operation on a given clock cycle, but each pixel operates on its local data. Pixels are connected to nearest neighbors and data can easily be transferred between them. It is very simple to perform 3×3 convolutions with them, and it is not impossible to perform larger kernel convolutions but at the expense of complicated sequencing and data transfers, which degrade speed rapidly with kernel size. Many SIMD architectures have been proposed in the literature [14], [15]. For example, the SIMPiL system reported in 1996 could perform 3×3 kernel convolutions on 256×256 images at rates of 60 frames/s. Other architectures tailored for convolution processing have been proposed by Etienne–Cummings [16], where convolutions are computed in a clever way during sequential readout of the video images.

[1]This refers to elementary operations such as additions and subtractions.

However, all these proposals are based on frame-by-frame processing techniques. In this paper, we rely on spiking techniques which do not process frames. Consequently, relevant image features will be communicated and processed first, resulting in extremely high-speed processing throughput.

The BCS–FCS algorithm [1], as well as the convolution neural networks [6]–[10], are claimed by the authors to be bioinspired. This is because the processing in the brain is supposed to be based on *projection-field* operations between consecutive layers [17]. A projection-field operation means that one cell (neuron or pixel) in one layer connects to an ensemble of cells in the next layer. Each connection is characterized by a weight. If the spatial distribution of weights going out of neuron i in a layer with position (x_i, y_i) is the same as for any other neuron j in the same layer with position (x_j, y_j), then the processing performed from this layer to the next is a convolution operation. The kernel of this convolution is precisely the spatial distribution of the connection weights for one neuron. But then, how can living brains implement multilayer projection-field algorithms in real time? The key issue is that the brain does not acquire images by frames and performs convolutions on each frame. The visual stimuli sensed and processed by the retina are coded and transmitted through the optic nerve to the cortex using spikes (quick electrical impulses) and the spikes represent activity for individual neurons. This way, as soon as there is a feature on the retina that elicits the firing of a set of simultaneous spikes in one layer, it may elicit a set of other spikes in the next layer, and so on from layer to layer, producing a wave of spikes through the layered structure of the cortex. This propagation of simultaneous spikes from layer to layer is equivalent to processing first the most active pixels of an image, and as time passes, the less and less active pixels are processed as well. Note that this way of processing does not wait for a complete frame to be processed in a layer, before starting the computations in the next layer.

The hardware convolution processor we present in this paper is based on the address–event-representation (AER) scheme, which is event (spike)-based, as opposed to frame-based representation. AER has some interesting capabilities for hardware implementations, one of which is the possibility of performing convolutions "on the fly," as we describe in this paper. In Section II, AER is briefly summarized. Section III explains how AER can be applied for generic convolution processing, and Section IV describes the specific system architecture and circuits we have implemented for this. Section V provides experimental results obtained from the fabricated prototype.

Finally, Section VI discusses up-scaling issues for multichip multilayer visual recognition system.

II. Address–Event-Representation Protocol

AER-based interchip communication was originally proposed by Mahowald and Sivilotti [18]–[20] to reproduce the state of a 2-D array of neurons from one emitter chip onto another receiver chip, continuously and in real time. A growing community of researchers is using the scheme for bioinspired vision [21]–[29] and audition [31] systems. Since then, the scheme has been evolving in efficiency and processing power.

Fig. 1 shows the essence behind the AER protocol. The emitter chip contains an array of cells or pixels whose intensity or activity changes in time with slow time constants. This happens, for example, in commercial cameras or artificial retinas where the bandwidth of the signal sensed by an individual pixel is in the order of hundreds of hertz at the most. Each pixel contains an integrate-and-fire neuron whose output frequency is proportional to pixel intensity.[2] The neuron produces spikes of very short duration (in the order of *nanoseconds*) but with much longer spike intervals (in the order of *milliseconds*). These spikes are called events. Every time a pixel sends a spike, its (x, y) coordinate is written on an interchip high-speed digital bus and sent to one or more receiver chips. Events are generated asynchronously. Therefore, additional handshaking signals are required for proper transmission of events from chip to chip. Also, Because events are generated asynchronously, collisions of events generated simultaneously by different pixels may occur. Several ways of handling collisions have been reported in the literature. One way is to detect and discard events that collide [21]–[24], while another is to introduce arbitration [33], [35], [36] and enforce sequencing of colliding events. The latter is more sophisticated but can handle much higher event traffic loads. The prototype presented in this paper uses the arbitrated approach.

In Fig. 1, each event produced by the emitter chip is received by one receiver chip. The receiver chip decodes the address of the event and sends it to the pixel with the same (x, y) coordinate. This pixel contains some type of integration mechanism that reconstructs the original low-frequency time waveform of the same coordinate pixel in the transmitter chip. The delay between events produced in the emitter pixel until they are received by the receiver pixel is in the order of *nanoseconds*. One can say the signals at the receiver pixels are identical and simultaneous to those in the emitter pixels, as if there were wires between pixels of the same coordinate. However, the only physical wires between chips are the ones forming the high-speed digital bus, which has a relatively small number of pins compared to the number of pixels of the images.[3]

Other researchers communicate directly analog values from chip to chip [37]. Besides the inherent limitations of this approach with respect to noise, precision, and fan-out, one could not take advantage of the extra computational processing capabilities offered by AER, as explained next.

[2]In this paper, we consider that pixel activity is coded as spike frequency. This is called "rate coding." However, this is not a restriction of AER. Different coding schemes can be adopted such as rank order coding [32], derivative of activity [33], and synchronicity, [34].

[3]If there are N^2 pixels, only $n_w = \log_2(N^2)$ physical wires are required. If $N^2 = 128 \times 128 = 16\,384$, then $n_w = 14$.

The AER protocol not only allows for a virtual wiring between pixels of emitter and receiver chips, but allows for extra processing on the addresses while they travel between chips. For example, image translation can be performed by inserting digital adders between chips that would add fixed offsets to the travelling (x, y) coordinates. Image rotations could be performed by inserting properly coded lookup tables, as well as any arbitrary transformations and distortions. Even sophisticated microcontroller-based approaches have been reported that generate a sequence of events (obtained through lookup tables) for each original event [38], [39]. In 1999, Serrano *et al.* introduced an architecture for performing AER-based real-time programmable convolutions [29]. However, these convolution operations were limited to kernels $w_k(x, y)$, which are decomposable into x and y components $w_k(x, y) = h(x) v(y)$. The processor presented in this paper does not suffer from this restriction and can be programmed to perform convolutions with arbitrary kernels. Other researchers presented in the past AER circuits for convolution processing. For example, Vernier *et al.* presented a chip with a fixed hardwired diffusive elliptical kernel, although spatial shape could be slightly fine tuned through analog biases [23]. Choi *et al.* [25], [26] have recently presented another way of performing real-time AER-based convolutions, but their technique is restricted to Gabor type of kernels. Another approach based on external microcontrollers was proposed by Goldberg *et al.* [38]. However, this scheme introduces a severe speed reduction in processing throughput.

III. AER for Generic Kernel Convolution Operations

The convolution processor presented in this paper is based on weighted charge package integration operations at the pixels. The idea is schematically illustrated in Fig. 2. Every time the receiver chip (in this case, our convolution chip) receives an event of coordinate (x_o, y_o), the convolution kernel $w_k(x_k, y_k)$ (programmed on an on-chip RAM) is copied around coordinate (x_o, y_o). This way, a pixel of coordinate $(x_o + x_k, y_o + y_k)$ contains the kernel value $w_k(x_k, y_k)$, which is used to modulate current $I(w_k)$ of Fig. 2. This modulated current is used to generate a charge packet for this event which is integrated on this pixel capacitor. The duration of the current pulse is fixed and identical for all pixels. The capacitor is part of an integrate-and-fire neuron: after the integration of each modulated charge packet, capacitor voltage v_c increases; when v_c reaches a fixed threshold (common for all pixels), the capacitor is reset and this pixel generates an event that is transmitted out of the chip to the next processing layer.

Many image processing applications require kernels with positive and negative weights. Consequently, in practice, the kernel values $w_k(x_k, y_k)$ can be positive or negative, and the charge integration circuits need to handle signed integration. As a consequence, pixels generate as well signed events, depending on whether capacitor voltage v_c reaches a positive or negative threshold. Therefore, in general, the (x, y) coordinate transmitted between AER chips should include a sign bit as well. Furthermore, if we want to cascade convolution chips, then each convolution chip should be capable of receiving and processing signed events with signed kernels. Thus, the simplified diagram of Fig. 2 needs to be extended to handle signed incoming events together with signed kernel values,

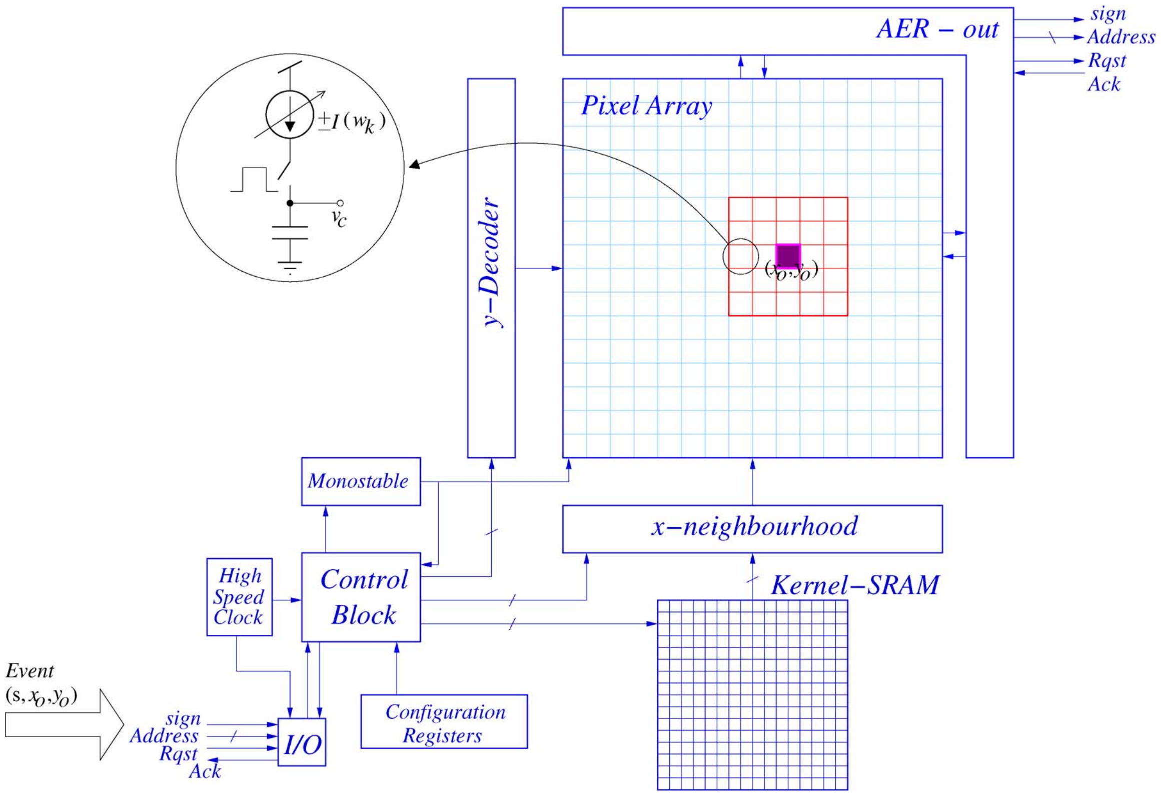


Fig. 2. Architecture of AER convolution chip, indicating the main blocks. The inset illustrates the weighted charge package integration principle used for the convolution computations.

and should be capable of producing signed events. The details of such circuits are described in Section IV. However, before going into circuit descriptions, let us first mention the multichip assembly capability of AER for convolution operations.

Suppose we have been able to fabricate an AER convolution chip with an array of 64 × 64 integrators. The convolution chip contains a RAM for storing kernels of size $K \times K$. In general, K could be larger or smaller than 64 (for example, the convolution chip presented in this work has a RAM size equal to the pixel array size). Let us assume $K = 64$. Suppose we would like to process images of size 256 × 256 with kernels of size up to $K = 64$. Then, we could use our 64 × 64 convolution chips to assemble an array of 4 × 4 of such chips, as shown in Fig. 3. Each chip needs to know its own $x_{\min}$, $x_{\max}$, $y_{\min}$, and $y_{\max}$ coordinates, and each chip sees the complete address space $x = 1, \ldots 256$ and $y = 1, \ldots 256$. Also, each chip stores the complete kernel (of up to 64 × 64) in its own RAM. Suppose now the 4 × 4 chips receive an event of coordinate (x_o, y_o), and kernel size is maximum, as shown in Fig. 3. Then, in general, there are four chips that need to process this event, and each chip has to copy a different part of the kernel to a different part of its own array of integrators. Note that each chip operates autonomously, without interacting with the rest. The only interaction is the sharing of the incoming AER address space and the proper handling of handshaking signals.[4] Alternatively, AER convolution chips can be designed to have multireceiver AER communication capability [40]. Consequently, AER convolution chips can be tiled to process larger images, but the kernel size is limited to the size of the kernel-RAM of each chip.

The fact that a convolution chip can see an input address space larger than the address space of its own pixels also allows to compensate for the boundary problem of convolution processing. As long as input space extends beyond the convolution array space by the size of the kernel, no boundary effects are observed.

IV. Architecture and Circuit Description of the Convolution Chip Processor

The basic architecture of the convolution chip is shown in Fig. 2. It includes an array of pixels, a row decoder (*y-decoder* in Fig. 2), a kernel static-RAM (SRAM), an x-neighborhood selection block, a digital controller, a set of configuration registers, a monostable, an input/output (I/O) block for handling incoming AER events, a high-speed clock, and an AER-out block that generates the outgoing AER events. The controller and I/O block are conventional digital state machines, described in VHDL[5][41], and synthesized using standard digital system design procedures. They are synchronous blocks, clocked by an on-chip high-speed clock. The configuration registers set the input address space and kernel size. The I/O block handles the communication with the incoming AER bus, samples periodically the "Rqst" line and, if active, reads the event address, and returns the "Ack" signal. Interchip communication is asynchronous, using conventional four phase handshaking signals

[4]In practice, we do this by using AER bus splitters at the input of the array, and mergers at the output [63].

[5]This allows to easily add more functionalities in the future.

