A 32 × 32 Pixel Convolution Processor Chip for Address Event Vision Sensors With 155 ns Event Latency and 20 Meps Throughput

Luis Camuñas-Mesa, Antonio Acosta-Jiménez, Carlos Zamarreño-Ramos, Teresa Serrano-Gotarredona, *Member, IEEE*, and Bernabé Linares-Barranco, *Fellow, IEEE*

***Abstract*—This paper describes a convolution chip for event-driven vision sensing and processing systems. As opposed to conventional frame-constraint vision systems, in event-driven vision there is no need for frames. In frame-free event-based vision, information is represented by a continuous flow of self-timed asynchronous events. Such events can be processed on the fly by event-based convolution chips, providing at their output a continuous event flow representing the 2-D filtered version of the input flow. In this paper we present a 32 × 32 pixel 2-D convolution event processor whose kernel can have arbitrary shape and size up to 32 × 32. Arrays of such chips can be assembled to process larger pixel arrays. Event latency between input and output event flows can be as low as 155 ns. Input event throughput can reach 20 Meps (mega events per second), and output peak event rate can reach 45 Meps. The chip can be configured to discriminate between two simulated propeller-like shapes rotating simultaneously in the field of view at a speed as high as 9400 rps (revolutions per second). Achieving this with a frame-constraint system would require a sensing and processing capability of about 100 K frames per second. The prototype chip has been built in 0.35 μm CMOS technology, occupies 4.3 × 5.4 mm^2 and consumes a peak power of 200 mW at maximum kernel size at maximum input event rate.**

***Index Terms*—Address event representation, convolution processing, event-based processing, feature extraction, frame-less vision, neuromorphic circuits and systems, object recognition, vision processing.**

I. Introduction

BIO-INSPIRED spiking signal address-event systems have attracted considerable attention in recent years, due to their fast sensing capability, reduced information throughput, efficient in-sensor preprocessing, and small per-event power consumption [1]–[8]. For example, a total of eight papers on event-driven sensing have been accepted to the IEEE International Solid State Circuits Conference since 2003, six based on vision [1]–[6], and two on audition [7], [8]. In this paper, we extend this previous work and demonstrate an event-based convolution chip, which is capable of performing higher order operations that are commonly found in feed-forward neural network architectures.

Early work on address-event-representation (AER) processing, originally pioneered by Carver Mead at Caltech almost twenty years ago [9], [10], combined asynchronous digital design with massive parallel analog computation and high transistor mismatch. Building on this work, during the next ten years a few sparse research groups reported AER sensory systems. For example, Lazzaro's [11] and Johns Hopkins University [12] pioneering work on audition, or Boahen's early developments on retinas [13], [14]. However, during these years some basic developments were accomplished such as a better understanding of asynchronous design [15], [16] leading to robust unarbitrated [17] and arbitrated [18], [19] asynchronous event readout, which combined with the availability of user-friendly FPGA external support for interfacing together with new submicrometer technologies allowing complex pixels in reduced areas, might be triggering a new trend in AER bio-inspired Spiking Sensor developments. Since 2003 many researchers have embraced this trend. AER has been used fundamentally in vision (retina) sensors, such as simple light intensity to frequency transformations [6], [20], time-to-first-spike coding [21], [22], foveated sensors [23], spatial contrast [4], [5], [24], [25], temporal contrast [1], [3], [6], [26], motion sensing and computation [13], and combined spatial and temporal contrast sensing [27], [28]. But AER has also been used for auditory systems [7], [8], [11], [12], [29], competition and winner-takes-all networks [30], [31], and even for systems distributed over wireless networks [32].

But sensing is just the initial part of the game. The next obvious step is to develop Spiking Signal Event Representation techniques capable of efficiently processing the signal flow coming from the sensors. For simple per-event heuristic processing and filtering, direct software based solutions can be used [33], [34]. Other schemes rely on lookup table rerouting and event repetitions followed by single-event integration [35]. Alternatively, we can find some pioneering work in the literature aiming at performing convolutional filtering on the AER flow produced by spiking retinas, in an attempt to mimic the early processing of visual cortex [36], [37]: Arreguit developed AER convolutional filters with elliptic-like kernels [38] while Choi reported more sophisticated Gabor like filters [39]. In both cases the shape of the filter kernel was hardwired (either elliptic or Gabor), although some parameters were tunable allowing slight reshaping of the 2-D kernel. In 1999 Serrano reported an AER architecture [40] that would allow more generic kernels, although with some geometric symmetry restrictions. It was not until 2006 that the same group reported a working AER Convolution chip with arbitrary shape programmable kernel of size

This work was supported by EU Grant 216777 (NABAB), Spanish Grants (with support from the European Regional Development Fund) TEC2006-11730-C03-01 (SAMANTA2) and TEC2009-10639-C04-01 (VULCANO), and Andalusian Grant P06TIC01417 (Brain System). The work of C. Zamarreño-Ramos was supported by a national FPU scholarship. Date of publication November 11, 2010; date of current version March 30, 2011. This paper was recommended by Associate Editor B. Shi.

The authors are with the Instituto de Microelectrónica de Sevilla (IMSE-CNM-CSIC). 41092 Seville, Spain (e-mail: bernabe@imse-cnm.csic.es).

up to 32×32 pixels preloaded onto an internal kernel-RAM [41], [42]. This opened the possibility of implementing in AER spiking hardware generic convolutional neural networks [43]–[45], where large number of convolutional modules with arbitrary size and shape kernels are required.

In the past, the authors have developed some AER ConvChips [41], [42] whose pixels performed kernel-dependent weighted aggregation by means of analog low current switching circuitry and capacitive charge accumulation. Those mixed analog-digital ConvChips required pixel-level calibration for transistor mismatch compensation and cumbersome biasing of analog components. Also, although they used 5-bit calibration, only 3-bit computing precision was achieved. As a result, they were tricky to set up and use, had low precision, were little robust, and required a strong expertize to operate them successfully. Consequently, they were not adequate for use by other people, nor to build large-scale multi-ConvChip systems. This motivated the development of a fully-digital ConvChip module, which we present in this paper. The main advantages of this chip over its mixed analog-digital predecessors are: a) no need for calibration; b) improved precision limited by the length of implemented registers; c) event latencies almost four orders of magnitude faster; and d) easier to set up and use.

The paper is structured as follows. The next section summarizes the history of ConvNets and their applications. Section III explains the potential advantages of implementing ConvNets with Spiking Hardware. Section IV quickly reviews the architecture of the proposed AER convolution chip, which is very similar to that of previous mixed signal designs, except for a few details. Section V describes the new pixel, and Section VI provides extensive experimental characterizations of the fabricated prototype. Finally, Section VII provides discussion and conclusions.

II. Convolutional Neural Networks

Convolutional neural networks (ConvNets) is a computational paradigm, inspired in the Nobel Prize award winning findings of Hubel and Wiesel on projection field processing in early stages of visual cortex [36], that has been developing as software algorithms for performing object recognition on conventional bitmap images, without any notion of spiking signal representation. ConvNets were originally proposed by Fukushima in 1969 [43], [44] and further developed by Yann LeCun [45], [46] and other groups [47]–[49], as a type of continuous-time gradient-based learning neural paradigm, with great success in a variety of (industrial) applications as well as research. Examples of industrial applications and developments are, to mention a few: 1) the early developments at AT&T/Lucent-Technologies/NCR [50] leading to check reading ATM machines; 2) Microsoft OCR and handwritten character recognition systems [51]; 3) Thomson developments in face/object recognition [52]; 4) France Telecom/Orange with face detection and recognition, text detection and recognition [53]–[55]; or 5) Google developments for detecting faces and license plates in StreetView images [56]. Examples of state-of-the-art research exploiting ConvNets are: 1) Poggio at MIT with object recognition and scene analysis [57]; 2) Seung at MIT with image segmentation, and biological image analysis (brain circuit reconstruction) [58]; 3) hands/gesture detection [59]; 4) vision based obstacle avoidance [60]; and 5) image restoration and segmentation [61], [62], and texture classification [63].

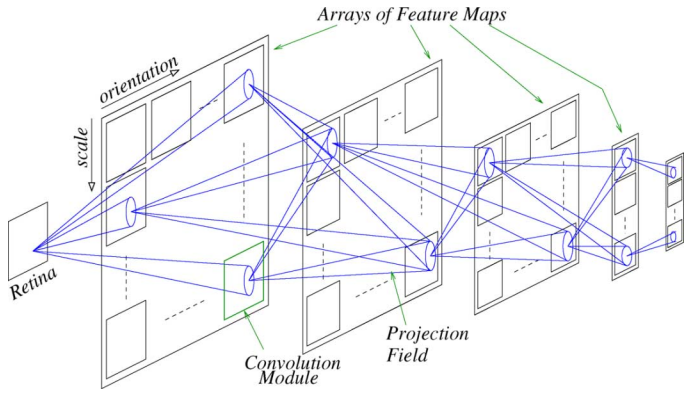

Fig. 1. Typical hierarchical structure of a feed forward convolutional network.