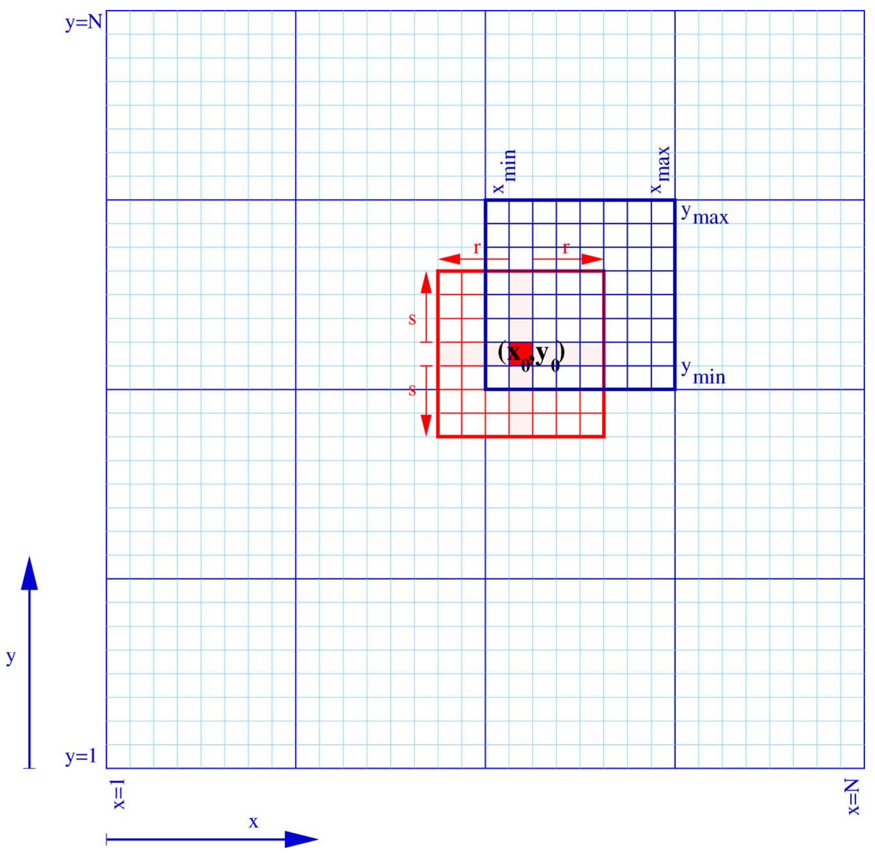


Fig. 3. Multichip AER tiling mechanism.

for request “Rqst” and acknowledge “Ack.” For each event, the controller first computes the limiting rows and columns of the pixel array onto which the kernel needs to be copied. Then, it sends a shift left/right signal word to the x-neighborhood block so that the RAM output columns are redirected left/right as many positions as required by the kernel copy operation. Afterwards, kernel rows of the RAM are copied sequentially to pixel array rows, one after the other. Once the kernel words are loaded onto registers in the appropriate pixels, the monostable triggers a fixed duration pulse for all pixels during which weighted charge packages are integrated at each pixel. Independently, when a pixel reaches its integration threshold, it produces an event that is transmitted to the AER-out block. This block uses digital asynchronous circuit techniques [35]. It arbitrates rows and columns to avoid event collisions, and generates the output event addresses, together with handshaking signals, of the events produced in the pixel array. This summarizes briefly the operations of the architecture of Fig. 2. Next, we describe in more detail some of the main blocks.

A. *High-Speed Clock*

An autonomous high-speed clock based on a five-inverter ring oscillator has been implemented. Two inverters have an analog control voltage that fine tunes their delay, so that the frequency of the oscillator could be tuned between 75–200 MHz. We do not expect this clock circuit will inject harmful noise by itself into the rest of the chip. However, the clock distribution circuit inside the controller will certainly be much more harmful. For this reason, careful layout isolation guard rings are added around this clock and the controller.

B. *I/O Block for Incoming AER Events*

This is a synchronous circuit described in VHDL and synthesized using standard cells. The circuit samples periodically the “Rqst” line of the AER-in bus. If it is active, it reads the incoming event address and stores it in a shift register. Then, it activates the “Ack” signal and finishes the handshaking. Incoming addresses are queued in a shift register. This way, the circuit can fetch in addresses faster than the controller would process them, until the queue is full. This allows to free the interchip bus before a bundle of events are processed, which comes in handy during short bursts of events. A shift register of n positions is used. It includes a pointer to one of the shift register positions. Addresses are loaded in starting from the last position $n-1$. If there are j addresses in the queue, the pointer value is $n-j$, which is the location of the latest address. Every time the controller reads an address from the last position $n-1$, all addresses are transferred one position ahead and the pointer increment by one.[6] In our convolution chip prototype, we have implemented a queue of $n=4$ positions.

C. *Control Block*

The function of the controller block, once it has read an event address from the AER-in I/O block, is to do the following: 1)

[6]In the actual implementation, a circular register is implemented. This way, instead of transferring all addresses one position, an extra reference pointer is shifted. This speeds up significantly the whole process.

compute the kernel left/right shift, 2) copy sequentially kernel rows from the RAM to the pixel array, 3) trigger the monostable, and 4) erase the copied kernel values in the pixel array. In practice, this sequence is performed in a pipe-lined manner, so that the maximum delay for processing an event is equal to the slowest step instead of the sum of all. Usually, the slowest step is the sequential copy of kernel rows from RAM to pixel array, except for very small kernel sizes.

Whether or not the convolution chip is part of a tile, as was shown in Fig. 3, or is alone, it knows its limiting coordinates $x_{\min}$, $x_{\max}$, $y_{\min}$, and $y_{\max}$. For this, the chip contains a set of configuration registers, which are loaded at startup together with the kernel. Another two configuration registers define the horizontal and vertical size of the kernel stored in the RAM. If the kernel is of size $(2s+1) \times (2r+1)$, these two registers store the values s and r. Note that we are here imposing an odd number of columns and rows for the kernel. This way, there exists a physical kernel center position which coincides with the incoming event address position[7] (x_o, y_o). The following four different situations need to be distinguished, in general, by the controller before starting to copy kernel rows from the RAM to the pixel array (see Fig. 3 as a reference).

1) No kernel rows lie within the present chip. This happens when either $y_o + s < y_{\min}$ or $y_o - s > y_{\max}$.
2) The top kernel rows need to be copied to the bottom part of the pixel array. This happens when $y_o - s < y_{\min}$ and $y_o + s > y_{\min}$. In this case, $q = y_o - y_{\min} + s - 1$ rows need to be copied, and the controller generates the following two sequences for selecting pixel array rows (y_{array}) and RAM rows (y_{ram}):

$$y_{\text{array}}(j) = j \quad y_{\text{ram}}(j) = (2s+1) - q + j, \qquad j = 1, \ldots q. \quad (1)$$

3) The bottom kernel rows need to be copied to the top part of the pixel array. This happens when $y_o - s < y_{\max}$ and $y_o + s > y_{\max}$. In this case, $q = y_{\max} - y_o + s - 1$ rows need to be copied, and the controller generates the following two sequences for selecting pixel array and RAM rows:

$$y_{\text{array}}(j) = N - q + j \quad y_{\text{ram}}(j) = j, \qquad j = 1, \ldots q \quad (2)$$

where $N = y_{\max} - y_{\min} + 1$ is the pixel array size.
4) The kernel rows are all copied to the pixel array. This happens when $y_o - s > y_{\min}$ and $y_o + s < y_{\max}$. In this case, all $2s + 1$ rows are copied, and the controller generates the following two sequences for selecting pixel array rows and RAM rows:

$$y_{\text{array}}(j) = y_o - y_{\min} - s + j \quad y_{\text{ram}}(j) = j, \qquad j = 1, \ldots 2s+1. \quad (3)$$

Besides this, the controller also introduces a left/right shift in the connections between the RAM output columns and the columns of the pixel array. This left/right shift is activated for each incoming event and is held fixed during the complete sequence of row copies from RAM to pixel array. The left/right shift computed by the controller is $\Delta x = x_o - x_{\min} - r$, where it is assumed that if the kernel is smaller than the pixel array ($2r + 1 < x_{\max} - x_{\min} + 1$), the RAM columns are filled initially with kernel values from left to right, leaving the right-most columns in the RAM empty. The circuit responsible for producing this left/right shift is the x-neighborhood selection block.

[7]Although we are imposing kernels of odd size, so that the central coordinate of the kernel coincides with the event address, in practice, there is no restriction on kernel symmetry: the user can always set to zero those kernel values necessary to shape the symmetry as desired.

D. X-Neighborhood Selection Block

The x-neighborhood selection circuit that has been implemented is shown in Fig. 4. Each RAM data column is routed to the left and to the right 45° over a connection matrix. Each dot in the connection matrix contains two tristate buffers. They receive two horizontal input signals, one for gating a left shift and the other for gating a right shift. Only one row of the connection matrix and for this row only one of the two horizontal input signals can be active. The other signals are two 45° lines, which are the RAM outputs shifted to the left or right Δx positions. The bottom most row in the connection matrix corresponds to $\Delta x = 0$, the next one to $\Delta x = \pm 1$, the following one to $\Delta x = \pm 2$, and so on. The outputs of the two AND gates are wired-OR to the corresponding input column of the pixel array. Two decoders on the left of the connection matrix activate only one of the possible horizontal left/right gating signals for the whole connection matrix. This guarantees that only one row of the connection matrix is active, and for this row only the left or the right shift AND gates are active.

E. Convolution Pixel

The convolution pixel performs a signed and weighted charge package integration for each incoming event. The basic operation principle was illustrated in the inset of Fig. 2, where a programmable current source is switched during a short-time pulse onto an integrating capacitor. The value of the current pulse can be quite small, in the order of *nanoamperes* or less. This value depends on the frequency range of incoming events, on the frequency range of the outgoing events, on the monostable pulse width (which is in the order of *nanoseconds*), and on kernel size and shape. Note that for larger kernels, the contribution of each pulse has to be reduced if we want to keep the same output event frequency range. Because the integrating capacitance has to be kept small ($\backsim$ 100 fF) because of area requirements, it turns out that the switched currents can become quite small (*nanoamperes* or even *picoamperes*) for some circumstances. Under these conditions, a current source followed by a switch (as shown in Fig. 2) would produce a charge packet dominated by clock feedthrough charge injection [42], and the resulting charge packet would be practically independent of the weighted current. To avoid this, current source transistors are pulsed from their sources, as shown in Fig. 5(a) [38]. The basic current mirror is formed by transistors M_1 and M_2, where current I_w is mirrored from M_1 to M_2. The source of M_2 is pulsed between V_{dd} and voltage V_{off} depending on digital signal *pulse* and digital word $\boldsymbol{w} = (w_0, w_1, \ldots w_{n-1})$. M_3 and M_4 act as switches. M_5 has been added for symmetry. Voltage V_{off} is such

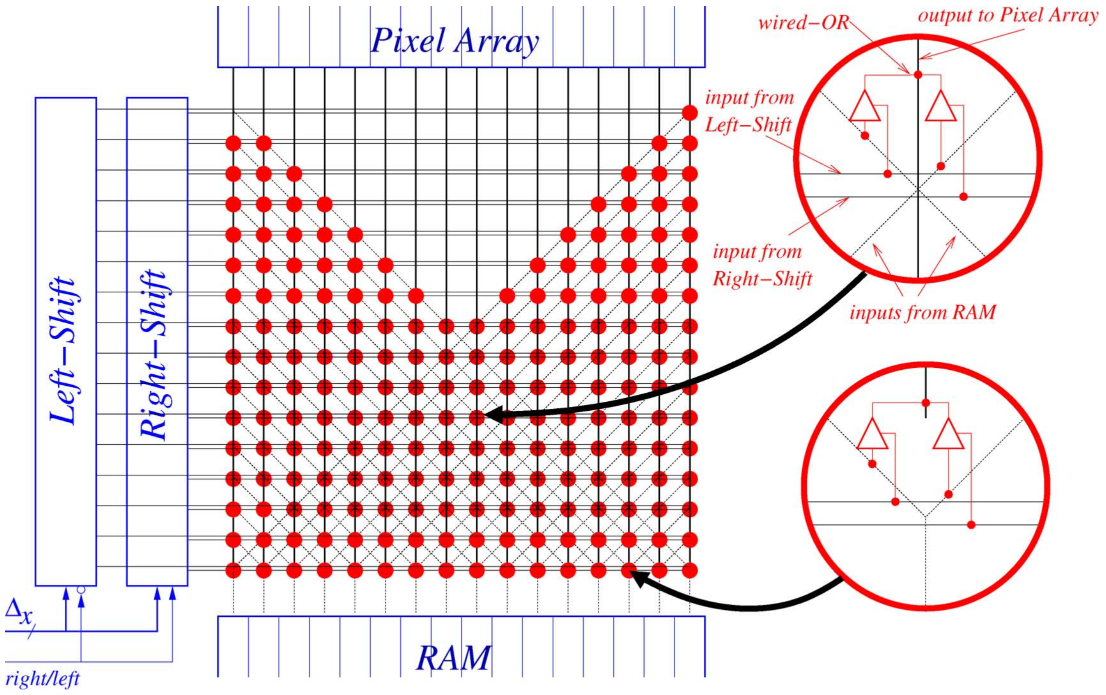


Fig. 4. X-neighborhood selection circuit diagram for kernel left/right shift.

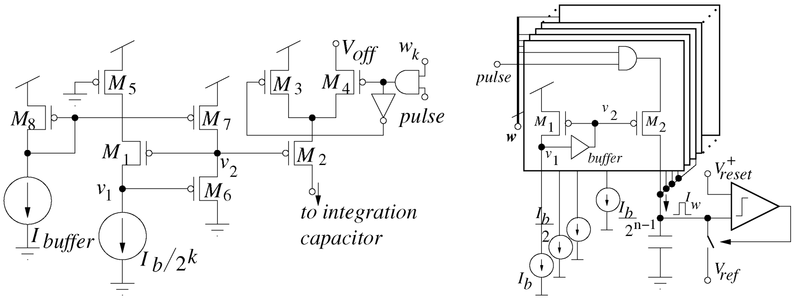


Fig. 5. (a) Detailed circuit of pulsing current sources for generating precise charge current packets. (b) Equivalent scheme for multibit digitally controlled charge-packet generation.

that M_2 does not drive any current when *pulse* is low. Transistors M_6–M_8 form a source follower buffer from node v_1 to v_2. This way, node v_2 presents lower impedance and is more insensitive to the switching at the source of M_2, specially for very small I_b currents. This circuit is capable of providing pulsed currents of $I_w = \Sigma_{k=0}^{n-1} w_k I_b/2^k$ down to 3 pA, while setting the bias current of the source follower to $I_{\text{buffer}} = 10$ nA [43]. In case currents smaller than picoamperes are required, we biased the sources of M_3 and M_5 slightly below the power supply [44]. In Fig. 2(b), current sources $I_b/2^k$ are scaled versions of a global bias current I_b set at the periphery of the chip.