Fig. 1 shows a typical hierarchical structure of a feed forward ConvNet. It contains a sequence of layers, where each layer includes an array of *Feature Maps*. Feature Maps extract specific features from the visual flow coming from the previous layer, by performing convolutions with specific kernels. For example, the first layer immediately after the retina could be a Gabor filter bank for different angles and scales for extracting oriented segments. The second layer combines the extracted segments of the first layer to detect more sophisticated shapes. So does the third layer on the second layer extracted features, and so on, until the last layer performs specific object recognition tolerating degrees of deformations and sizes.

III. Advantagesof Spiking ConvNets: Pseudosimultaneity, Scalability, and Power Efficiency

Adapting the well established ConvNet computational paradigm to Spiking Signal Event based representations yields some very interesting properties. The first one is the very reduced latency between the input and output event flows of a spiking convolution processor. We call this the *pseudosimultaneity* between input and output visual flows. To illustrate this, let us look at Fig. 2, where a flow of events produced by Delbrück's motion retina [26] is filtered by an AER ConvChip with a vertical Gabor kernel. The retina is observing two persons walking, and retina pixels send an address event (their x, y coordinates) when they detect a change of light above a given threshold. As a result, the retina output continuous event flow represents the moving silhouettes of the two persons walking. There are no frames nor a global periodic reset. The flow of events is continuous and each pixel is autonomous. Dots in Fig. 2(a) represent a 2-D histogram of the retina event flow for a duration of about 40 ms, containing a total of about 980 events. Circles are the output events computed by a ConvChip during the same 40 ms. The ConvChip was programmed with an 11×11 vertical Gabor kernel. During these 40 ms the ConvChip produced about 260 events. As can be seen, the extracted vertical edges overlap very well with the corresponding input edges. No time delay is observed. Fig. 2(b) shows the x-axis projection versus time of the events in Fig. 2(a). Fig. 2(c)–(d) are the same as (a)–(b) but for the events collected during the first 4 ms only, and Fig. 2(e)–(f) for the first 1 ms. Again, extracted vertical edges seem synchronized with their corresponding input edges. The ConvChip produces an output event after collecting a number of significant space-time correlated input events. The delay of processing one single event is in the order of tens to hundreds of nanoseconds (as shown later in Section VI). Consequently, the delay between

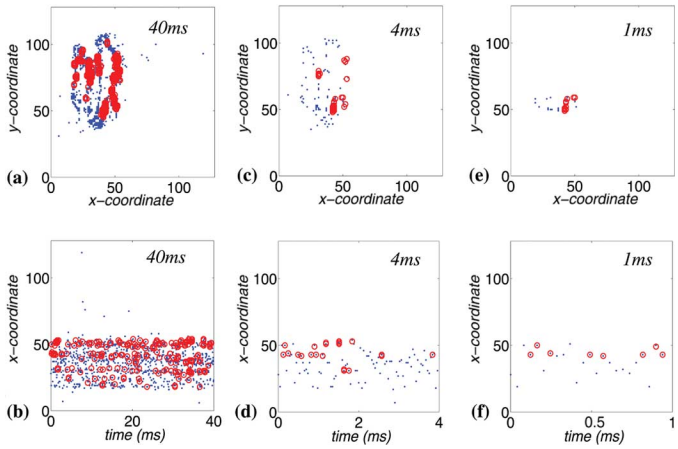

Fig. 2. Illustration of pseudosimultaneity when a ConvChip filters events produced by a motion sensitive retina. (a) ConvChip input (dots) and output (circles) events captured simultaneously during 40 ms, represented in the (x, y) coordinate plane. (b) Same events represented in the $(x, time)$ plane. (c)–(d) Represent the same than (a-b) but for a subset of events during the first 4 ms, and (e)–(f) for the first 1 ms.

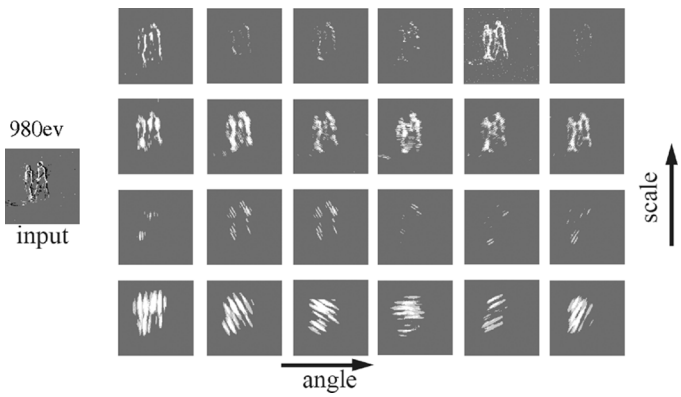

Fig. 3. Illustration of pseudosimultaneity for a 6×4 array of Gabor filters of different scales and orientations. Input and output events correspond to the same 40 ms time window.

input and output flow is mainly given by the event density of the input data, kernel stored, and ConvChip settings. Basically, this means that the ConvChip needs a given number of space-time correlated events to produce an output event. In this case *reality* moves slowly and we need between 4–40 ms to collect enough retina events for "seeing" silhouettes or parts of them. For faster moving *realities* (as discussed in Section VI-E) the retina would provide much faster event rates, and the timing characteristics of the ConvChip (e.g., its forgetting rate) need to be adapted to observe a much faster *reality*. In this case, the ConvChip also needs to detect a given number of space-time correlated input events to provide an output event representing a detected feature. We will discuss in more detail forgetting rate and time domain filtering considerations in Section IV.

Fig. 3 shows the situation where the same 40 ms retina event flow is sent in parallel to an array of 4×6 ConvChips each programmed with a Gabor kernel of different scale and orientation. All 24 convolution output flows shown were collected during the same 40 ms than the input flow. This pseudosimultaneity between input and output event flow of a spiking convolutional module contrasts strongly with the convolution operations on standard sequential frame-based video sensing and processing algorithms, where for computing each convolution one needs to have available the full input image.

Other spike coding approaches, such as rank-order coding [64], have been reported and exploited with great success [65]. However, rank-order coding is a frame-constraint scheme where each frame can be represented by a compact nontimed ordered list of spikes that can be processed very quickly. However, the complete sequence has to be processed to obtain one convolution. Although this processing can be very fast, it is not like the *pseudosimultaneity* property explained here.

The second interesting property of implementing Spiking Event Convolutions (or other operators, in general) is its modular scalability. Since event flows are asynchronous, each AER link between two convolutional modules is independent and needs no global system level synchronization. AER links in a generic multi-AER-module system can be point-to-point (one sender–one receiver), one-to-many (one sender–many receivers), many-to-one (many senders–one receiver), or many-to-many (many senders–many receivers). This has been handled in several ways in the literature, such as establishing special timing and handshaking characteristics of the AER links [66], [67], or by always using a point-to-point handshaking protocol but inserting AER splitters, mergers, and address remapping modules, either using synchronous FPGA components [68] or fully asynchronous self-timed schemes [69]. In any case, it is not difficult to assemble many convolutional modules into large size ConvNets of the type shown in Fig. 1. Furthermore, given the pseudosimultaneity of input-to-output Spiking Convolutional processing in individual modules, this pseudosimultaneity also applies to larger scale multilayer and multimodule systems.

The third interesting property of spike based hardware, in general, is that since processing is per-event, power consumption is, in principle, also per-event. Since events usually carry relevant information, power is consumed as relevant information is sensed, transmitted and processed. In our chip, however, this is not exactly true. The reason is that our ConvChip includes a forgetting mechanism which is operating always and consequently consumes a fixed amount of background event-independent power.

IV. Architecture of the Convolution Chip

The chip described in this paper is a fully digital convolution chip with programmable arbitrary-shape kernels. It receives input AER events, which represent visual information from a previous sensing or processing stage, and generates output AER events, which represent the result of the convolution operation. In general, input events can be asynchronous (if they are generated by an asynchronous AER sensor) or synchronized to some external clock (if they come, e.g., from some computer interface or other device with an internal clock). Our convolution chip includes a synchronous controller described and synthesized through VHDL. Consequently, we will need to link the external asynchronous input events with the internal controller clock. One option could have been using "clock stopping" [15] during absence of input events. However, as we will explain shortly, the synchronous controller includes a periodic forgetting mechanism which needs to be kept active during absence of input events. Consequently, we chose to use "synchronizers" to link the input asynchronous events with the internal controller clock [70].

The system level architecture of the chip is illustrated in Fig. 4, where the following blocks are shown: 1) array of 32×32 digital pixels; 2) static RAM that holds the stored

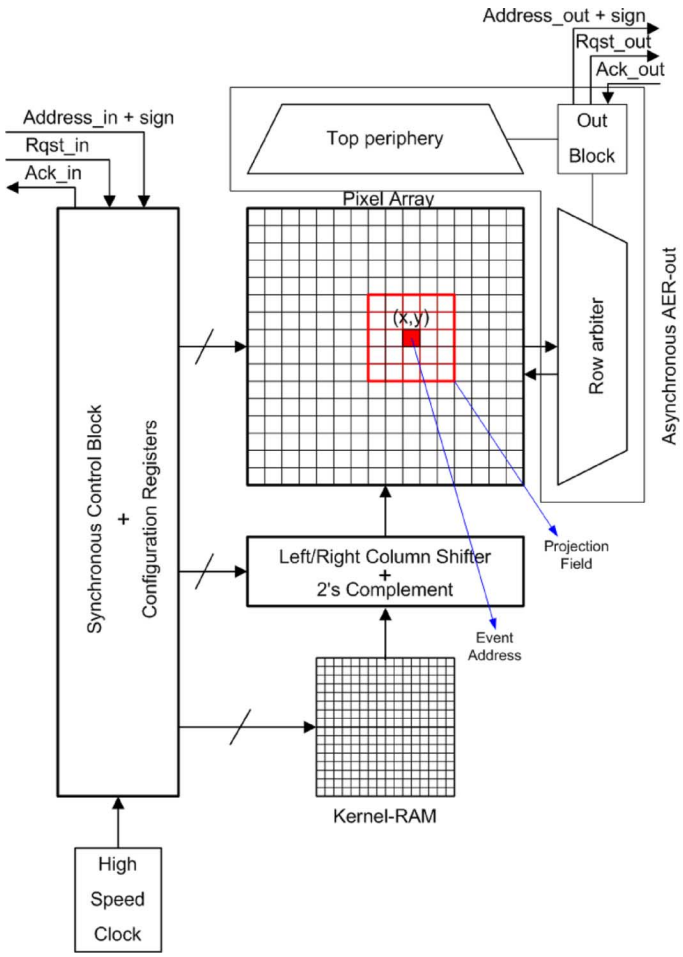

Fig. 4. Architecture of the convolution chip.

kernel in 2's complement representation; 3) synchronous controller, which performs the sequencing of all operations for each input event and the global forgetting mechanism; 4) high-speed clock generator, used by the synchronous controller (it can be programmed to use this internal clock or an external one); 5) configuration registers that store configuration parameters loaded serially at startup; 6) a 2's complement block that inverts the kernel data before being added to the pixels, if the input event is negative; 7) left/right column shifter, to properly align the stored kernel with the incoming event coordinates; and 8) AER-out, asynchronous circuitry for arbitrating and sending out the output address events generated by the pixels.

The operation of the chip is as follows: when the synchronous controller detects a falling edge in the input *Rqst_in* line, the event address (x, y) and sign at *Address_in* is latched and the asynchronous handshaking completed. Then the controller, using the available kernel size information, computes the limits of the projection field with three different possible results: 1) the projection field fits fully inside the array of pixels; 2) it can be partially inside the array; or 3) it can be completely outside the array. If it is outside the array, the controller discards the event and waits for the next one. However, in any of the other possible situations, the controller calculates the left/right shift between the RAM columns holding the kernel and the projection field columns in the pixel array. After this, it enables the addition, row after row, of the kernel values onto the pixels. Hence, after receiving an input event, the pixels inside the projection field change their state. If any of them reaches the programmed threshold it resets itself and generates an output event that will be handled by the asynchronous AER-out block and sent off chip with its corresponding handshaking signals. Parallel to this per-event processing, there is a global forgetting mechanism common for all the pixels.

The asynchronous AER-out block follows the row-parallel event read-out technique [71]. Events are arbitrated by rows (for the same row all request signals are wired-or). Once the row arbiter answers, all the events generated in this row are latched on the top periphery, freeing the row arbiter. This way, the row-arbiter can acknowledge the request of another row, while the events of the previous row are sent out in a burst. Note that individual pixels will generate their requests in synchrony with the controller clock. Consequently, the AER event read out could have been implemented by some synchronous mean as well (or even using the same synchronous controller). However, it is more efficient to use an independent asynchronous mechanism, because the next stage (chip) AER receiver works asynchronously with respect to the controller's clock, and eventually might need to slow down the event communication between both chips. In this case, it is more efficient to keep the synchronous controller working on the incoming events and maintain its operation completely decoupled from the output event read out process.

The size of the array is 32×32 pixels, but the input address space it can "see" is larger (128×128). This allows to build arrays of convolution chips to process larger pixel arrays, programming each one of them to see a part of the address space by setting some configuration registers [41], [42]. The size of the RAM is 32×32 words of 6 bits in 2's complement representation.

In general, since convolution kernels can have positive or negative values, output events generated by a convolution chip can also be either positive or negative. In a multilayer system convolution operations can be cascaded, which implies that a generic convolution chip must be able to handle signed input events, and produce signed output events. For this reason, the chip described in this paper includes a sign bit both for the input and output address events, and also for the values stored in the kernel RAM (in 2's complement representation). The pixels must be able to compute signed addition and produce positive and negative events. When processing a negative input event, the controller enables the 2's complement block to invert the kernel values before being added to the pixels. Another possible option, as done in biology, could have been duplicating channels and neurons for positive (ON) and negative (OFF) signals. However, for our digital hardware it is much more effective to add the sign bit processing circuitry and the 2's complement block.

The forgetting mechanism is also handled by the synchronous controller. The aim of this mechanism is that the absolute state values stored in the pixels are decremented at a programmable rate, so that they can "forget" their previous state after some controlled time. This functionality is implemented by a 20-bit counter in the controller which generates a periodic forgetting pulse for all the pixels every time it reaches the programmed limit. The forgetting pulse period is set to $N_f T_{\text{clk}}$, where N_f is the programmed value and T_{clk} is the clock period. The maximum possible value is $(2^{20} - 1) * T_{\text{clk}} = 1048575 * T_{\text{clk}}$. Each forgetting pulse will decrement by "1" the state of all the pixels with positive state, and increment by "1" those with negative state. Consequently, the chip implements a (programmable)

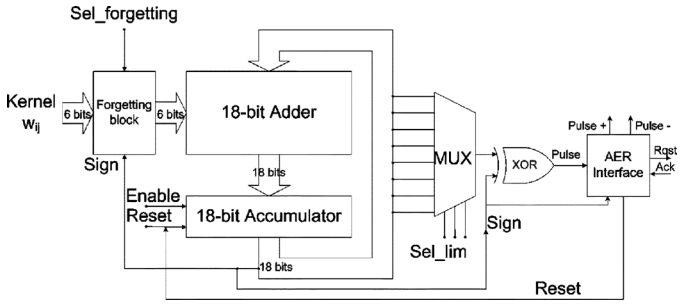

Fig. 5. Block diagram of the convolution pixel.

constant-rate (or linear) forgetting mechanism, as opposed to more biological models where the forgetting mechanism is implemented via an RC discharge exponential type mechanism. This latter mechanism would adapt automatically to the rate of input events by settling to a DC level dependent on such rate. In our hardware an exponential mechanism would yield a very sophisticated and prohibitively large pixel. However, a linear rate forgetting mechanism just needs minor pixel modifications. The only drawback is that the time constant associated to the forgetting rate has to be set globally and common for all pixels, depending on the time constants one wants to capture from the sensed reality.

V. The Digital Convolution Pixel

A block diagram of the convolution pixel is shown in Fig. 5. The convolution operation is performed at the pixel level by the integration of input events weighted by the kernel values. The main part of the pixel is an 18-bit full adder with an 18-bit accumulator. In this first digital prototype a conservative over sized accumulator length was implemented intentionally to allow for maximum possible precision. The criteria was to allow accumulation of a scaled-up 64×64 kernel with all weights at maximum 6-bit value, while allowing the least significant bit of the kernel to contribute as well to the accumulation. This results in a dynamic range of 18 bits. Pixel area could have been reduced by sharing the adders column-wise and implement in the pixel only the accumulators. However, the main delay source in chip's event processing is transferring data from the periphery to the pixels. In the present version we just have one such delay per kernel line. Sharing the adders at the periphery would imply having two such delays, plus the need of 18 lines per column for accumulator state transfer, instead of the present 6 lines for transferring the weights. Alternatively, it could be possible to pipeline the adder input and output transfers of consecutive rows to reduce transfer delays, but at the cost of doubling the transfer lines per column from 18 to 36.

The accumulator stores the state of the pixel, represented by a 2's complement signed number. For each input event, the *Enable* signal (which is common for all the pixels in the same row) is activated, the accumulator is updated with the corresponding kernel weight w_{ij}, and if it reaches a programmed threshold it fires an event and the accumulator is reset. This threshold is controlled with a 3-bit parameter (*Sel_lim*) that selects through a multiplexer one of the bits of the accumulator. This selected bit is compared continuously with the sign bit (which is the most significant bit). This way, for positive words ($\mathrm{msb} =$ "0") the comparator will fire when the selected bit becomes "1," while for negative words ($\mathrm{msb} =$ "1") it will fire when the selected bit becomes "0." Note that not any of the 18 accumulator bits can be selected for comparison, but only 8 (since *Sel_lim* is only 3 bit). We have chosen bits "0" to "5," "7," and "16." This way we have more flexibility for lower (more critical) thresholds which are needed for fast processing, and keep only the possibility of selecting bits "7" and "16" when higher precisions might be required in cases where speed is not a concern. The 8 possible threshold settings are detailed in Table I. Note that since we are using a simple XOR gate (instead of a comparator) to detect the threshold, the maximum possible kernel weight w_{ij} has to be chosen properly depending on the bit selected for comparison.