The digital weighting of current I_w is implemented by putting several circuits of Fig. 5(a) in parallel sharing the integration capacitor. The number of parallel sections is given by the number of bits of the programming weight. For each section, the pulse current $I_b/2^k$ ($k = 0, \ldots n-1$) is proportional to a power of two, depending on the weight bit number. This is shown in Fig. 5(b). The digital weighting word w is available in a pixel dynamic register. This register value is updated for each event, and is the corresponding kernel value copied from the kernel RAM, as was explained in Section III. The circuit of Fig. 5(b) corresponds to a single sign version of a weight charge package integration circuit. For the double sign version, everything, except the capacitor, has to be symmetrically replicated (NMOS to PMOS and PMOS to NMOS). There are two voltage comparators, one connected to a positive V^+_{reset} value and the other to a negative V^-_{reset} value. The reference voltage V_{ref} is the same for both comparators and should be $V_{\text{ref}} = \left(V^+_{\text{reset}} + V^-_{\text{reset}}\right)/2$. Both symmetrical half sections have a different "pulse" signal. Only one of the two is activated depending on the combination of signs of the incoming event and the sign of the register weight w. Voltages V^+_{reset} and V^-_{reset} are global voltages common for all pixels. This way the output event frequency produced by a pixel is proportional to the equivalent average input current I_{avg} integrated onto its pixel capacitor

$$f_{\text{pixel}}\big|_{\text{average}} = \frac{|I_{\text{avg}}|}{C\left(V^{+/-}_{\text{reset}} - V_{\text{ref}}\right)}. \tag{4}$$

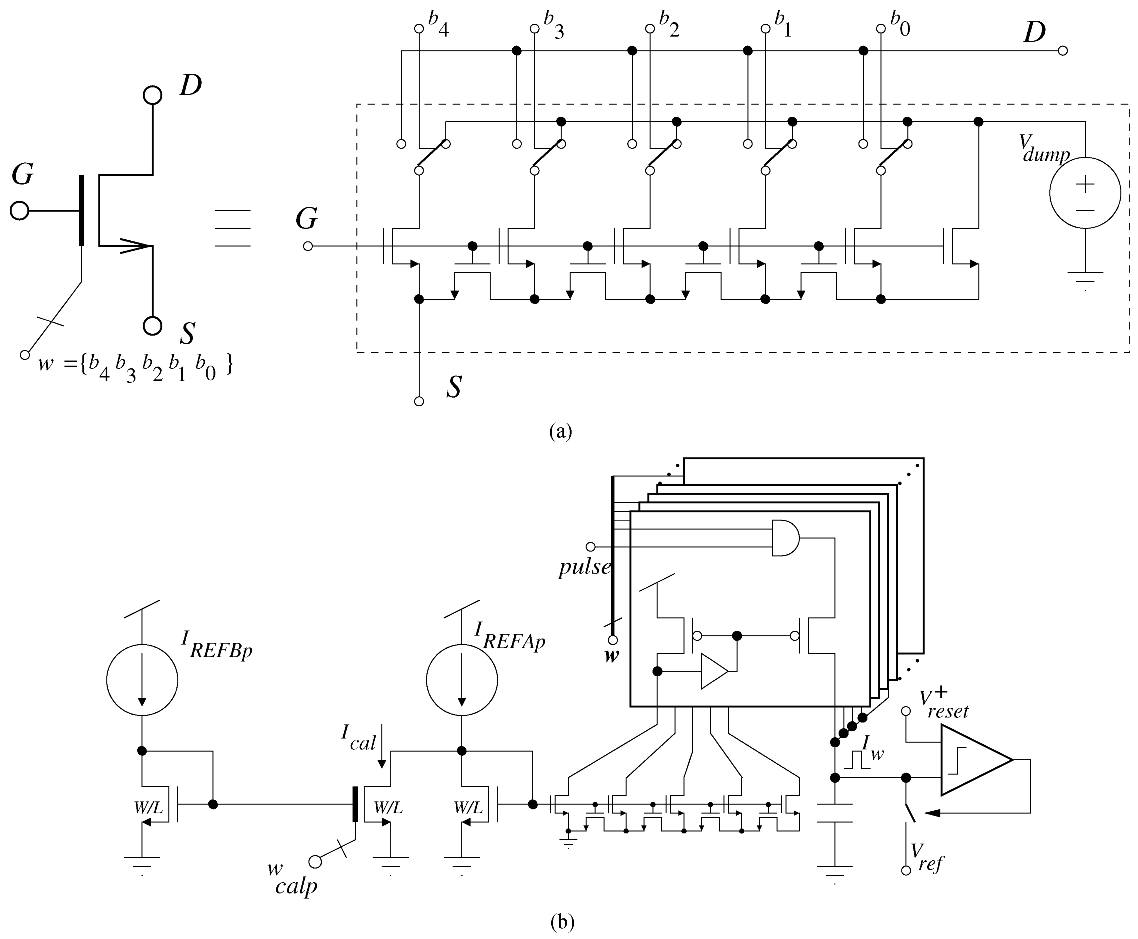


Fig. 6. (a) Circuit schematic of the digi-MOS concept. (b) Single sign calibrated circuit for weighted charge package integration and output events generation.

The pixels output event frequency range can be globally adjusted by either tuning $V_{\text{reset}}^{+/-}$ or global bias current I_b. In practice, $V_{\text{reset}}^{+/-}$ is kept constant and large to maximize voltage excursions. This way impact of comparators offset voltage mismatch is minimized. Thus, we use I_b to adjust output events frequency operating ranges.

The binary weighted currents in Fig. 5(b) $I_b, I_b/2, \ldots I_b/2^{n-1}$ may need to be adjusted to quite small values, down to *nanoamperes* or even *picoamperes*. At such current levels, transistor mismatch is quite important. The 0.35-μm complimentary metal–oxide–semiconductor (CMOS) technology measurements of NMOS transistor current mismatch standard deviations $\sigma(\Delta I/I)$ for a wide range of transistor sizes and for several decades of operating currents have been reported elsewhere [45]. From such measurements, we can see that current mismatch for 2.5 μm $\times$ 1.5 μm transistors at currents smaller than 10 nA has a standard deviation of $\sigma = 8.5\%$. Assuming Gaussian distributions, this means that the current sources in an array are spread over a 6σ ($\pm 3\sigma$) interval of $6 \times 8.5 = 51\%$. Consequently, we could not even guarantee a precision of 1 bit. Furthermore, note that these considerations do not take into account mismatch caused by die gradients, which can be quite relevant for current sources spread over large surfaces. Because we want to perform weighted charge packet integration with several bits of resolution, but have to do it with very low currents, we require the use of some calibration method. We have used the mini digital-to-analog converter (DAC)-based method, proposed recently by Serrano *et al.* [46]. In this method, a compact metal–oxide–semiconductor (MOS) ladder structure with selection switches, as shown in Fig. 6(a), called the "digi-MOS," is used as a digitally controlled MOS transistor. This structure can be exploited, as shown in Fig. 6(b) [46], to calibrate the set of binary weighted pulse currents of Fig. 5(b). By properly adjusting I_{cal} for each pixel, current I_b in Fig. 5(b) can result in acceptable interpixel mismatch. The relationship between required precision after calibration, transistor sizes, and operating currents for this structure is explained in [46]. An alternative scheme has been reported recently [65].

F. Kernel Static RAM With High-Speed Readout

The kernel RAM is loaded initially at startup, and needs to hold the kernel weights for the rest of the time. Consequently, we implemented a static RAM. The speed of this SRAM writing process is not critical. However, during the incoming event processing operation, the slowest step of the convolution processing is the copy of the kernel lines to the pixel array. Consequently, it is critical to have a high-speed readout SRAM. Conventional high-speed SRAMs are based on the use of efficient peripheral sense amplifiers [47]. However, such techniques are justified

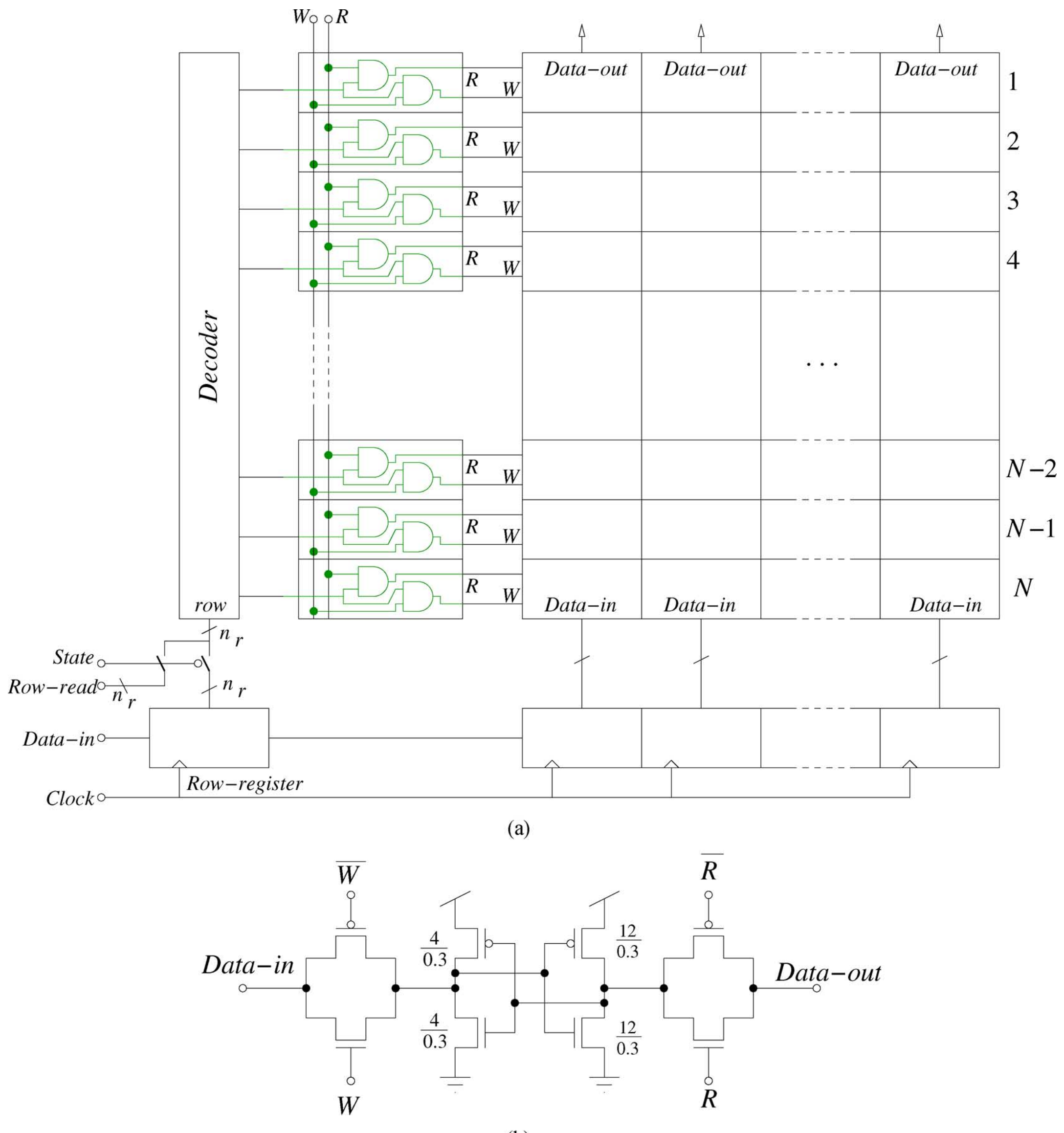


Fig. 7. High-speed readout SRAM circuit. (a) Global floorplan structure. (b) SRAM cell detail.

for large-size SRAMs. In our particular case, we are using a quite small 16×16 RAM matrix. Therefore, we will achieve high-speed readout by implementing a more *ad hoc* design.

Fig. 7(a) shows a floor plan of the SRAM. Kernel data is loaded initially through the bottom shift register (*data-in*) row after row. The first n_r bits indicate the SRAM row into which the kernel data is loaded. The rest of the bits are $N \times (n+1)$, where N is the number of array pixel columns[8] and n is the number of the bits per weight. The extra bit is the sign bit of the weight. Signal "*state*" in Fig. 7(a) indicates whether we are writing the SRAM through the bottom shift register, or the controller is reading out the SRAM. The circuitry of each SRAM bit cell is shown in Fig. 7(b). It contains a latch with a strong and a weak inverter. The weak inverter side is for writing in the data, while the strong inverter side is for reading it out. Transistor sizes of data-out side have been optimized for maximum readout speed. Depending on the size of the SRAM (and pixel array), it might be convenient or not to implement a tree of readout switches and branches, to minimize parasitic output path capacitances.

[8]Remember we are making the size of the RAM equal to the size of the pixel array.

G. Monostable Circuit

The convolution chip processor includes a single monostable of programmable output pulsewidth, which delivers a global output pulse to the pixel array for the charge package integration process. A schematic diagram of the implemented monostable is shown in Fig. 8. Depending on the sign of the input event being processed, the monostable provides the integration pulse through either the "pulse-pos" or "pulse-neg" lines. Initially, the trigger signal P is low and no output pulse is produced ($\overline{\text{Pulse}} = 1$). This makes $P_i = 0$, closing switches S_1 and S_3, and opening S_2 and S_4. This makes capacitor C_T terminal voltage $v_c = 0$. If trigger input P becomes momentarily high, S_1 and S_3 open while S_2 and S_4 close. Signal $\overline{\text{Pulse}}$ becomes

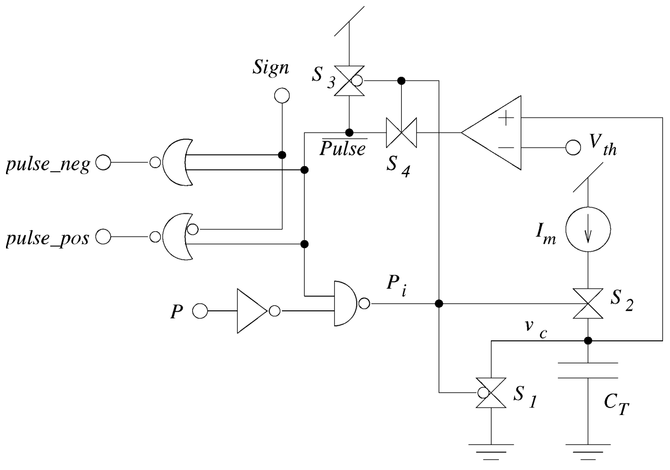
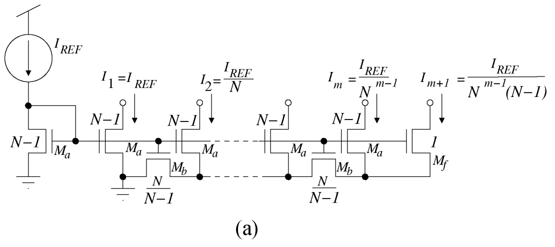


Fig. 8. Conceptual block diagram of the programmable width monostable circuit.

low and one of the output pulse signals is activated. Capacitor C_T starts to be charged by programmable current I_m. The charging continues until capacitor voltage v_c reaches threshold $V_{\rm th}$, and $\overline{\rm Pulse}$ becomes the high finishing integration pulse. The total length of the integration pulse is $T_{\rm pulse} = C_T V_{\rm th}/I_m$.