TABLE I
Programmable Limits for the Accumulator

Sel	Maximum	Minimum
000	65536	−65537
101	128	−129
011	32	−33
010	16	−17
110	8	−9
111	4	−5
100	2	−3
001	1	−2

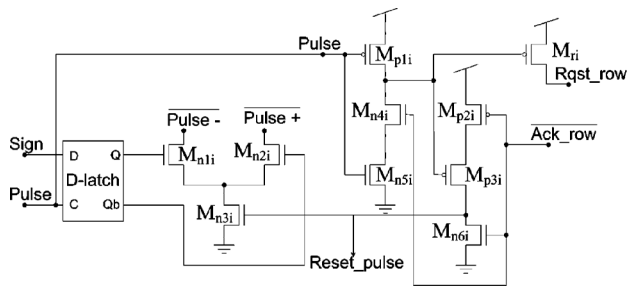

Fig. 6. Schematic of the interface block inside the pixel.

The periodic forgetting pulses generated by the controller activate pixel signals *Enable* and *Sel_forgetting* for all pixels in the array, thus producing another add/accumulate operation. However, instead of adding the kernel value w_{ij} coming from the RAM, the *Forgetting block* selects a different input for the adder. This block includes a set of switches controlled by the *Sel_forgetting* pulse, so that during normal event processing behavior the kernel value is selected, but for a global forgetting pulse the switches select either the value 1 (if the accumulator sign is negative) or -1 (if it is positive).

The timing of the communication between row pixels and row arbiter is very critical for optimum operation. Unfortunately, this timing relies heavily on parasitics and changes significantly from chip to chip, and even between different rows of the same chip. To overcome this, we included a row-wise coarse calibration for the row pull-down transistors. Each row's pull-down transistor size can be independently adjusted to nW/L with $n = \{1, 2, 3\}$. This row-wise calibration, together with the possibility of globally tuning the pull-down transistor analog gate voltage allows to optimize the timing and compensate for process and in-chip variations.

The details of the AER interface block of Fig. 5 are shown in Fig. 6.

VI. Experimental Results

A 4.3×5.4 mm^2 prototype chip has been fabricated in the AMS 0.35 μm CMOS process. A die photograph is shown in Fig. 7, and chip specifications are summarized in Table II. The

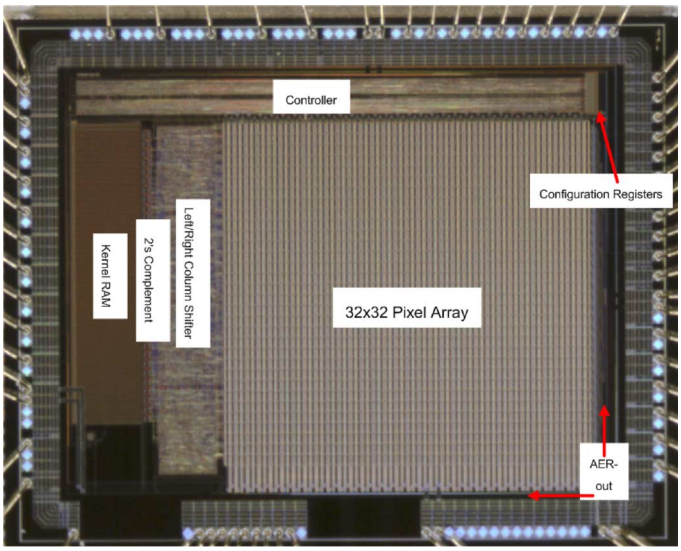

Fig. 7. Photograph of the fabricated 32×32 pixel fully digital convolution chip.

TABLE II
CHIP SPECIFICATIONS

Technology	4M 2P $0.35\mu m$ CMOS
Pixel Size	$95.6 \times 101.3\mu m^2$
Chip Size	$4.3 \times 5.4mm^2$
Pixel Array	32×32
Pixel Size	$95.6 \times 101.3\mu m^2$
Pixel Computing Resolution	18 bit
Weights Resolution	6 bit
Signed Computations	yes
Max Clock Frequency	120MHz
Input Event Throughput	1.77-20 Meps (depending on kernel lines)
Peak Output Event Rate	45 Meps ($T_{burst} = 22ns$)
Min Latency in-to-out	$155ns$
Power consumption	200mW max

largest block is the 32×32 array of pixels, with an approximate area of 3.0×3.2 mm^2. The synchronous controller consumes around 4500×300 μm^2, the static kernel-RAM of 32×32 6-bit words 600×2700 μm^2, and the left/right column shifter 600×3100 μm^2. The rest of the circuits, like the AER-arbiters, 2's complement and clock generator, consume much less area.

The pixel has an area of 95.6×101.3 μm^2. Most of this area is consumed by the 18-bit adder and accumulator. The rest of the circuits are: the forgetting block, the multiplexer, the comparator and the AER interface. We also include a capacitor between supply and ground inside each pixel to filter power glitches. This 240 *fF* capacitance is placed under the supply and ground stripes, to avoid extra area consumption. The routing of lines inside the pixel is a highly critical issue, with critical parasitic capacitance couplings, as some lines are shared by all the pixels in the same row or column, and can be as long as 3 *mm*. Some of these lines are used for configuration parameters, which are loaded at startup and remain silent throughout normal operation. These "static" lines were laid out between fast dynamic lines, shielding couplings among dynamic lines.

Although the chip resolution is 32×32 pixels, it can address an input space of 128×128. To specify the coordinates of the chip within the whole address space, a set of configuration registers is loaded at startup. The different parameters and biases that can be controlled by the user are described in Table III. The top eight are digital words written serially using a shift register, while the two bottom parameters represent analog voltage biases. Other digital words loaded at startup are the kernel weights to be written in the kernel RAM.

TABLE III
PROGRAMMABLE BIASES AND PARAMETERS

Parameter	Bits/Lines	Meaning
$(x, y)_{min}$	14	coordinate of the upper left pixel within the 128×128 input address space
$(x, y)_{max}$	14	coordinate of the lower right pixel within the 128×128 input address space
(p, q)	10	kernel dimensions inside the RAM
N_f	20	clock cycles specified between two global forgetting pulses
sel_{clk}	1	selection of either internal or external clock
sel_{acc}	3	selection of the accumulator limit
sel_{pd}	64	configuration of W/L for row communication pull-downs
sel_{syn}	2	selection of synchronizer
V_{bias_clk}	1	voltage bias that controls the internal clock frequency
V_{bias_arb}	6	voltage biases that control the pull-downs and pull-ups in the arbiter circuits

The frequency of the internal clock can be adjusted through parameter $V_{\mathrm{bias\_clk}}$. It could be set up to 120 MHz before observing some sparse erroneous events appearing at the output. If these spurious events can be tolerated, clock frequency could be further increased until 200 MHz, beyond which the convolution operation degrades completely. The main delay limiting the clock speed comes from copying the kernels from the peripheral kernel-RAM onto the pixel lines, through long wires. In our experiments we set the clock frequency at 120 MHz.

The power consumption of the chip depends both on the input throughput and the kernel size. For instance, when setting an event emitter to provide input events at a rate of 5 Meps the power consumption varies between 66 mW and 198 mW, for the smallest possible kernel (1×1) and the largest (32×32), respectively. Next we show a series of experiments to characterize chip performance. For these experiments we used a dedicated AER infrastructure based on FPGAs [72], [73]. We have used two boards: 1) an AER data player, which stores sequences of AER events and reproduces them as physical AER streams; and 2) an AER data logger, which collects AER streams and stores them into memory for later analysis.

A. Event Timings Characterizations

Event timing characterizations are crucial to understanding and optimizing the main sources of delays within the chip. We distinguish between output events peak rates which are due to the asynchronous AER-out circuitry, input event throughput which is due to the delays of the synchronous operations, and input to output latencies where both synchronous and asynchronous circuits play a role. We will show how to decouple the measurements of the synchronous and asynchronous delays by changing clock frequency. Careful characterization of each component yields a minimum input to output latency of 155 ns. Next we explain how to characterize this delay.

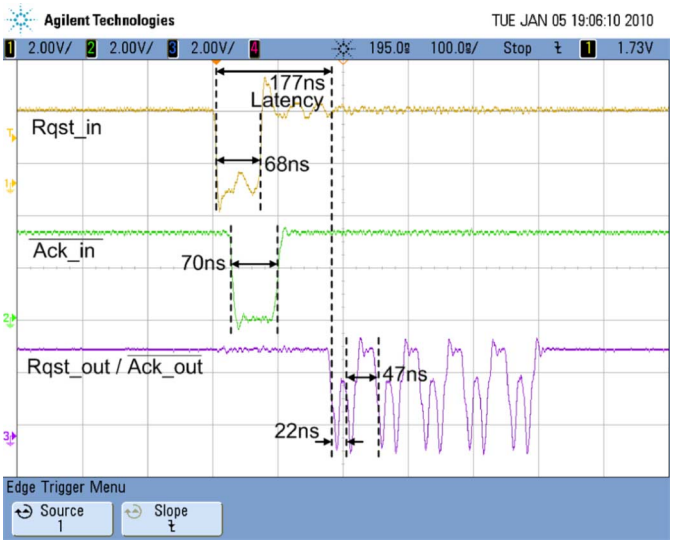

Fig. 8. Measured input and output events *Rqst* and *Ack* signals when maximum clock frequency is set to $f_{clock} = 120$ MHz, with shorted *Rqst_out* and *Ack_out*. The oscilloscope used is Agilent DSO7054A, with a bandwidth of 500 MHz and a sample rate of 4 GSa/s.