H. I-Pots for Programming Global BIAS Currents

The convolution chip processor has a total of 31 global bias currents. Because the chip reported in this paper is a first preliminary experimental prototype, it is desirable to allow a wide programming range for all these bias currents while, at the same time, there is a fine enough tuning capability in case some biases turn out to be critical. The obvious option would be to provide one external bias pin for each current, but this complicates excessively chip packaging. To solve this dilemma, we have developed the *I-Pot* cell, which only needs three external pins independently of the number of bias current sources needed. This cell is based on the use of the current splitting ladder circuit of Fig. 9(a) [48], [51]. This circuit provides m current sources where each is the Nth fraction of the previous one. Factor N is set by transistor size ratios. If $N = 10$, we have a current decades generator, or generic range selector (for $N > 10$). If $N = 2$, we have a conventional current DAC. Our *I-Pot* cell exploits both modes of current splitting, as shown in Fig. 9(b). A global reference current $I_{\rm REF}$ is replicated for each *I-Pot* cell. This current is fed into a current range selector. Only one of its output branches is selected by the digital word in shift register A. This current is then fed into a linear DAC controlled by the digital word in shift register B. The extra bit in register C decides whether the generated bias current is driven to its destination bias point or to an external pin (*I-Test*) for characterization. Optionally, register B may contain a sign bit that controls whether the output current is fed or not to an extra current mirror for current sign inversion. Each *I-Pot* needs to be initially characterized by an external instrument. The resolution of the DAC is such that the number of bits is larger than the precision limit imposed by transistor mismatch. Consequently, the lower bits just produce mismatch-induced random values. However,



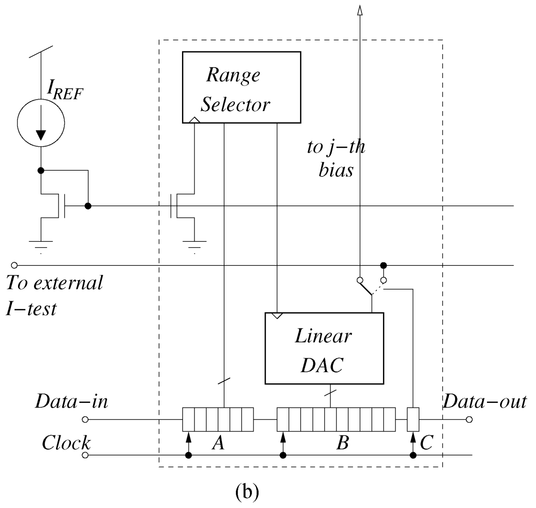


Fig. 9. (a) Ladder structure circuit for providing currents ratioed by factor N. (b) Conceptual circuit diagram of *I-Pot* cell.

our goal is just to generate sufficiently dense values randomly scattered. After characterization of the *I-Pot*, the digital register words are ordered so that a monotonically increasing sequence of bias currents results. As more bits are implemented in the DAC, the denser this sequence is. An *I-Pot* specific lookup table is stored on a computer for each *I-Pot*. During normal operation, this computer loads the shift registers of all *I-Pots*. After experimentation with all bias currents for proper system operation, the final *I-Pots* bias words to be loaded into the shift registers can be stored into a programmable read-only memory (PROM) that downloads them at startup. Note that these words are chip-specific and need to be characterized for all *I-Pots* of each chip.

A similar but simpler *I-Pot* circuit has been reported recently, which uses only the linear DAC in Fig. 9 but with a very large number (24) of bits and ladder branches [49]. This structure achieves much less precision for low currents, uses more bit memory, and has larger area. An improved version of the *I-Pot* in Fig. 9 has been reported recently [50].

I. AER-Out

The peripheral AER-out block used is the burst-mode module proposed and reported in full details by Boahen in 1999 [36]. We simply adapted it for handling signed events [52]. This is a fully asynchronous self-timed block, synthesized by Boahen using the hardware description language and synthesis methodology of Martin for asynchronous circuits [53]. The basic operation mechanism can be briefly explained with the help of Fig. 10. There is an array of pixels that generates the events. Each pixel event output is wired-OR over its row to the *row arbiter* on the

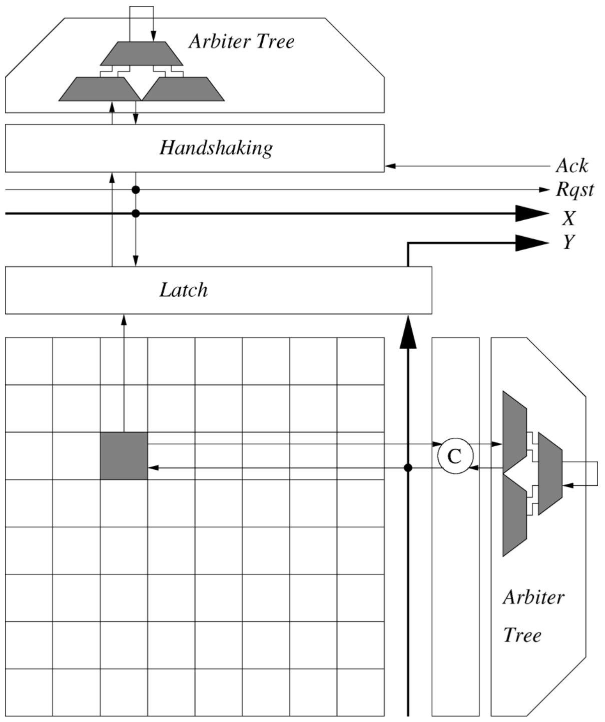


Fig. 10. Basic operation diagram of Boahen's burst-mode AER-out module.

right. Once the row arbiter selects and acknowledges one specific row, this row writes its active events on column lines.[9] Then, the row latch copies the content of the column lines. Once the asynchronous handshaking between the row arbiter and the active row finishes, a new row can start to request selection from the row arbiter. At the same time, the events latched in the top row latch are arbitrated and directed sequentially to the chip AER output port, together with the proper handshaking signals.

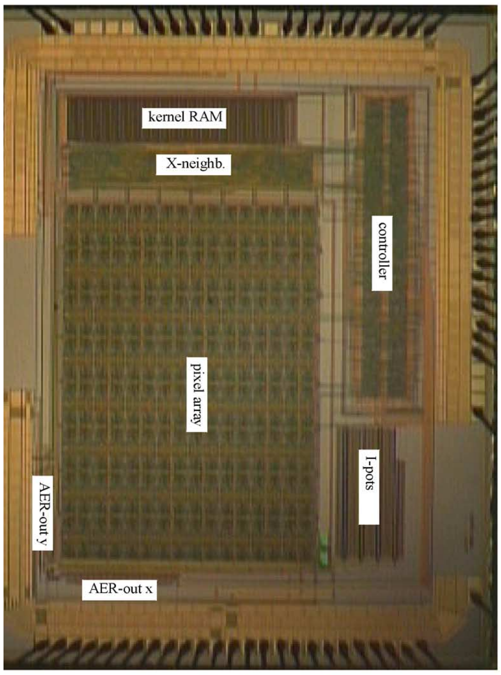
In the original unsigned circuit by Boahen, the number of column lines between pixel array and top row latch were one per pixel column. In the signed version, we are using two per pixel column, one for each sign. The pixel communication with the row arbiter and the row arbiter itself remain the same. Regarding the top arbitration and communication circuitry, everything is identical to the original Boahen burst-mode version, except that the sign information (an extra sign bit) is added to the output event address.

V. Experimental Results

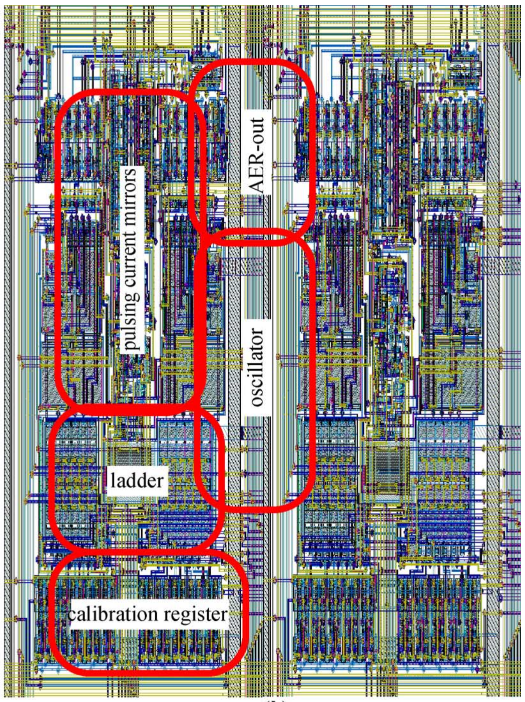
A small-size convolution chip prototype has been fabricated in a 0.35-μm CMOS technology. Because we are already using techniques for compensating leakage (*fA* circuits [44]) and calibrating for mismatch, we do not expect severe problems when scaling this architecture below 0.35 μm. The area of the complete prototype is 3.3×4.2 mm^2. The purpose of this first prototype is to test the proper operation of all subsystems involved, verify correct communication among them, and provide a first chip for testing the viability of the AER technology for real-time image convolutions. The fabricated convolution chip prototype contained a pixel array of size 16×16 and a maximum kernel RAM size of 16×16. Each pixel occupied an area of 100×140 μm^2. As a first prototype, we decided to oversize the resolution of the programmable kernel to 6 bits (sign bit plus five weight bits). For future prototypes, we expect that a resolution of 4 bits (sign bit plus three weight bits) could serve most purposes well [54]. A microphotograph of the fabricated prototype is shown in Fig. 11(a). Labels indicate the location of different blocks: the 16×16 pixel array, the $16 \times 16 \times 6$ bits programmable kernel high-speed readout static RAM, the left/right shift x-neighborhood selection block, the digital controller, the row and column arbiters with their AER outputs, and the programmable *I-Pots* for analog current biases. The area breakdown of different elements is as follows: pixel array 2400×1740 μm^2, kernel RAM 330×1540 μm^2, x-neighborhood 300×1700 μm^2, controller 2200×560 μm^2, *I-Pots* 900×470 μm^2, AER-out-x 100×750 μm^2, and AER-out-y 800×100 μm^2. The layout of a single pixel is shown in Fig. 11(b). Pixel pitch is 106×147 μm^2. It has a left/right symmetry because all circuitry on the left-hand side is replicated symmetrically on the right, changing NMOS to PMOS transistors and vice versa. This is because of the double sign operation. In the central part of the pixel layout, we can see the circuitry labeled "oscillator" of size 22×53 μm^2, which includes the integrating capacitor and the two voltage comparators, and the asynchronous digital AER-out circuitry of size 23×41 μm^2. The left/right symmetric regions include two calibration registers, each of size 36×21 μm^2; two ladder structures, each of size 26×32 μm^2; and two pulsing current mirrors with its respective selection logic, each of size 28×68 μm^2. Note that the area used for calibration (ladders and calibration registers) is almost half of the total pixel area.

AER events can be fed in up to a peak rate of 5×10^7 events per second (eps) by the I/O circuit and queue. However, this event input rate can be kept only until the queue is full. Maximum sustained input event rate depends on the number of kernel lines n_k in the RAM. The more lines in the kernels, the more clock cycles are needed per input event. Consequently, for large kernels, input event rate is limited by event processing time. The measured event processing delay is $(40 + 20 \times n_k)$ ns/event. In any case, since communication is asynchronous with handshaking, the speed is always limited by the slowest component. Therefore, for example, if there are five kernel lines, the delay per event is 140 ns. However, if the convolution chip is receiving inputs from a retina with maximum event rate of 200 ns/event, then this is the effective rate of the link. On the other hand, if the retina sends events at a rate higher than the convolution chip can handle, then the convolution chip will limit the rate of the link. In that case, we would have to slow down the output rate of the retina. Similarly, if a third chip is receiving the output events of our convolution chip and it cannot handle its high output rate, then we need to scale down our chip output event rate by reducing global bias current I_b (see Fig. 5). In general, in a practical multichip AER system, the event rate range (max/min) of each link is preset *a priori*

[9]Although one pixel requests access to the row arbiter, once the row arbiter has finished arbitration and returns the acknowledged signal, more pixels on this row may become active. All these active pixel states are transferred simultaneously to the top for further processing, and the pixels reset for further integration.



(a)



(b)

Fig. 11. (a) Microphotograph of fabricated convolution chip prototype. (b) Detail of pixel layout.

according to the slowest element in the link and does not need to be readjusted dynamically.

Experimental measurements and characterizations are provided next at the pixel level and the system level.

A. *Pixel Level Experimental Characterizations*

From the pixel operation point of view, the most critical aspect is the verification of the correct operation of the programmable charge packet integration mechanism. In Section IV-E, we introduced the circuits responsible for performing the weighted (and signed) charge packet integration, on which the whole convolution processing operation relies. All pixels produce equal width current pulses, whose current amplitude is modulated by an event coordinate-dependent convolution kernel value. The maximum current pulse is set by a global bias (I_b in Fig. 5). This maximum current pulse amplitude needs to be adjusted depending on maximum input event rate and desired output event frequency ranges as well as kernel size and shape. Note that [see (4)] pixel output event frequency increases with average pixel capacitor current I_{avg} and this current scales linearly with input event rate. Also, if kernels are large, pixels will be activated more frequently and, consequently, the capacitor average input currents will increase as well. Therefore, if we want to maintain a specific output event rate, we need to adjust I_b depending on all these factors. In practice, this may result in current pulses ranging from several microamperes down to fractions of picoamperes [43]. As an illustration, Fig. 12(a) shows the charging and discharging voltage transients at the capacitor of the integrating pixel. All traces are obtained by applying incoming events to the same pixel at a constant time rate of 10 μs, with pulse width equal to 3 μs. Positive slopes correspond to charge packages with positive sign, while negative slopes to those with negative sign. In this particular case, the current pulses were programmed between $I_w = 1$ nA and $I_w/32 \approx 31$ pA, with either positive or negative sign. This is done by adjusting a global bias current of value $I_w = 1$ nA using one of the *I-Pots*, and then storing locally in the pixel a digital value between $[-32, 32]$ using a 6-bit register. Fig. 12(b) shows the resulting slopes of the traces in Fig. 12(a) as a function of the 6-bit pixel weight word. Fig. 12(c) and (d) shows the same thing but when keeping a constant (positive) weight and sweeping global bias current I_b to change the height of the current pulses I_w. Here, we can see there is no mismatch impact (because the pixel current DAC is maintained with a fixed weight).