1) Output Events Peak Rates: The chip can send out events at a maximum output rate of 45×10^6 eps (events per second), measured shorting *Rqst_out* and *Ack_out* and for events generated by pixels in the same row (in burst mode). This corresponds to an event cycle time of $T_{\text{burst}} = 22$ ns. However, depending on the position of the pixels within the array the event cycle time changes significantly. Pixels closer to the arbiter will produce shorter propagation delays. However, this influence can be minimized by setting carefully the values of the bias voltages and calibration switches for the pull-ups and pull-downs on the periphery. This way, we could adjust this delay between 20 ns and 24 ns for all the pixels in the array. Fig. 8 shows the measured signals *Rqst_in*, $\overline{Ack\_in}$, *Rqst_out* (shorted with $\overline{Ack\_out}$) when the following 5-line kernel is loaded:

$$\begin{bmatrix} 1 & 0 & 1 \\ 1 & 0 & 1 \\ 1 & 0 & 1 \\ 1 & 0 & 1 \\ 1 & 0 & 1 \end{bmatrix}. \qquad (1)$$

Pixel threshold was set to fire an output event when receiving one single input event. Consequently, this kernel activates 10 different pixels (belonging to 5 different rows) for the same input event. The output events in Fig. 8 show a delay of $T_{\text{burst}} = 22$ ns when both events belong to the same row (once the row is acknowledged by the arbiter, all the events produced in the same row are sent out in burst mode) and a delay of $T_{\text{nonburst}} = 47$ ns for events from different rows. Consequently, the difference $47 \text{ ns} - 22 \text{ ns} = 25$ ns corresponds to the row arbitration delay T_{arb}.

2) Input Events Throughput: The input event throughput depends on the kernel size and the internal clock frequency. The synchronous controller needs [41], [42] $n_{\text{clk}} = 4 + 2n_k$ clock cycles to process a single input event, where n_k is the number of rows of the programmed kernel (up to 32). This way, for a clock frequency of 120 MHz ($T_{\text{clk}} = 8.33$ ns), the maximum possible input event rate is 20×10^6 eps, which corresponds to an input event cycle time of 50 ns when the kernel has only one row. For a full kernel of 32 rows, the input event maximum rate would be 1.77×10^6 eps (566 ns input event cycle time).

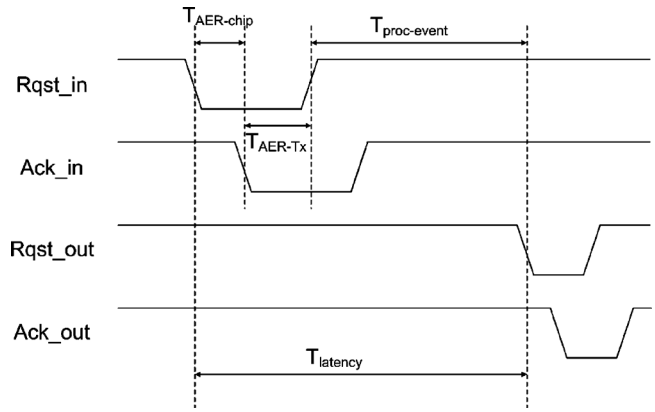

Fig. 9. Time delay between input and output events.

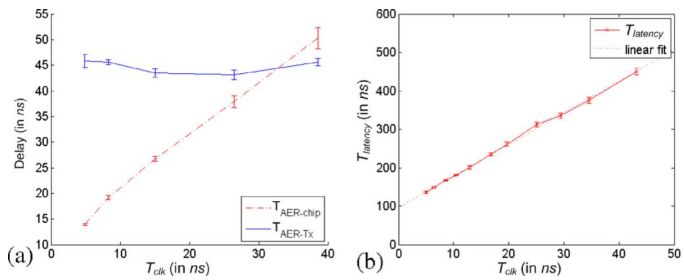

Fig. 10. (a) Measured values for $T_{\text{AER-chip}}$ and $T_{\text{AER-Tx}}$ for different settings of T_{clk}. (b) Measured values of T_{latency} for different settings of T_{clk}; each measurement was repeated 5000 times and error bars indicate measured spread; dotted line represents linear fit ($y = 8.2 \times x + 97$).

3) Input to Output Latency: In Fig. 9, the latency between input and output events is represented by T_{latency}, which can be expressed as $T_{\text{latency}} = T_{\text{AER-chip}} + T_{\text{AER-Tx}} + T_{\text{proc-event}}$. Delay $T_{\text{AER-chip}}$ is introduced by the convolution chip when *Rqst_in* is activated by the emitter. Delay $T_{\text{AER-Tx}}$ is introduced by the emitter when the convolution chip acknowledges. Delay $T_{\text{proc-event}}$ is the time used by the chip to process the input event and generate an output event (considering a situation where one input event produces one output event). Delay $T_{\text{proc-event}}$ can be expressed as $T_{\text{proc-event}} = T_{\text{syn}} + T_{\text{asyn}}$, where T_{syn} represents the time the synchronous controller needs for adding kernel weights onto the pixels, and T_{asyn} represents the time for the asynchronous arbitration and generation of the output event.

Fig. 10(a) shows the measured $T_{\text{AER-chip}}$ and $T_{\text{AER-Tx}}$ for different values of T_{clk}. Vertical bars indicate deviations over 5000 measurements. Delay $T_{\text{AER-Tx}}$ is approximately constant, as it depends only on the emitter. The delay measured for the emitter board used in our tests is about $T_{\text{AER-Tx}} = 44$ ns. However, $T_{\text{AER-chip}}$ depends almost linearly on T_{clk}, as can be seen in the figure, reaching 14 ns for $T_{\text{clk}} = 5$ ns and with an estimated residual value of 10 ns for $T_{\text{clk}} = 0$ (probably due to internal lines and pads delays). Fig. 10(b) shows the measured T_{latency} versus T_{clk}. The linear fit of this data reveals that at $T_{\text{clk}} = 0$ we would have a latency of 97 ns, which corresponds to $T_{\text{proc-event}} = T_{\text{asyn}}$, as T_{syn} would be 0. Extrapolating also the values of $T_{\text{AER-chip}}$ and $T_{\text{AER-Tx}}$ we can estimate the value of

$$\begin{aligned} T_{\text{asyn}} &= T_{\text{latency}}(T_{\text{clk}}=0) - T_{\text{AER-Tx}} - T_{\text{AER-chip}}(T_{\text{clk}}=0) \\ &= 97 \text{ ns} - 44 \text{ ns} - 10 \text{ ns} = 43 \text{ ns} \end{aligned}$$

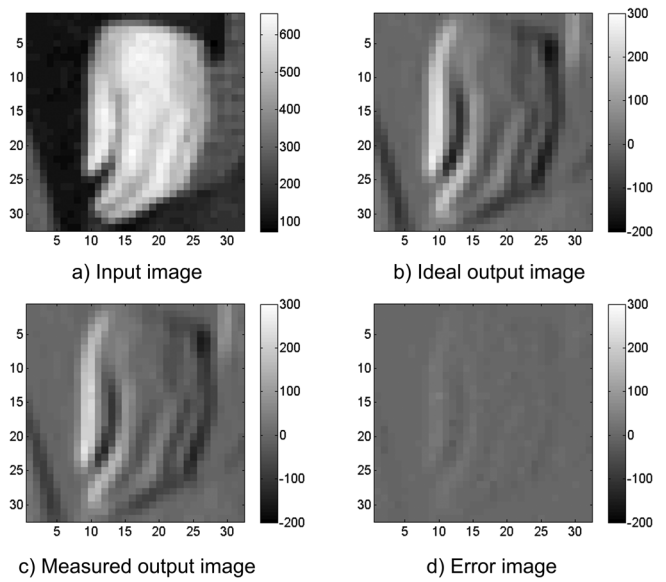

Fig. 11. Experimentally obtained convolution processing results with a kernel for vertical edge extraction. (a) Input image. (b) Ideal output image. (c) Measured output image. (d) Error image.

which is coherent with the 47 ns delay measured between two output events from different rows. For $f_{\mathrm{clk}} = 120$ MHz, the measured T_{latency} is 177 ns.

When cascading convolution chips, the delay $T_{\mathrm{AER-Tx}}$ of the emitter board should be replaced by either $T_{\mathrm{burst}} = 22$ ns or $T_{\mathrm{nonburst}} = 47$ ns. Consequently, the true minimum latency when cascading these chips would be given by

$$\begin{aligned} T_{\mathrm{latency}} &= 177\ \mathrm{ns} - T_{\mathrm{AER-Tx}} + T_{\mathrm{burst}} \\ &= 177\ \mathrm{ns} - 44\ \mathrm{ns} + 22\ \mathrm{ns} = 155\ \mathrm{ns}. \end{aligned}$$

Note that this latency is independent of the number of lines of the kernel because, as shown in Fig. 8, it is the delay between the input event *Rqst_in* and the first output event *Rqst_out* produced by the first kernel row. Consequently, this characterizes minimum event latency as 155 ns.