To illustrate the output spikes produced by a pixel under different situations, Fig. 13 shows measurements obtained from a pixel excited with different stimuli. Input spikes have constant amplitude of 100 nA, pulse width of 100 ns, and period 10 μs. Only the duty cycle and frequency of the sign bit signal is changed. The top subfigures show pixel capacitor voltage and the bottom subfigures pixel output signed spikes (the sign is indicated with labels "+" or "−" at each spike). In the left-hand side column, the pixel is receiving more negative than positive input events, thus producing a net negative output. In the central column, it is receiving more positive than negative input events, thus producing only positive output spikes. In the right-hand side column, a long period of only negative input spikes is followed by a long period of only positive output spikes. Note that, in general, the pixel output frequency is much smaller than the frequency of the incoming events. This ratio is adjustable through bias current I_b. In practice, what happens is that a single

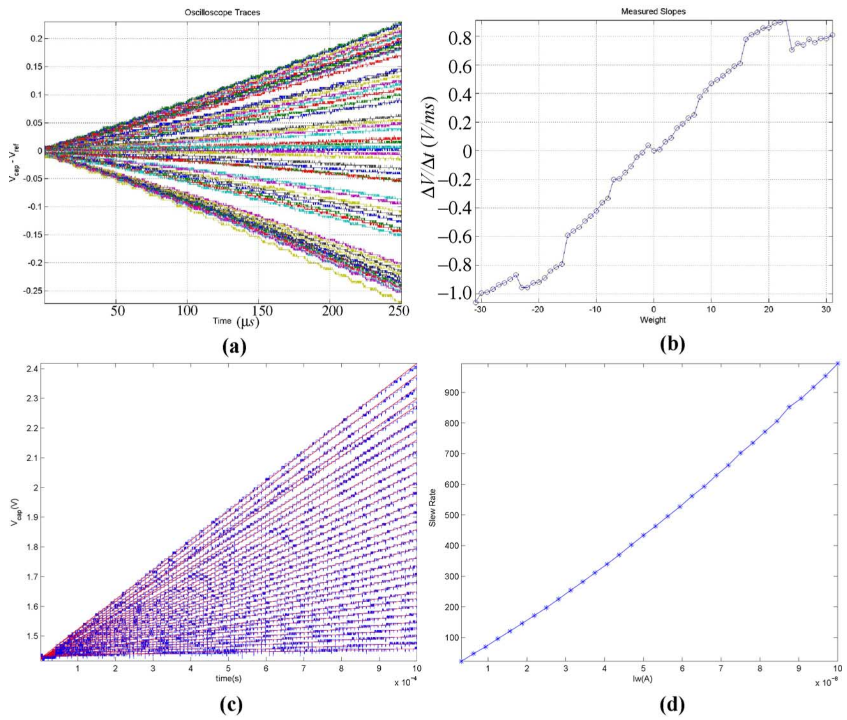


Fig. 12. (a) Capacitor voltage time evolution of the charge packet driven integrator of Figs. 5 and 6. Each trace corresponds to a different weight value programmed onto the pulsing current source I_w. There are a total of 65 traces shown, which correspond to pixel weights from -32 to $+32$. The capacitor is receiving 3 μs charge pulses every 10 μs. The pulsing current is programmed to be either positive (increasing slopes) or negative (decreasing slopes) from values ranging from $I_w = 1$ nA to $I_w =$ 1/32 nA, using a 6-bit signed word. (b) Resulting slopes for the traces in (a). (c) Weight is maintained constant (positive) and global bias current I_b is swept to change I_w between 0 and 100 nA. The capacitor is receiving 100-ns pulses every 10 μs. Staircases are measured capacitor voltages, while continuous lines are computed interpolations. (d) Resulting capacitor slew rate versus I_w.

pixel receives spikes from many pixel coordinates of the preceding layer, depending on kernel size. Depending on kernel size and shape, bias current I_b can be adjusted so that pixel frequencies in subsequent layers are similar.

The main problem when building large arrays of such pixels is the large mismatch that results between the I_w current sources of the pixels, specially when such currents are generated by transistors well biased into weak inversion. For instance, if one repeats the measurements of Fig. 12 for all pixels, a maximum mismatch of $\sigma = 14\%$ is obtained for the most negative weight. This means that the $\pm 3\sigma$ interval is over 80%, well below 1-bit resolution. One option to overcome such large mismatch could be to use a tree of current mirrors of large size to distribute a reference current to all pixels [55], but this consumes a tremendous amount of area. In our implementation, we opted to use a calibration technique proposed recently based on programmable current splitters [46]. This technique uses the calibration circuit shown in Fig. 6, but replicated for double sign operation. The calibration process is as follows. First, we set the pixels weight word to $w = 32$ for all 16×16 pixels and measure the resulting pixel frequencies as function of calibration word w_{calp}. Fig. 14(a) shows the resulting frequencies for all pixels as a function of calibration word w_{calp} (see Fig. 6). The value of global bias current I_{REFBp} is adjusted with respect to global bias current I_{REFAp} so that the minimum frequency at the left ($w_{\text{calp}} = 0$) in Fig. 14(a) is equal (or slightly larger) than the maximum frequency at the right ($w_{\text{calp}} = 31$). In this particular case, we adjusted $I_{\text{REFAp}} = 160$ nA and $I_{\text{REFBp}} = 71$ nA. This way a common frequency can be found for all pixels, as indicated by the thick horizontal line. The optimum calibration word for each pixel is then the one that would make its frequency as close as possible to this common frequency. The resulting optimum calibration words for each pixel are then stored and loaded onto the chip at startup. The calibration process for the opposite sign is performed in a similar manner. Fig. 14(b) shows the resulting 16×16 pixel frequencies when setting kernel weight to $w = -32$ and sweeping calibration words w_{caln}. Again global bias current I_{REFBn} is adjusted relative to I_{REFAn} to maximize the calibration range of the calibration word, while finding a common frequency for all pixels. The optimum calibration words for each pixel are stored off chip and loaded at startup. After calibrating the negative and positive sides, the reference currents are fine tuned to make the output frequencies of the two horizontal lines in Fig. 14 identical.

Note that this calibration technique [46] only calibrates at maximum weight. This way a reasonable compromise is obtained between pixel complexity and calibration capability. Fig. 15 shows the spread (maximum minus minimum value) of all 16×16 pixels as function of kernel weight w before calibration (trace with crosses) and after calibration (trace with circles). After calibration, the worst case error is about 12% (equivalent to a $\sigma = 2\%$), which corresponds to a precision of 3 bits.

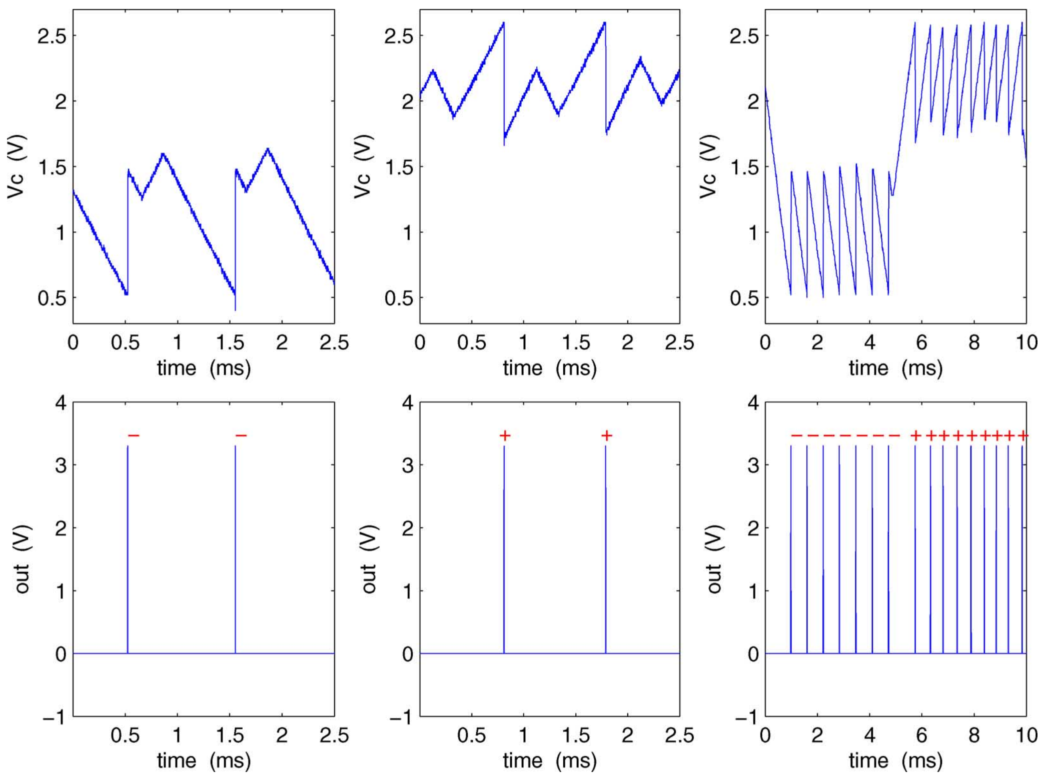
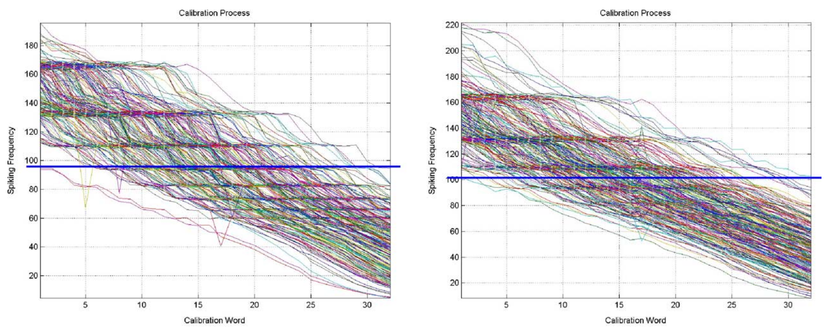


Fig. 13. Pixel behavior under different situations. Top row shows pixel capacitor voltage $V_c(t)$. Bottom row shows pixel output signed spikes (spikes are labelled "$+$" for positive sign and "$-$" for negative sign). Left-hand side column corresponds to a situation of pixel receiving more positive input spikes than negative ones, thus producing only positive output spikes. Central column illustrates the opposite situation. Right-hand side column shows the situation of a pixel receiving negative spikes during a long period and producing negative output spikes, followed by a period where it receives only positive spikes and produces positive output spikes.



(a) (b)

Fig. 14. Illustration of calibration process for (a) negative current pulses and (b) positive current pulses. The figures show the resulting pixel frequencies for all 16×16 pixels, as function of calibration word w_{cal}, for (a) $w = 32$ and (b) $w = -32$. Each trace corresponds to one of the 256 pixels. The meaning of pixel weight w and pixel calibration word can be seen from Figs. 5 and 6.

B. Computer Interfaces

In order to test the performance of the AER convolution chip at the system level we developed some custom made computer interfaces that perform the following functions. The first function, which is rather simple, is to capture (or grab) the address events generated by the convolution chip and dump them into the computer's memory. Each captured event is assigned a timestamp and stored in the computer. The convolution chip output AER bus is monitored during a time period, after which the sequence of captured events is stored on a file in the computer. This file is then analyzed offline.

The opposite function, providing an AER input stimulus to the computer, is more sophisticated. For this purpose, we have developed a special purpose PCI card that uses a Spartan II 200 field-programmable gate array (FPGA). The hardware programmed into the FPGA is capable of transforming the frame-based video streams (available in the computer) onto the spike-based address event representations in real time [56]. This task can be achieved in real time by exploiting linear feedback shift register (LFSR) pseudorandom number generation techniques. This hardware is capable of providing synthetic AER at a maximum peak rate of 10^7 eps (events per second) for images of size 64×64.

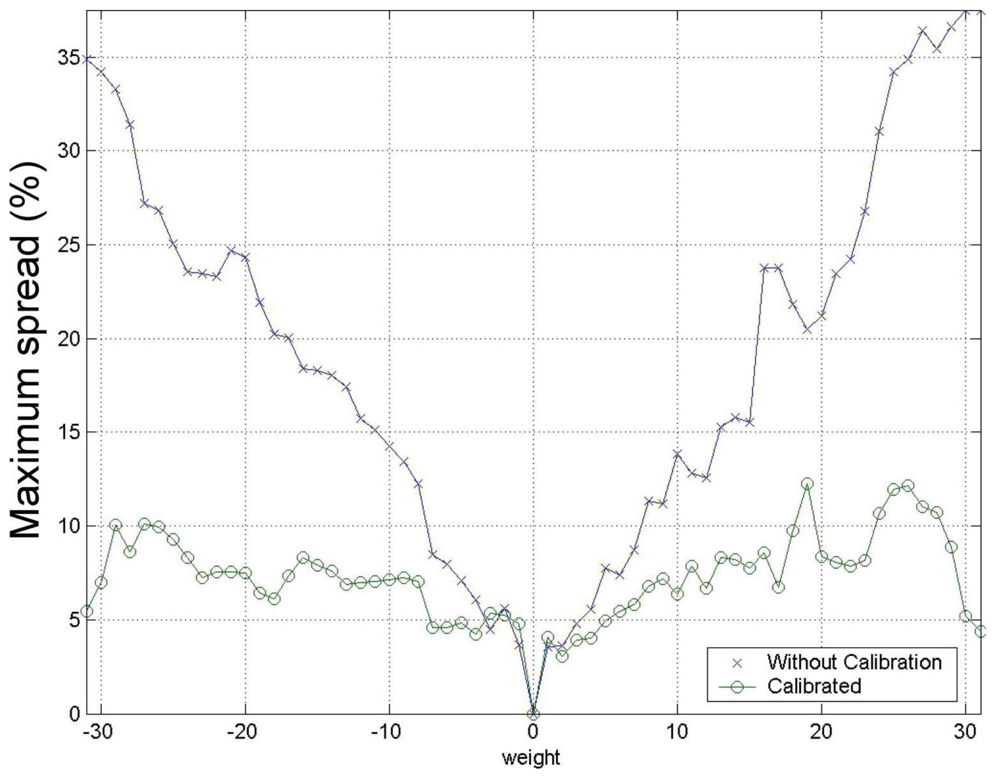


Fig. 15. Spread of pixel frequencies as a function of kernel weight w, before and after calibration.

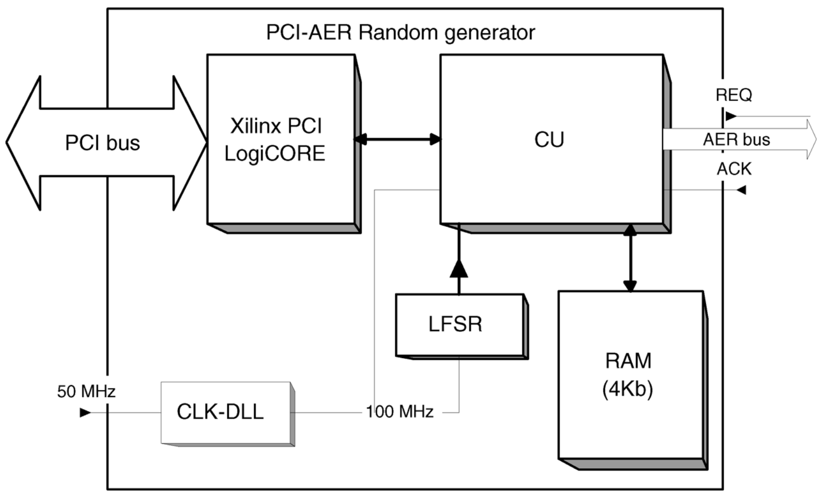


Fig. 16. Block diagram of the PCI-AER interface programmed onto FPGA.

The hardware interface (see Fig. 16) includes the Xilinx LogiCORE PCI core, a 4-kB RAM for storing the image to be converted to AER, a *control unit* (CU), a 20-bit LFSR, a delay line loop (DLL) for internal clock management, and a block for configuring PCI core and interfacing issues. The image frame (which can be of up to 64×64 pixels) is transferred from the computer through its PCI bus and the PCI core to the 4-kB RAM memory. The 20-bit LFSR is used for the pseudorandom number generation and is the core of the synthetic AER generation algorithm [56], which was later shown to obey Poisson statistics [57]. The 20 bits is a good compromise between hardware complexity and length of random numbers period. The CU, clocked at 100 MHz, is the operation center. The LFSR works at a slower speed, using a clock which is generated by the CU and triggered by the communication with the AER receiver.

Using this interface, we can feed the convolution chip with a moving or steady image of very well-known characteristics and analyze the chip response under different convolution kernels and configuration parameters, as described next.