B. Convolution Processing of 32 × 32 Pixel Static Images

To illustrate convolution processing, a 32×32 pixel input image was selected from a real photograph (shown in Fig. 11(a)) to perform an edge-orientation extraction with an 11×7 difference of gaussians kernel described by:

$$\begin{aligned} F(p,q) &= \frac{1}{2\pi} H(p)\, V(q) \\ H(p) &= \frac{1}{\sigma_h} e^{-(1/2)(p/\sigma_h)^2} \\ V(q) &= \frac{1}{\sigma_v}\left[e^{-(1/2)(q/\sigma_v+1/2)^2} - e^{-(1/2)(q/\sigma_v-1/2)^2}\right] \end{aligned} \quad (2)$$

where σ_h and σ_v are the horizontal and vertical spatial width parameters of the Gaussian lobes. The image was rate coded into AER events with a maximum frequency of 660 Hz. The frequencies associated to the intensity levels are indicated in the vertical bars on the right side of each image in Fig. 11. The mathematical computation of the convolution operation was performed with MATLAB, obtaining the image in Fig. 11(b). Fig. 11(c) shows the output from the convolution chip, by mapping the signed output event frequency of each pixel into a gray level. A

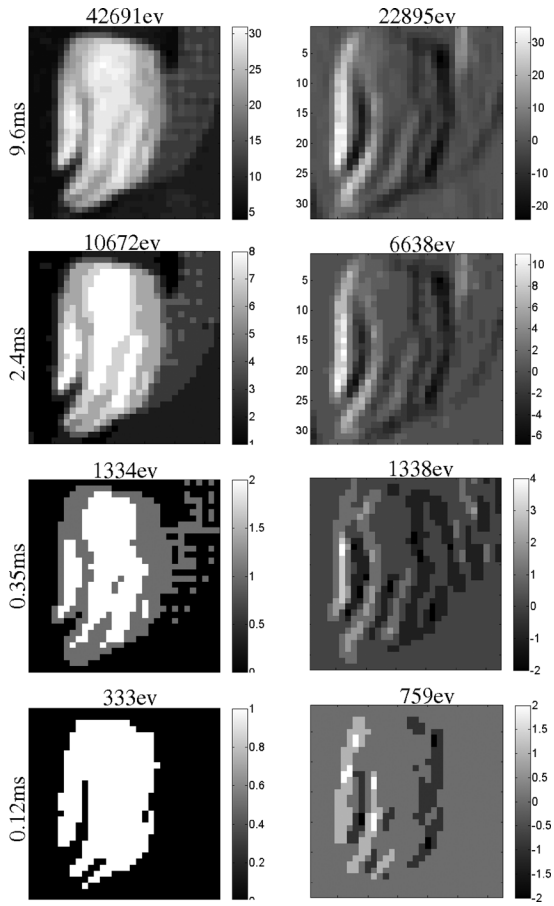

Fig. 12. Speed characterizations of convolution processing. Top row: input image pixels coded from 0 to 32 events with an event burst duration of 9.6 ms, convolution output is captured synchronously with the input during the same 9.6 ms. Second row: same as above, but capturing input and output events synchronously during the first 2.4 ms only. Third row: same, but during the first 0.35 ms. Fourth row: same, but during the first 0.12 ms.

negative pixel frequency means that the sign bit of the output events for this pixel was negative. Fig. 11(d) represents the error image as the difference between the ideal and measured output. Maximum error is 42.3 Hz and error standard deviation is 2.61 Hz. This error is produced by the quantization effect produced every time a pixel accumulator is reset to zero when reaching threshold. If pixels are not reset to zero but to the excess with respect to the threshold, then no errors would be observed. However, this would complicate the pixel hardware and increase its area consumption.

The measurements in Fig. 11 were performed by stimulating the ConvChip with a continuous flow of events, where each pixel had a fixed event frequency proportional to the pixel gray level within the image. The ConvChip output event flow was captured during a sufficiently long time and, for each pixel, event frequency was extracted. The vertical side bars in Fig. 11 indicate pixel frequency in Hz. In order to measure the time delay it takes to compute the 2-D convolution of this image, we proceeded differently. Instead of providing a continuous event flow, we generated a single burst of events representing the image. Each pixel was assigned a number of events between 0 and 32, depending on its gray level. The top left subimage in Fig. 12 shows, coded in gray scale, the number of events assigned to each pixel as indicated by its side bar. The total events of the burst is 42 691. This burst of events was then fed

to the ConvChip setting the AER data-player [72], [73] at the maximum speed it could provide (about 10 ns/event). This way, since the ConvChip is processing a large kernel (11 lines), the ConvChip will slow down the event communication to the maximum input event throughput it can handle for this kernel size (which is about 220 ns/event). As a result, we measured a burst duration 9.6 ms. The top right subimage shows the output event burst provided by the ConvChip during the same 9.6 ms, coding the number of events collected per pixel and its sign bit in gray scale as shown in its vertical side bar. The figures in the second row of Fig. 12 are obtained by collecting the events during the first 2.4 ms of the burst. Input pixels have now 8 events at the most, while output pixels have about 3 times less events than before. The third row is for the case events are collected during the first 0.35 ms only. In this case, input pixels have 2 events at the most, while output pixels range from 2 negative events up to 4 positive events. The last row corresponds to collecting events during the first 0.12 ms. Input pixels have one event at the most, while output pixels have up to two events positive or negative. As can be seen, there is a trade-off between resolution and response time, and that even for low resolutions the output still conveys the most representative pixels.

Note that AER visual flow representing luminance information may require a large number of events. Therefore, using plain luminance AER sensors in a generic AER processing system is not the most efficient way. In practice, it is much more efficient to use sensors with some focal plane preprocessing such as spatial and/or temporal contrast [1]–[6], [13], [14], [21], [22], [24]–[28].

C. Convolution Processing of Larger Static Images

Although the chip pixel array size is only 32×32, the input address space it can see is 128×128. This allows to assemble a 2-D array of convolution chips for processing pixel arrays which are multiples of 32×32 [41], [42], [68]. Each convolution chip is programmed with parameters ($x_{\min}$, $x_{\max}$, $y_{\min}$, $y_{\max}$) to indicate where the 32×32 pixel array is with respect to the total 128×128 input space it can see. Using AER splitter and merger boards [73], it is possible to build an array of 4×4 convolution chips to process arrays of 128×128 pixels. To process larger input spaces we need to use also AER mapper blocks [73] to map conveniently the larger pixel space into the 128×128 pixels each convolution chip can see. In Fig. 13 a large 256×256 static image was processed. The original 256×256 pixel array was split into 8×8 smaller subarrays. Then, each subarray was transformed into a sequence of AER events, processed by the convolution chip with the same kernel than for Fig. 11 and the output events recorded. The 64 recorded AER event sequences were then assembled offline into 64 subimages and remapped to obtain the 256×256 output shown in Fig. 13(c). If this mathematical convolution is computed directly with MATLAB, the result obtained is shown in Fig. 13(b), while the error image that represents the difference between the ideal frequencies and the measured ones is shown in Fig. 13(d). Maximum error is 59 Hz while standard deviation is 2.7 Hz.

D. Convolutions for Moving Stimuli

Although the experiments shown previously correspond to static images, the aim of the chip is to compute in real time convolutions of dynamic stimuli coming from an AER retina. To illustrate this, a sequence of events was captured with the

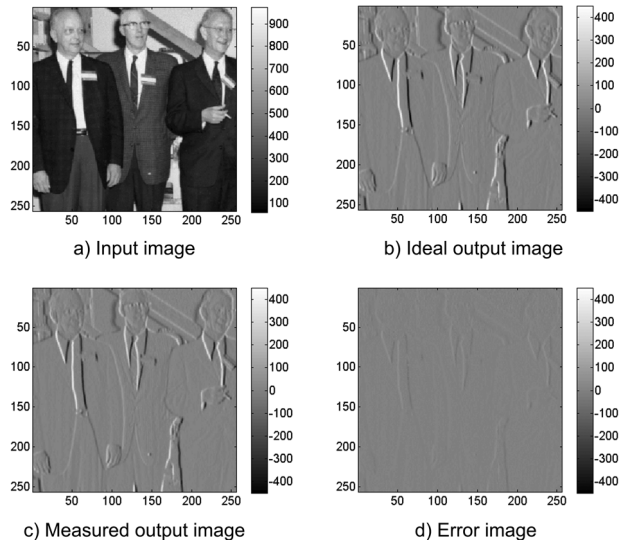

Fig. 13. Experimentally obtained convolution processing results with a kernel for vertical edge extraction of a 256×256 input image. (a) Input image. (b) Ideal output image. (c) Measured output image. (d) Error image.

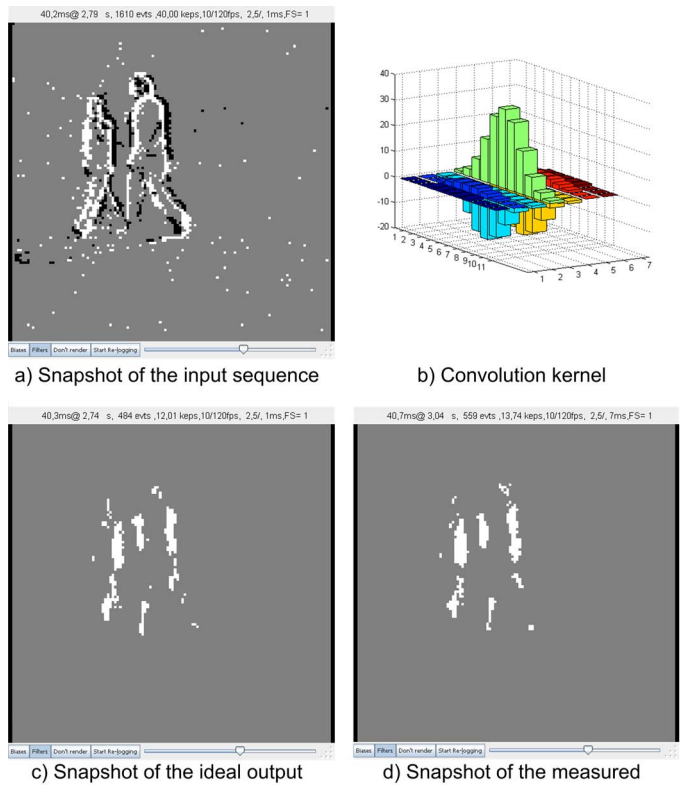

Fig. 14. Experimentally obtained convolution processing results with a Gabor kernel for vertical edge extraction of a 128×128 input sequence. (a) Snapshot of the input sequence. (b) Convolution kernel. (c) Snapshot of the ideal output sequence. (d) Snapshot of the measured output sequence.

128×128 temporal contrast vision AER retina, developed by Lichtsteiner and Delbrück [26], showing the moving contours of two persons walking. A 40 ms histogram of this sequence is shown in Fig. 14(a), which uses only 1810 events from the retina. As the retina address space is 128×128, this requires an array of 4×4 convolution chips. Programming the 7×11 Gabor kernel in Fig. 14(b) for vertical edge detection, we obtained the corresponding output sequence. A histogram of this

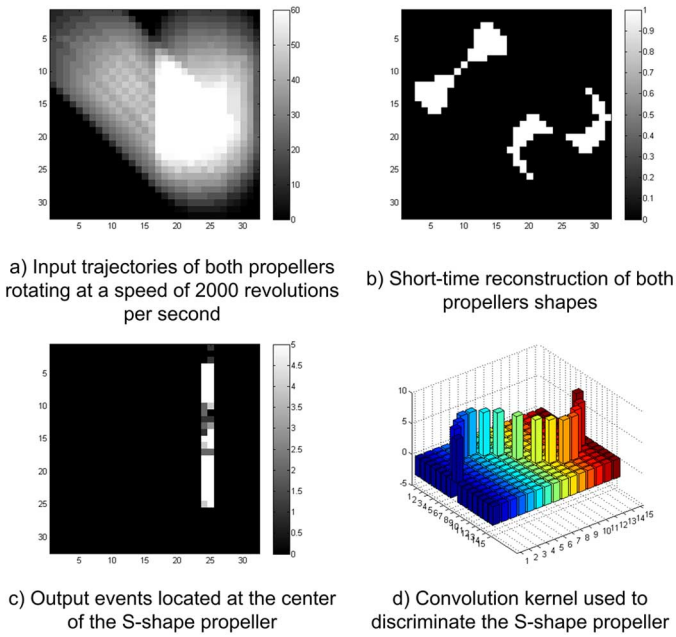

Fig. 15. Real-time discrimination of simultaneous rotating propellers at 2000 revolutions per second. (a) Input trajectories of both propellers rotating at a speed of 2000 revolutions per second. (b) Short-time reconstruction of both propellers shapes. (c) Output events located at the center of the S-shape propeller. (d) Convolution kernel used to discriminate the S-shape propeller.

output sequence for the same 40 ms of the input sequence is shown in Fig. 14(d).

To compare with the theoretical response, we fed the same retina stimulus to an AER behavioral simulator [74] performing the same convolution kernel. The AER output stream produced by this simulator was virtually identical to the one obtained experimentally by the chip. Fig. 14(c) shows the events produced by the behavioral simulator for the same 40 ms of the input in Fig. 14(a).

E. Discrimination of Rotating Propellers

To demonstrate the high-speed processing capabilities of the convolution chip, a very interesting experiment is the discrimination between high-speed rotating propellers [41]. We create artificially a sequence of events corresponding to two rotating propellers with different shapes, following the mathematical method explained elsewhere [41]. One of the propellers is rectilinear, and the other one has an S-like shape, as is illustrated on the right capture in Fig. 15. Both have a diameter of 16 pixels. The propellers will be rotating at a high speed and moving slowly across the screen. A human observer would only see two solid circles without being able to discriminate between them.

In this experiment, the artificially generated sequence of input events is loaded in an AER data player board [73], which sends the events to the convolution chip with the correct timing. After programming the specific kernel, the input events are sent to the chip and the chip output events are recorded by an AER data logger board [73]. This way, both sequences can be carefully analyzed in a computer. The aim of the experiment is to track the center of the S-shaped propeller by programming the 23×23 kernel shown in Fig. 15(d). This kernel is designed to produce positive output events in the center of the propeller when it is in horizontal position, and the large neighborhood with negative weights prevents from positive events being produced outside of the center of the propeller. Fig. 15(a)–(c) show the results of the experiment, where the two propellers rotate at 2000 revolutions per second. Fig. 15(a) shows the complete trajectories of both propellers moving across the screen and intersecting at a given point. This corresponds to a 50 ms capture, while the snapshot in Fig. 15(b) corresponds to a 50 μs capture. Average input event rate is 1.69 Meps (one revolution of the two propellers generates about 850 events). Fig. 15(c) shows how the output of the convolution chip follows the center of the S-shaped propeller as it moves, using the 23×23 kernel represented in Fig. 15(d). As expected, no output is produced for the center of the rectilinear propeller. The measured output event rate is 9.5 keps. Since the kernel has 23 lines, processing one event requires 50 clock cycles (see Section VI-A2), or 417 ns at 120 MHz clock frequency. Processing the 850 events of one revolution thus needs at least 355 μs, which results in a theoretical maximum propellers rotating speed of 2821 rps (revolutions per second). To find out the real maximum rotating speed the chip is able to handle for the two propellers, we proceeded as follows. We set the rotation speed in the AER data player above this theoretical limit. This way, the ConvChip slows down the event throughput to the limit it can handle. Measuring the event throughput under these circumstances reveals the maximum rotating speed, which we measured as 2688 rps.

In order to be able to work with higher rotating speeds, a pair of smaller propellers of 10 pixel diameter each (which generate about 325 events per revolution) were used, for which we need a smaller kernel of only 15 lines. At this rotating speed the input event throughput is 1.62 Meps, while output event rate is 19.6 keps. For a 15 line kernel, each event needs 283 ns, yielding a maximum possible event throughput of 3.53 Meps, which corresponds to a theoretical maximum rotation speed of 10860 rps. By setting the AER data player to provide rotations at a higher speed, the convolution chip slowed them down to 9433 rps.

These experiments reveal the potential of frame-free event-driven (vision) representation sensory and processing systems for very high speed object recognition. Note that for recognizing 10 krps propellers in a frame-constraint representation system would require to sense and process images at least at 100 k frames per second.

F. Recognition Latency Experiments

In order to show the short latency between input and output event flows, we did the following experiment. As input, we used a sequence of events recorded with a temporal contrast retina [26] when a circle of flashing light-emitting diodes (LEDs) is turned on and off every 40 ms. This experiment was done in [68] with the analog convolution chip [41], and now we can show an important improvement for the digital one. The convolution chip was programmed with a circular kernel to detect the center of the input circle of LEDs. Both input and output events are shown in Fig. 16(a) in a 3-D representation. Note that only the ON or OFF transients generate events. Input events are represented with dots and the output ones with circles. The recorded stimulus from the retina can be played back at different accelerated speeds with the AER data player. Each subfigure in Fig. 17 shows a y versus $time$ 2-D projection of a single ON or OFF transient. The left column graphs correspond to results obtained with the previous analog chip, while the right column graphs correspond to the present digital convolution chip. Each row in Fig. 17 corresponds to playing back the retina recorded data at different acceleration rates. The top row is for real time. Each ON (or OFF) retina transient (dots) lasts for about 5 ms

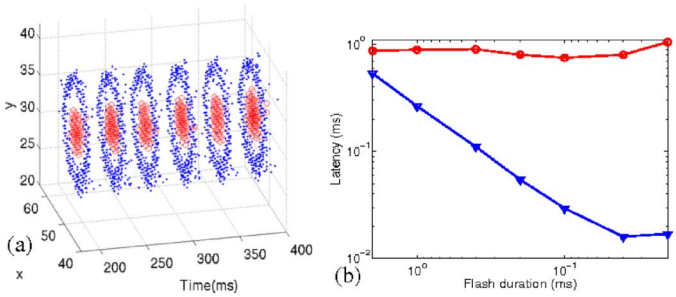

Fig. 16. (a) 3-D representation in $(x, y, time)$ space of the 40 ms period ON and OFF retina transients (dots) and convolution chip outputs (circle) when exposed to a flashing circle of LEDs. (b) Measured latency between input and output events for different values of the LEDs flash duration; circles correspond to the analog chip, and triangles to the digital chip.