C. *System Level Experimental Results*

The first experiment is to illustrate the effects of calibration. For this we fed the convolution chip with a uniform image and

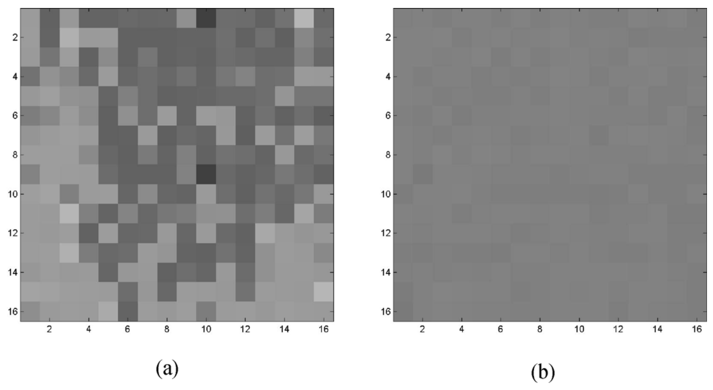


Fig. 17. Output of convolution chip for uniform input image and unity kernel. (a) Without calibration. (b) With calibration.

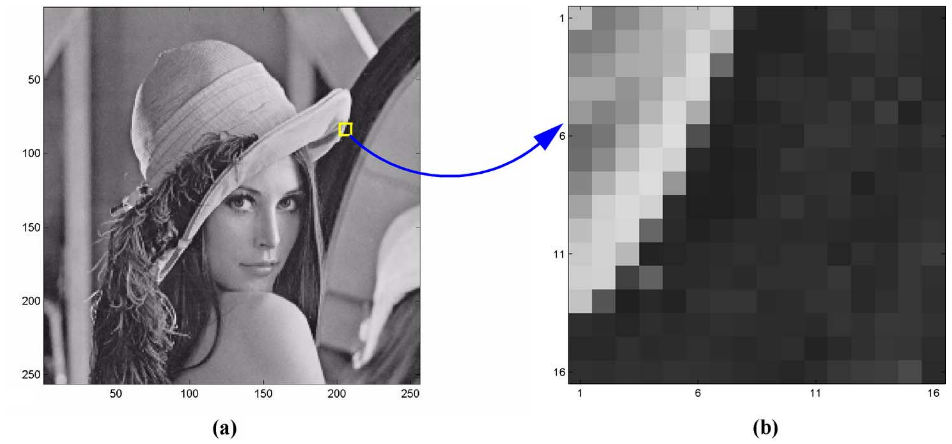
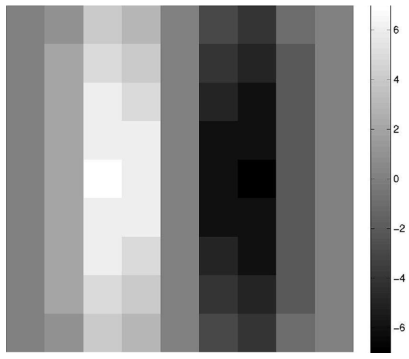


Fig. 18. From a natural image (a) we select a subframe of 16×16 pixels of high contrast.



0	1	4	3	0	−3	−4	−1	0
0	2	5	4	0	−4	−5	−2	0
0	2	6	5	0	−5	−6	−2	0
0	2	6	6	0	−6	−6	−2	0
0	2	7	6	0	−6	−7	−2	0
0	2	6	6	0	−6	−6	−2	0
0	2	6	5	0	−5	−6	−2	0
0	2	5	4	0	−4	−5	−2	0
0	1	4	3	0	−3	−4	−1	0

Fig. 19. Gabor kernel for vertical edge extraction.

programmed a kernel of size 1×1 with maximum weight. This makes the convolution chip to provide at the output the same image that is being received at the input, except for some additional scaling.[10] Fig. 17(a) shows the output of the convolution chip without calibrating the chip, while Fig. 17(b) shows the output for the same image and kernel after applying calibration. Both images reflect the difference between a pixel standard deviation of $\sigma = 14.2\%$ (see Fig. 15 for $w = -31$ before calibration) and $\sigma = 1.8\%$ (see Fig. 15 for $w = -31$ after calibration).

For the next experiments, we use a subframe of a natural image with an important contrast. Specifically, from the 256×256 pixels natural image in Fig. 18(a) we extracted the subframe of size 16×16, shown in Fig. 18(b). We loaded the chip with a kernel for vertical edge detection (as shown in Fig. 19) and applied the input image of Fig. 18(b) to our convolution chip,[11] using the interfacing hardware described in Section V-B. The second and third columns in Fig. 20 show the ideal output image that results from convolving the image in Fig. 18(b) with the kernel, as computed by Matlab.[12] Although the input image has only positive pixel values, the output image

[10]A 1×1 kernel is a Dirac delta convolution.

[11]We actually applied a larger size image to eliminate boundary effects of the convolution operation.

[12]Using its built-in 2-D convolution operator.

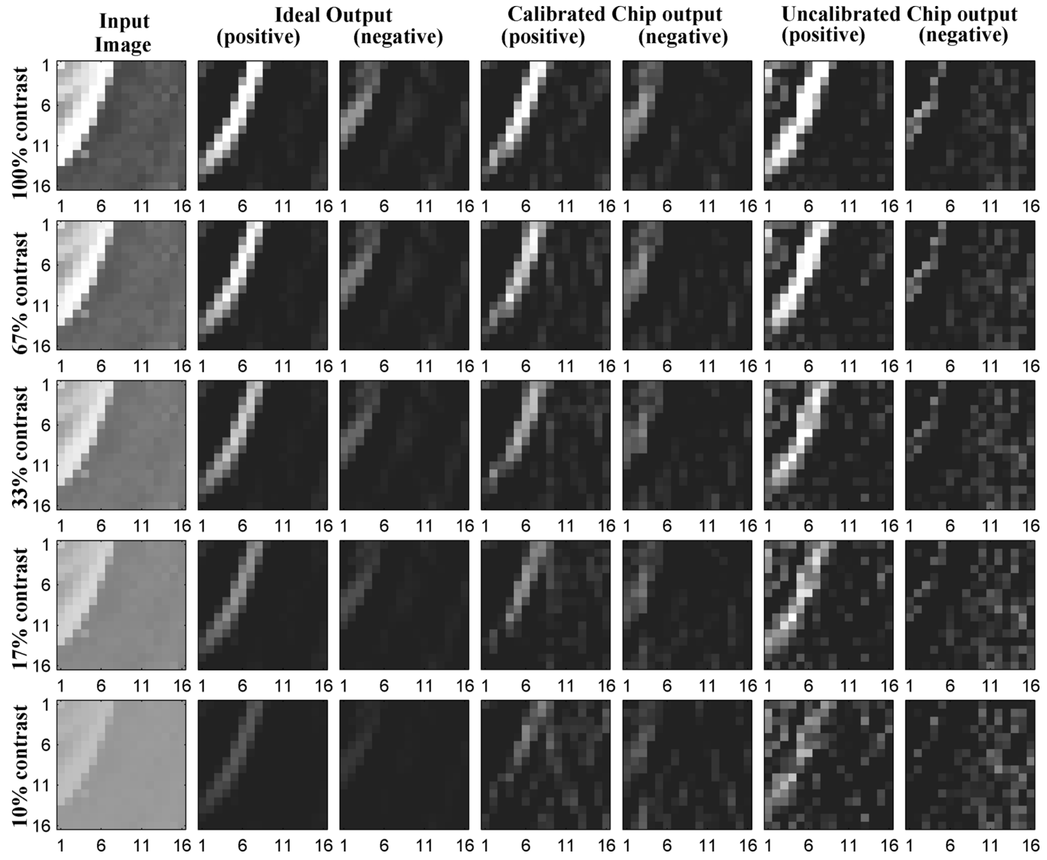


Fig. 20. Convolution processing for different contrast level input images.

has both positive and negative pixel values, because an edge detection kernel has also double sign values. The top row in Fig. 20 shows separately the pixels with positive output events and the ones with negative output events. The numbers around each subframe indicate pixel rows and columns (from 1 to 16). Now, using the image in Fig. 18(b) to generate a physical stream of address events with the hardware described in Section V-B, and feeding it as input AER to our convolution chip, properly programmed with the edge detection kernel, results in the output images in Fig. 20 (four right-most columns). These images are obtained by collecting the convolution chip output AER stream during a time of 40 ms and representing the number of events generated by each pixel. Two columns in Fig. 20 correspond to the case when the chip has been properly calibrated for pixel mismatch, while two other columns correspond to the case when no calibration is applied to the chip.

To further illustrate the effect of calibration, we repeated the same convolution operation while progressively reducing the contrast of the input image of Fig. 18(b). This is shown in Fig. 20. The first column shows the input image used, where the contrast has been reduced from the original 100% level progressively down to a 10% level. The second and third columns show the ideal output computed with Matlab, the fourth and fifth columns correspond to the chip output with calibration, and the sixth and seventh columns to the chip output without calibration. As can be seen, there is an important gain in performance as a consequence of applying calibration. In order to provide a quantitative measure, Fig. 21 shows the numerical values of row $n^{\circ}5$ of the images in Fig. 20. The numbers on the horizontal axes show the pixel column for this row. The first column in Fig. 21 represents the pixels event frequency for the input image (in kilohertz), the second column the mathematically computed ideal output[13] (in hertz), the third and fourth columns correspond to the measured chip output frequencies with and without calibration, respectively, (in hertz), and the last column represents the percent error of both outputs. Each inset indicates also the numerical average mean square error over all pixels of all columns for the calibrated (top) and uncalibrated (bottom) cases. If input event rate is increased/decreased, output rate scales linearly.

Current consumption of the chip varied between 20 and 50 mA, depending on the kernel size, the event throughput, and the event output rate. The *I-Pots* section alone consumed 10 mA because its $I_{\rm REF}$ value was set fairly high to 100 μA. An important part of the power consumption is caused by the AER output pads. A summary of chip performance and characteristics is given in Table I.

VI. Up-Scaling of AER-Based Convolution Processing Systems

So far we have shown the concept and experimental verification of a preliminary programmable kernel AER convolution chip of a small size. Now, we will discuss and illustrate how we can scale up this technique to build realistic vision processing

[13]To map Matlab's ideal outputs to a frequency value, we mapped the output range of Matlab's 2-D output to the experimental 2-D output frequency map, but only for the 100% contrast stimulus. The same mapping was then used for the rest of input contrast stimuli experiments.

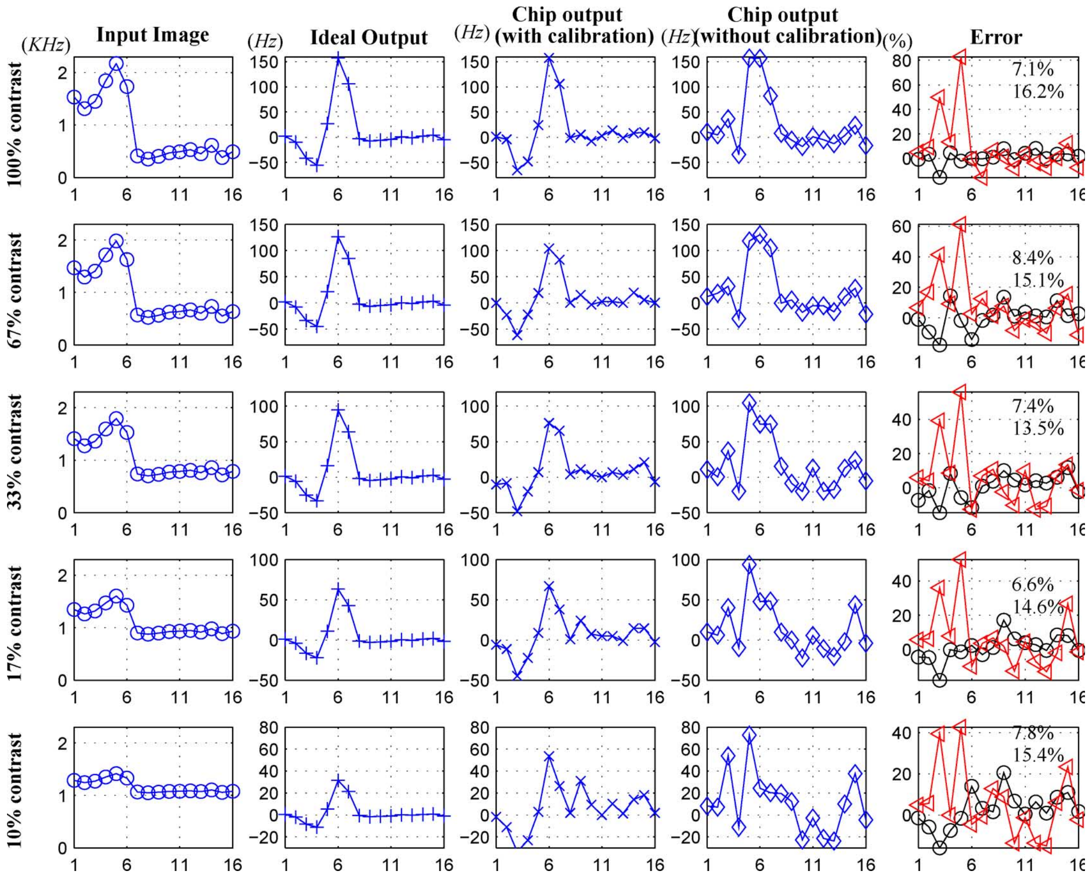


Fig. 21. Numerical values for the fifth column of the 16 × 16 images in Fig. 20.

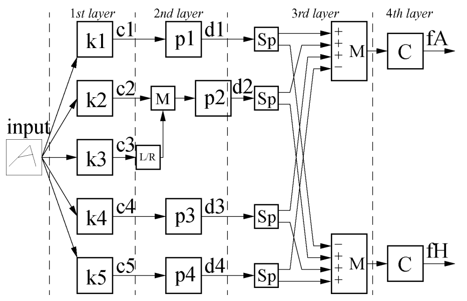


Fig. 22. Illustration of a multichip multilayer AER convolution processing systems to distinguish between handwritten characters "A" and "H." This system is loosely inspired in the neocognitron architecture.

systems. Up-scaling means growing the system capabilities in two separate "directions": 1) processing larger arrays of pixels and 2) process both in parallel and sequentially arbitrarily complex hierarchical multilayered cortical-like structures.

A. *Up-Scaling to Larger Arrays of Pixels*

The chip presented experimentally in this paper is a proof of concept chip of small size (16 × 16 pixels). Using the same pixel, it is feasible to design a larger chip with 64 × 64 pixels in the same technology. Such chip would have a size around 8 × 11 mm^2. If we want to use such a chip for processing arrays of pixels of a more realistic present day consumer video, we should be able to process at least arrays of 256 × 256 pixels or more. As discussed in Section III and illustrated in Fig. 3, it is possible to assemble chips for processing larger arrays of pixels. The intrinsic limitation will be the bandwidth of the AER links. For example, suppose we have available 64 × 64 pixel chips, capable of handling 10^7 eps, and we tile a 4 × 4 array of them for processing 256 × 256 pixels, sent by an AER retina. Such a retina should have its maximum output event rate not larger than 10^7 eps. Today's reported AER retinas include some internal preprocessing to reduce output event flow. For example, Lichtsteiner *et al.* [59] have developed a temporal contrast retina of 128 × 128 pixels to compute motion, whose maximum output event rate does not increase over 10^6 eps. An up-scaled 256 × 256 version would provide a maximum rate of 4 ×10^6 eps. Costas *et al.* [62] report on a spatial contrast retina setup of 32 × 32 pixels whose average output signed event rate is in the order of 10 keps. This would scale up to 160 keps for 256 × 256 pixels. Even if we use a 256 × 256 pixel retina without any preprocessing that would directly transform sensed pixel intensity into an event output stream for each pixel, we can adjust the retina so that pixel maximum event rate is maintained below 150 Hz. This would result in a maximum event rate for the complete retina of 9.8 Meps. Consequently, when scaling up image size, the maximum event rate of an AER sender should increase (because there are more pixels). This maximum rate, however, should be kept below the max rate capability of each convolution chip (as well as intermediate splitters). Usually, AER retinas can be adjusted to control the average pixel output event rate [59], [62]. However, lowering pixel event rate implies lowering the response time of any

TABLE I
SUMMARY OF CHIP PERFORMANCE AND CHARACTERISTICS

pixel array	16 x 16
pixel size	$100\mu m \times 140\mu m$
kernel max size	16 x 16
peak input event rate before queue is filled	50 Meps
sustained input event rate (depends on number of kernel rows)	2.8-16.7Meps @ 100MHz controller clock
output events handshaking cycle (shorting Rqst and Ack)	15ns
power consumption (depends on kernel size, input event rate and output event rate)	66.7 - 165 mW
pixel mismatch standard deviation (after calibration)	2%

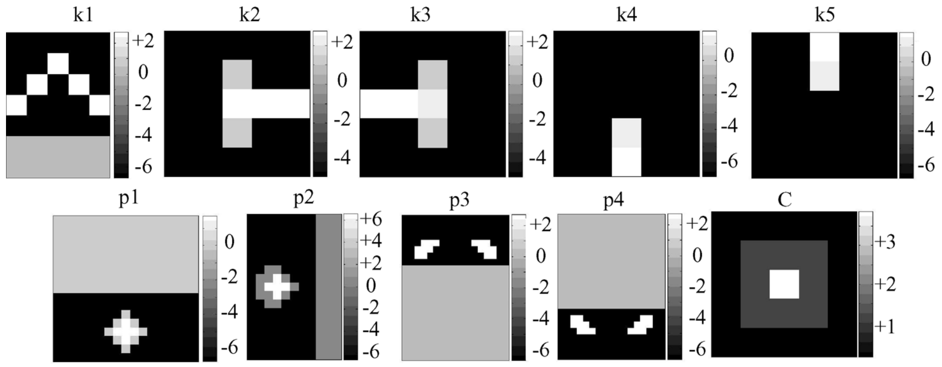


Fig. 23. Kernels used for the different convolutions in Fig. 22. The bar to the right side of each kernel shows the grayscale coding of the kernel value. For each kernel, “white” is assigned to its maximum value and “black” is assigned to its minimum value.

later processing. Consequently, there will always be a compromise between the array size, the event traffic, and the system response time.

B. Up-Scaling to Multilayer Cortical-Like Hierarchical Processing Structures

There is a solid framework of vision processing (software) based on local convolution operations, which has been developed in the late 1960s [3]–[10]. In recent years, this computational paradigm has been named “convolution neural networks.” The structure of a convolution neural network is similar to what is claimed for biological cortical structures [17]: a reduced number of sequential layers (8–10 in the human cortex), but each layer may include many different parallel convolution filters (of different shapes, scales, angles, etc.). Chips like the one presented in this paper can be easily assembled into multilayer structures with multichips per layer, by exploiting AER splitters and mergers conveniently. To illustrate this, we will show experimental results, obtained using our 16×16 convolution chip, of a simplified version of the neocognitron system reported in 1991 by Fukushima *et al.* [5]. Fukushima’s 1991 neocognitron consists of eight sequential layers, a total of 441 convolution filters each programmed with a different kernel, and it operates on an input black and white image of 19×19 pixels. Input images consist of a catalog of handwritten alphanumeric characters.

We have simplified this neocognitron to a structure of four sequential layers including 13 different 16×16 pixel convolutions, whose function is simply to distinguish between letters “A” and “H.” The structure is shown in Fig. 22. It receives an input visual stimulus (of 16×16 pixels), which can be either letter “A” or letter “H,” and it can tolerate slight deformations. The first processing layer performs five convolutions in parallel, of kernels ki ($i = 1$ to 5). These are shown on the top of Fig. 23. Kernels have positive and negative values. Therefore, the convolution outputs would include events with both positive and negative events. In the system of Fig. 22, convolution chips negative output events are ignored. Only positive events will be transmitted. Consequently, each convolution chip will compute a half-wave rectification, besides the programmed convolution.

Kernel $k1$ is intended to detect the presence and position of the upper peak in letter “A.” Kernel $k2$ detects the presence and position of a horizontal segment ending on the left and touching a vertical segment. Kernel $k3$ does the same, but ending on the right. Kernel $k4$ detects presence and position of the bottom end of a vertical segment, and kernel $k5$ does the same but for the top end. Consequently, the first layer of convolutions is intended to detect a set of five geometrical features, which can be used to detect and discriminate between letters “A” and “H.”

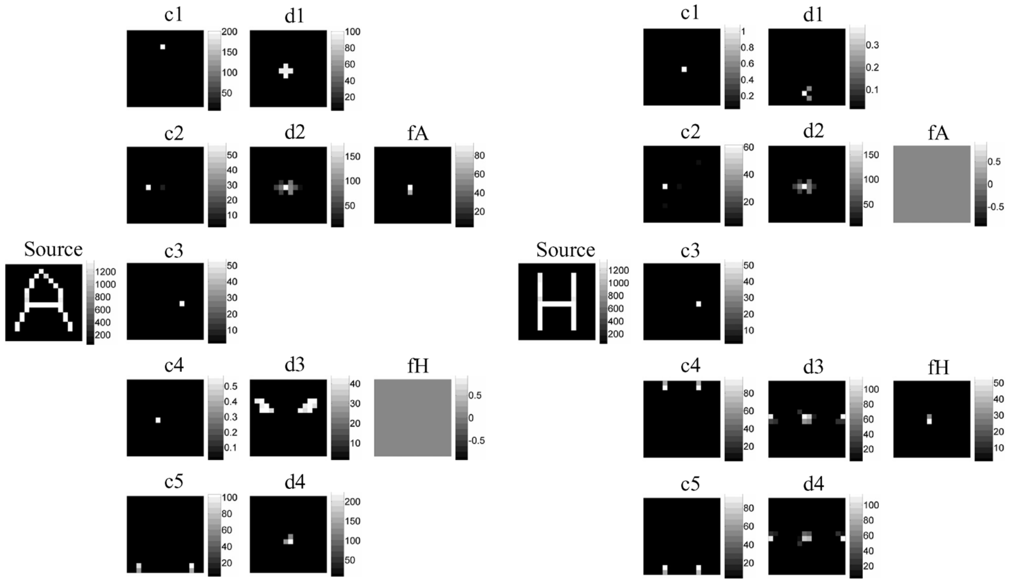


Fig. 24. Output frequencies produced by the different convolution stages pixels: On the left, when the input stimulus "A" was presented, and on the right when letter "H" was presented. For each convolution array, grayscale represents pixel frequency in hertz, as indicated by the scale bar on its right.

The second layer of the convolution processing is intended to evaluate whether the spatial relative positions of detected features in the first layer are meaningful for the character to be detected. For example, for letter "A," the top peak (detected by $k1$ and present at the output $c1$) should be in the upper part above all the other features. Consequently, kernel $p1$ will produce a positive contribution in the region below, because this would be the place in the output $d1$, where the center of letter "A" would be if all its features were detected simultaneously. In a similar manner, if there is the output at $c2$, the center of "A" could be to the right. Therefore, kernel $p2$ will add contribution to the pixels in $d2$, which are to the right of those that fired in $c2$. The output at $c3$ has to be treated more symmetrically than the one for $c2$. Therefore, we do not need to add an extra convolution chip for this. We can simply flip the left/right (block "L/R" in Fig. 22) output coordinates at $c3$ and use the same kernel $p2$ to evaluate the correct position of the feature detected by kernel $k3$. The flipped events of $c3$ and the ones of $c2$ are sequenced by a merger block (labeled M in Fig. 22) before feeding them to the convolution chip with kernel $p2$. Kernel $k4$ places events at $c4$ if a bottom end of the 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 $p3$. Kernel $p4$ operates in a similar manner, but for top ends of the vertical segments. Letter "A" should produce activity at outputs $\{c1, c2, c3, c4\}$, while letter "H" at $\{c2, c3, c4, c5\}$. When the input is the letter "A," the activity at $\{c1, c2, c3, c4\}$ will be on different pixels. However, the activity at $\{d1, d2, d3\}$ would be around the center of the letter "A." Similarly, if "H" is the input letter, the activity at $\{d2, d3, d4\}$ would show up around the center of the letter.

The purpose of the third layer is to add with positive or negative weight the outputs of the second layer. For letter "A," outputs $\{d1, d2, d3\}$ should contribute positively, while output $d4$ should inhibit. Similarly, for letter "H" $d1$ should inhibit, while $\{d2, d3, d4\}$ should contribute positively. Consequently, all outputs $d1$–$d4$ are split (blocks "Sp" in Fig. 22) into two separate pathways with two separate four-input merger blocks, one for detecting letter "A" and the other for letter "H." Only positive events come out at outputs $d1$–$d4$. However, the sign bits are hardwired at the inputs of the merger blocks, with the sign indicated in Fig. 22. The merger blocks simply sequence the events coming from their four input channels, and are fed to a convolution chip programmed with a 1×1 kernel. This way signed events are integrated at each pixel over time to obtain net pixel activity, which is also rectified.

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.

Currently, we do not have available 13 convolution chips and the large number of splitters/mergers required to assemble physically the structure of Fig. 22. However, we can stimulate a single chip with a specific stimulus, record its output event stream, play back this output, use it as a stimulus for the chip after programming it with a different kernel, record again its output, and so on [63]. This way we can obtain the experimental behavior of the complete structure. The results are shown in Fig. 24, where each pixel array represents the convolution chip pixels output frequencies. Fig. 24 shows that when presenting the input stimulus "A," there is the output activity at "fA" and zero activity at "fH," meaning that letter "A" has been recognized. Similarly, when the input stimulus is "H," there is zero output at "fA," while there is the output activity at "fH." The output activity appears at the pixels which are at the location of the center of the input letter.

C. Discussion

An interesting property of AER cortical structures, like the one in Fig. 22, is that the processing delay depends on the number of layers and the number of events that carry meaningful information. As systems scale up to perform more sophisticated processing, the total number of layers does not grow much (human cortex has 8–10 layers [17]). What grows is the number of parallel convolutions per layer [3]–[10], and this does not slow down the global delay. Also, AER sensors send first the most relevant events: for example, a motion retina will send out first the events produced by those pixels that have sensed a faster transition [59]; a contrast retina will provide first the events for pixels with highest contrast [62]. Consequently, as the most relevant events appear first, they are processed by the first layer first, the most relevant features are detected first, and so on. Therefore, object recognition can be very fast, by processing only a small percentage of the total number of events [56], [64]. For the system in Fig. 22, the delay between pattern presentation and recognition can be as low as[14] 3 μs. Consequently, if we assemble Fukushima's neocognitron [5], which has 441 convolutions in eight sequential layers, the neocognitron delay would be about doubled (because the number of layers is doubled and the number of events flowing in each link would be similar as for the system in Fig. 22).

For processing more sophisticated stimuli, larger pixel arrays are required, but the number of sequential layers will be similar, as in the case of face recognition tasks [8]–[10]. In such cases, many more events are required to represent meaningful information. For example, a 64×64 pixel face with an 8-bit resolution per pixel can be represented by a 500 kiloevent stream [56], which would require a transfer time of 50 ms for an AER channel with a bandwidth of 10 Meps. However, the most relevant features are already available in the first 10% of the events [56], thus requiring only 5 ms. A contrast representation of the same face needs only 28% of the events of the original face, and the first 7.5 kiloevents (75 μs) are sufficient for the recognition [56]. Consequently, an eight-layer AER cortical structure would require less than $8 \times 75\ \mu\text{s} = 600\ \mu\text{s}$ for such processing, independently of the number of convolution operations. Scaling this to 256×256 pixels would result in delays of about 16 times slower, in the order of 10 ms. Performing this task with conventional frame-based image convolutions, using, for example, the Öwall's special convolution hardware [12], would require 55 ms per convolution, plus the overhead for communicating images (3 ms per image) and the necessary image additions/subtractions. If around 500 convolutions are needed for the face recognition task, this would result in a total delay of around 30 s, when using one single convolution chip. If using one chip per convolution, because chips are structured in eight sequential layers, the total delay would be $(55\text{ ms} + 3\text{ ms}) \times 8 = 465$ ms, ignoring additions/subtractions.

VII. Conclusion

A convolution test chip prototype based on AER has been presented, fabricated, and tested. The purpose of this first prototype was to test different system components and operation principles as well as performance. The chip design is based on digital calibration of analog computing circuits. For this reason, a small size (3.3×4.2 mm^2) chip was fabricated in a 0.35-μm CMOS process, for processing images of size 16×16. The chip can be programmed to perform kernels of arbitrary shape and of size up to 16×16. Extensive experimental results are provided that demonstrate the correct operation of the chip, and the potential of AER for performing real-time convolutions. As a first test prototype, the resulting pixel size of $100 \times 140\ \mu\text{m}^2$ was conservatively oversized to allow kernel weights of up to 6-bit resolution. Consequently, for this pixel size, and using the presented circuit techniques, it is feasible to build convolution chips of 64×64 pixel arrays (and larger), with programmable convolution kernels of 64×64 or larger, in an area of less than 1 cm^2. In future prototypes [54], a significant reduction in pixel area will be expected after reducing kernel weight resolution. Also, a significant percentage of pixel area will be consumed by the in-pixel calibration circuitry. Currently, we are investigating other calibration techniques to further reduce pixel area while maintaining or improving precision performance [65]. Nonetheless, using the present calibration technique and reducing kernel weight resolution, it should be feasible to fabricate convolution chips for images of size 128×128 in 0.35-μm CMOS. Kernel can be of large size, equal to image size or larger. Kernel and image sizes are independent of each other.

The processing power of such AER-based convolution chips becomes apparent when using them for multilayer cortical-like processing systems. This is because the processing delay depends on the number of layers but not on the complexity of each layer. Consequently, sophisticated but fast processing is possible, as in biological cortical structures, where there is a reduced number of layers (8–10).

[14]This was estimated through behavioral simulations of the system in Fig. 22 [58].

References

[1] S. Grossberg, E. Mingolla, and J. Williamson, "Synthetic aperture radar processing by a multiple scale neural system for boundary and surface representation," *Neural Netw.*, vol. 8, no. 7/8, pp. 1005–1028, 1995.

[2] S. Thorpe, D. Fize, and C. Marlot, "Speed of processing in the human visual system," *Nature*, vol. 381, no. 6582, pp. 520–2, Jun. 1996.

[3] K. Fukushima, "Visual feature extraction by a multilayered network of analog threshold elements," *IEEE Trans. Syst. Sci. Cybern.*, vol. SSC-5, no. 4, pp. 322–333, 1969.

[4] K. Fukushima, "Neocognitron: A self-organizing neural-network model for a mechanism of pattern recognition unaffected by shift in position," *Biol. Cybern.*, vol. 36, pp. 193–202, 1980.

[5] K. Fukushima and N. Wake, "Handwritten alphanumeric character recognition by the neocognitron," *IEEE Trans. Neural Netw.*, vol. 2, no. 3, pp. 355–365, May 1991.

[6] Y. LeCun and Y. Bengio, "Convolutional networks for images, speech, and time series," in *The Handbook of Brain Science and Neural Networks*, M. Arbib, Ed. Cambridge, MA: MIT Press, 1995, pp. 255–258.

[7] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proc. IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov. 1998.

[8] C. Neubauer, "Evaluation of convolution neural networks for visual recognition," *IEEE Trans. Neural Netw.*, vol. 9, no. 4, pp. 685–696, Jul. 1998.

[9] S. Lawrence, C. L. Giles, A. Tsoi, and A. Back, "Face recognition: A convolutional neural network approach," *IEEE Trans. Neural Netw.*, vol. 8, no. 1, pp. 98–113, Jan. 1997.

[10] B. Fasel, "Robust face analysis using convolution neural networks," in *Proc. Int. Conf. Pattern Recognit.*, Quebec, QC, Canada, Aug. 11–15, 2002, vol. 2, pp. 40–43.

[11] A. Gentile and D. S. Wills, "Portable video supercomputing," *IEEE Trans. Comput.*, vol. 53, no. 8, pp. 960–972, Aug. 2004.

[12] V.wall, M. Torkelson, and P. Egelberg, "A custom image convolution DSP with a sustained calculation capacity of >1 GMAC/s and low I/O bandwidth," *J. VLSI Signal Process.*, vol. 23, pp. 355–349, 1999.

[13] H. Kwon, "A low-power image convolution algorithm for variable voltage processors," in *Proc. IEEE Int. Conf. Acoust. Speech Signal Process.*, 2003, vol. 2, pp. 677–680.

[14] H. H. Cut, A. Gentile, J. C. Eble, M. Lee, O. Vendier, Y. J. Joo, D. S. Wills, M. Brooke, N. M. Jokerst, and A. S. Brown, "SIMPiL: An OE integrated SIMD architecture for focal plane processing applications," in *Proc. 3rd Int. Conf. Massively Parallel Process. Using Opt. Interconnects*, 1996, pp. 44–52.

[15] F. Paillet, D. Mercier, and T. M. Bernard, "Making the most of $15k\lambda^2$ silicon area for a digital retina PE," in *Proc. SPIE Adv. Focal Plane Arrays Electron. Cameras II*, Zurich, Switzerland, May 1998, vol. 3410, pp. 158–167.

[16] R. Etienne-Cummings, Z. K. Kalayjian, and D. Cai, "A programmable focal-plane MIMD image processor chip," *IEEE J. Solid-State Circuits*, vol. 36, no. 1, pp. 64–73, Jan. 2001.

[17] G. M. Shepherd, *The Synaptic Organization of the Brain*, 3rd ed. Oxford, U.K.: Oxford Univ. Press, 1990.

[18] M. Sivilotti, "Wiring considerations in analog VLSI systems with application to field-programmable networks," Ph.D. dissertation, Comp. Sci. Div., California Inst. Technol., Pasadena, CA, 1991.

[19] M. Mahowald, "VLSI analogs of neural visual processing: A synthesis of form and function," Ph.D. dissertation, Comp. Sci. Div., California Inst. Technol., Pasadena, CA, 1992.

[20] M. Mahowald, *An Analog VLSI Stereoscopic Vision System*. Norwell, MA: Kluwer, 1994.

[21] A. Mortara and E. A. Vittoz, "A communication architecture tailored for analog VLSI artificial neural networks: Intrinsic performance and limitations," *IEEE Trans. Neural Netw.*, vol. 5, no. 3, pp. 459–466, May 1994.

[22] A. Mortara, E. A. Vittoz, and P. Venier, "A communication scheme for analog VLSI perceptive systems," *IEEE J. Solid-State Circuits*, vol. 30, no. 6, pp. 660–669, Jun. 1995.

[23] P. Vernier, A. Mortara, X. Arreguit, and E. A. Vittoz, "An integrated cortical layer for orientation enhancement," *IEEE J. Solid-State Cicuits*, vol. 32, no. 2, pp. 177–186, Feb. 1997.

[24] E. Culurciello and A. G. Andreou, "A comparative study of access topologies for chip-level address-event communication channels," *IEEE Trans. Neural Netw.*, vol. 14, no. 5, pp. 1266–1277, Sep. 2003.

[25] T. Y. W. Choi, B. E. Shi, and K. Boahen, "An ON-OFF orientation selective address event representation image transceiver chip," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 51, no. 2, pp. 342–353, Feb. 2004.

[26] T. Y. W. Choi, P. A. Merolla, J. V. Arthur, K. A. Boahen, and B. E. Shi, "Neuromorphic implementation of orientation hypercolummns," *IEEE Trans. Circ. Syst. I, Reg. Papers*, vol. 52, no. 6, pp. 1049–1060, Jun. 2005.

[27] J. Kramer, R. Sarpeshkar, and C. Koch, "Pulse-based analog velocity sensors," *IEEE Trans. Circuits Syst. II, Analog Digit. Signal Process.*, vol. 44, no. 2, pp. 86–101, Feb. 1997.

[28] E. Culurciello, R. Etienne-Cummings, and K. Boahen, "A biomorphic digital image sensor," *IEEE J. Solid-State Circuits*, vol. 38, no. 2, pp. 281–294, Feb. 2003.

[29] T. Serrano-Gotarredona, A. G. Andreou, and B. Linares-Barranco, "AER image filtering architecture for vision-processing systems," *IEEE Trans. Circuits Syst. I, Fundam. Theory Appl.*, vol. 46, no. 9, pp. 1064–1071, Sep. 1999.

[30] T. Serrano-Gotarredona, A. G. Andreou, and B. Linares-Barranco, "A programmable VLSI filter architecture for application in real-time vision processing systems," *Int. J. Neural Netw.*, vol. 10, no. 3, pp. 179–190, Jun. 2000.

[31] J. Lazzaro, J. Wawrzynek, M. Mahowald, M. Sivilotti, and D. Gillespie, "Silicon auditory processors as computer peripherals," *IEEE Trans. Neural Netw.*, vol. 4, no. 3, pp. 523–528, May 1993.

[32] S. Thorpe and J. Gautrais, "Rank order coding," in *Proc. 6th Annu. Conf. Comput. Neurosci.: Trends Res.*, 1998, pp. 113–118.

[33] K. Boahen, "Retinomorphic vision systems," in *Proc. 5th Int. Conf. Microelectron. Neural Netw. Fuzzy Syst.*, Lausanne, Switzerland, Feb. 1996, pp. 2–14.

[34] W. Maass and C. M. Bishop, *Pulsed Neural Networks*. Cambridge, MA: MIT Press, 1999.

[35] K. Boahen, "Point-to-point connectivity between neuromorphic chips using address events," *IEEE Trans. Circuits Syst. II, Analog Digit. Signal Process.*, vol. 47, no. 5, pp. 416–434, May 2000.

[36] K. Boahen, "A throughput-on-demand address-event transmitter for neuromorphic chips," in *Proc. 20th Anniversary Conf. Adv. Res. VLSI*, D. S. Wills and S. P. DeWeerth, Eds., 1999, pp. 72–86.

[37] K. Shimonomura and T. Yagi, "A multichip aVLSI system emulating orientation selectivity of primary visual cortical cells," *IEEE Trans. Neural Netw.*, vol. 16, no. 4, pp. 972–979, Jul. 2005.

[38] D. H. Goldberg, G. Cauwenberghs, and A. G. Andreou, "Probabilistic synaptic weighting in a reconfigurable network of VLSI integrate-and-fire neurons," *Neural Netw.*, vol. 14, pp. 781–793, 2001.

[39] R. J. Vogelstein, U. Mallik, E. Culurciello, G. Cauwenberghs, and R. Etienne-Cummings, "Saliency-driven image acuity modulation on a reconfigurable silicon array of spiking neurons," in *Advances in Neural Information Processing Systems (NIPS'2004)*. Cambridge, MA: MIT Press, 2005, vol. 17.

[40] J. P. Lazzaro and J. Wawrzynek, W. J. Dally, J. W. Poulton, and A. T. Ishii, Eds., "A multi-sender asynchronous extension to the address-event protocol," in *Proc. 16th Conf. Adv. Res. VLSI*, 1995, pp. 158–169.

[41] *IEEE Standard VHDL Language Reference Manual*, IEEE Std 1076-1993 and Std 1076a-2000, 2000.

[42] H. K. Yang and E. I. El-Masry, "Clock feedthrough analysis and cancellation in current sample/hold circuits," *Inst. Electr. Eng. Proc. Circuits Devices Syst.*, vol. 141, no. 6, pp. 510–516, 1994.

[43] B. Linares-Barranco, T. Serrano-Gotarredona, and R. Serrano-Gotarredona, "A new charge-packet driven mismatch-calibrated integrate-and-Fire neuron for processing positive and negative signals in AER based systems," in *Proc. IEEE 2004 Int. Symp. Circuits Syst.*, 2004, vol. 5, pp. 23–26.

[44] B. Linares-Barranco and T. Serrano-Gotarredona, "On the design and characterization of femtoampere current-mode circuits," *IEEE J. Solid-State Circuits*, vol. 38, no. 8, pp. 1353–1363, Aug. 2003.

[45] T. Serrano-Gotarredona and B. Linares-Barranco, "CMOS mismatch model valid from weak to strong inversion," in *Proc. Eur. Solid State Circuits Conf.*, Sep. 2003, pp. 627–630.

[46] B. Linares-Barranco, T. Serrano-Gotarredona, and R. Serrano-Gotarredona, "Compact low-power calibration mini-DACs for neural massive arrays with programmable weights," *IEEE Trans. Neural Netw.*, vol. 14, no. 5, pp. 1207–1216, Sep. 2003.

[47] N. H. E. Weste and K. Eshraghian, "Principles of CMOS VLSI design," in *A Systems Perspective*, 2nd ed. Reading, MA: Addison-Wesley, 1993.

[48] K. Bult and J. G. M. Geelen, "An inherently linear and compact MOST-only current division technique," *IEEE J. Solid State Circuits*, vol. 27, no. 12, pp. 1730–1735, Dec. 1992.

[49] T. Delbruck and P. Lichtsteiner, "Fully programmable bias current generator with 24 bit resolution per bias," in *Proc. IEEE Int. Symp. Circuits Syst.*, Kos, Greece, May 2006, pp. 2849–2852.

[50] R. Serrano-Gotarredona, L. Camuas-Mesa, T. Serrano-Gotarredona, J. A. Leero-Bardallo, and B. Linares-Barranco, "The stochastic I-pot: A circuit block for programming bias currents," *IEEE Trans. Circuits Syst. II, Brief Papers*, vol. 54, no. 9, pp. 760–764, Sep. 2007.

[51] C. C. Enz and E. A. Vittoz, "CMOS low-power analog circuit design," in *Proc. Int. Symp. Circuits Syst.*, 1996, Tutorials 1.2, pp. 79–132.

[52] R. Serrano-Gotarredona, T. Serrano-Gotarredona, and B. Linares-Barranco, "On-event generators for address event representation transmitters," in *Proc. SPIE Microtechnol. New Millennium*, Sevilla, Italy, May 2005, vol. 5839, pp. 148–159.

[53] A. Martin, "Programming in VLSI: From communicating processes to delay-insensitive circuits," in *Proc. UT Year Program. Inst. Concurrent Program.*, Reading, MA, 1990, pp. 1–64.

[54] R. Serrano-Gotarredona, T. Serrano-Gotarredona, A. Acosta-Jimenez, and B. Linares-Barranco, "A neuromorphic cortical-layer microchip for spike-based event processing vision systems," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 53, no. 12, pp. 2548–2566, Dec. 2006.

[55] T. Serrano-Gotarredona and B. Linares-Barranco, "A real-time clustering microchip neural engine," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 4, no. 2, pp. 195–209, Jun. 1996.

[56] A. Linares-Barranco, G. Jimenez-Moreno, B. Linares-Barranco, and A. Civit-Ballcels, "On algorithmic rate-coded AER generation," *IEEE Trans. Neural Netw.*, vol. 17, no. 3, pp. 771–788, May 2006.

[57] A. Linares-Barranco, M. Oster, D. Cascado, G. Jimnez, A. Civit, and B. Linares-Barranco, "Inter-spike-intervals analysis of AER Poisson like generator hardware," *Neurocomputing*, vol. 70, pp. 2692–2700, May 2007.

[58] J. A. Pérez-Carrasco, T. Serrano-Gotarredona, C. Serrano-Gotarredona, B. Acha, and B. Linares-Barranco, "High-speed character recognition system based on a complex hierarchical AER architecture," presented at the IEEE Int. Conf. Circuits Syst., Seattle, WA, May 18–21, 2008.

[59] P. Lichtsteiner, C. Posch, and T. Delbruck, "A 128 × 128 120 dB 30 mW asynchronous vision sensor that responds to relative intensity change," in *IEEE Int. Solid-State Circuit Conf. Dig. Tech. Papers* , San Francisco, CA, 2006, pp. 508–509.

[60] K. A. Zaghloul and K. Boahen, "Optic nerve signals in a neuromorphic chip: Part 1," *IEEE Trans. Biomed Eng.*, vol. 51, no. 4, pp. 657–666, Apr. 2004.

[61] K. A. Zaghloul and K. Boahen, "Optic nerve signals in a neuromorphic chip: Part 2," *IEEE Trans. Biomed Eng.*, vol. 51, no. 4, pp. 667–675, Apr. 2004.

[62] J. Costas-Santos, T. Serrano-Gotarredona, R. Serrano-Gotarredona, and B. Linares-Barranco, "A spatial contrast retina with on-chip calibration for neuromorphic spike-based AER vision systems," *IEEE Trans. Circuts Syst., I, Reg. Papers*, vol. 54, no. 7, pp. 1444–1458, Jul. 2007.

[63] F. Gmez-Rodrguez *et al.*, “AER tools for communications and debugging,” in *Proc. IEEE Int. Symp. Circuts Syst.*, May 2006, pp. 3253–3256.

[64] S. Thorpe, A. Delorme, and R. V. Rullen, “Spike-based strategies for rapid processing,” *Neural Netw.*, vol. 14, pp. 715–725, 2001.

[65] J. A. Leñero-Bardallo, T. Serrano-Gotarredona, and B. Linares-Barranco, “A calibration technique for very low current and compact tunable neuromorphic cells. Application to 5–bit 20 nA DACs,” *IEEE Trans. Circuits Syst. II, Exp. Briefs*, 2008, to be published.

involved in different national and European R&D projects. His current research interests are in the areas of complimentary metal–oxide–semiconductor (CMOS) digital and mixed-signal very large scale integration (VLSI) design, low-power and low-noise CMOS, description of timing phenomena in VLSI digital system, and asynchronous and self-timed circuits.

Dr. Acosta was General Chair of the 2002 PATMOS International Workshop.