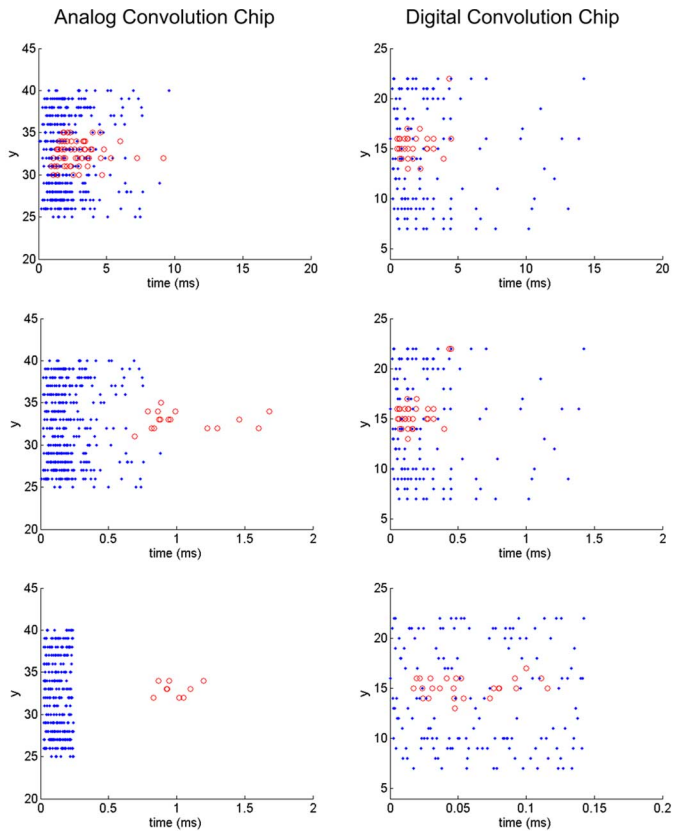

Fig. 17. Results for processing the circle of LEDs and discriminate the center, both with the analog convolution chip and the digital one. Dots represent the events generated by the retina, and circles represent the events generated by the convolution chip, being the y-axis the y-address of the events. Each row shows the results for different durations of the input burst.

and has about 400 events distributed in a circle in the x/y plane [see Fig. 16(a)]. The convolution chip output events (circles), which appear in the center of the circle, are practically simultaneous to the input events, for both the analog and the digital convolution chips. The second row in Fig. 17 corresponds to playing back the recorded retina events at a speed 10 times faster than real time. We can see that for the analog chip the output events appear slowed down, while for the digital chip the graph is virtually identical to the real time one. In the third row the recordings have been accelerated a factor of 25 with respect to real time. As can be seen, in the analog chip there is a latency between the retina burst and the convolution chip output burst which is always in the order of 1 ms. This is because the analog chip includes analog comparators in the pixel that decide when to generate output events. These comparators were biased for low power and have a bandwidth in the order of KHz. However, the present digital chip will produce output events as soon as sufficient representative events for recognizing the circle are received. This happens after receiving about 30–40 events. Fig. 16(b) shows the measured latencies between the retina first event and the convolution first event, as function of transient duration, for both chips. As can be seen, the latency for the analog chip stays approximately constant (it varies between 800 μs and 1 ms), while for the digital convolution chip it decreases linearly with the transient duration, until it saturates at about 18 μs. Consequently, this chip is capable of performing recognition of this shape within 18 μs, if input events are fed at maximum speed.

VII. Discussion and Conclusions

We have presented a convolution chip for AER event-driven vision sensing and processing systems. In such systems, dynamic visual information is represented by a continuous flow of timed events. In this work we have shown that in these systems input and output events of an AER convolution processing module are simultaneous in practice, since delays are in the range of 150 ns. This inherent property of AER processing modules opens the possibility of assembling hierarchically large number of them in a scalable manner, mimicking the cerebral cortex. Such structures would have minimum delays between input visual stimulus and output (recognition), although hundreds or thousands of convolutions were performed, because of the pseudosimultaneity property of continuous spike flow processing.

In this paper we have presented one such AER convolutional module, which connects to others through input and output AER asynchronous links. It is perfectly feasible with present day technology to assemble several tens of these convolutional modules in a network on chip (NoC) die [75], with some extra routing modules. This way, arbitrary networks of Conv modules can be assembled, even including feedback connections. Several tens of these NoC dies can then be assembled on PCBs, making it possible to have systems with several thousands of convolutional modules processing visual information in parallel and with submicrosecond event delays between them. At present, the largest reported AER multimodule system only had about 12 AER modules, four of which were ConvChips [68]. Future research is oriented towards miniaturizing modules, links, and simplifying reconfigurability.

The AER ConvChip module we have presented in this paper processes an array of 32×32 pixels with kernels of arbitrary shape and of size up to 32×32. Event latency can be as low as 155 ns, input event throughput can be as high as 20 Meps and output event rate can reach 45 Meps. A variety of experiments have been devised to characterize the performance and illustrate the processing capability of the chip.

Other researchers have reported attempts using commercial GPUs [79] achieving speed-ups of up to x35 with respect to baseline CPU. In particular, they could process 64×64 kernels at 200 keps on 128×128 input AER flow, but at the expense of loosing time resolution by collapsing multiple input spikes into a single data stream representing time windows of 1–5 ms, as well as a power consumption of about 250 W. Other groups have reported FPGA implementations of spiking convolutions

but using Poisson statistics probabilistic mappers, with potential of reaching 1.35 Meps for 11×11 kernels (163 MOPS) on 64×64 images [80].

Frame-based ConvNet Hardware Implementations are being developed by other groups [76]–[78]. Farabet [77] compares performance estimations of very well optimized ConvNets algorithms implemented on CPU (Core2 Duo 2.4 GHz Macbook), FPGA (Xilinx Virtex-4 SX35 at 200 MHz) and ASIC (Tezzaron 3D at 400 MHz). For an object detection ConvNet requiring a total of 920 convolutions of 7×7 kernels on a 500×500 pixel input image, the CPU can process at about 0.7 frames per second (fps), the FPGA at about 15 fps, and the ASIC at about 100 fps. Although these figures are quite impressive and have high potential for immediate commercial deployment, these techniques process image patches at high speeds, and thus their main limitation for up scaling is off-chip memory bandwidth. Nevertheless, the frame-based approach is very mature and ready to use in many embedded systems. On the other hand, the spike driven Convolutional approach presented here is still at an incipient development stage. However, the *pseudosimultaneity* and scalability properties are quite attractive for approaching brain-size systems with fast responses.

Acknowledgment

The authors are grateful to T. Delbrück for providing the AER temporal contrast retina [26] the jAER open software [33] and constructive feedback, A. Civit's group for the AER interfacing boards [72], P. Häfliger for the CAVIAR PCB for holding the ConvChip and the lens mount holder for the retina, and E. Culurciello and C. Farabert for valuable discussions.

References

[1] U. Mallik, M. Clapp, E. Choi, G. Cauwenberghs, and R. Etienne-Cummings, "Temporal change threshold detection imager," in *IEEE ISSCC Dig. Tech. Papers*, 2005, pp. 362–363.

[2] P. Lichtsteiner, C. Posch, and T. Delbruck, "A 128 × 128 12 dB 30 mW asynchronous vision sensor that responds to relative intensity change," in *IEEE ISSCC Dig. Tech. Papers*, 2006, pp. 508–509.

[3] C. Posch, M. Hofstatter, D. Matolin, G. Vanstraelen, P. Schon, N. Donath, and M. Litzenberger, "A dual-line optical transient sensor with on-chip precision time-stamp generation," in *IEEE ISSCC Dig. Tech. Papers*, 2007, pp. 500–618.

[4] N. Massari, M. Gottardi, and S. Jawed, "A 100 μW 64×128-pixel contrast-based asynchronous binary vision sensor for wireless sensor networks," in *IEEE ISSCC Dig. Tech. Papers*, 2008, pp. 588–638.

[5] P. F. Ruedi, P. Heim, S. Gyger, F. Kaess, C. Arm, R. Caseiro, J.-L. Nagel, and S. Todeschini, "An SoC combining a 132 dB QVGA pixel array and a 32 b DSP/MCU processor for vision applications," in *IEEE ISSCC Dig. Tech. Papers*, 2009, pp. 46–47, 47a.

[6] C. Posch, D. Matolin, and R Wohlgenannt, "A QVGA 143 dB DR asynchronous address-event PWM dynamic vision and image sensor with lossless pixel-level video compression and time-domain CDS," in *ISSCC Dig. Tech. Papers*, 2010, pp. 400–401.

[7] R. Sarpeshkar, M. W. Baker, C. D. Salthouse, J.-J. Sit, L. Turicchia, and S. M. Zhak, "An analog bionic ear processor with zero-crossing detection," in *ISSCC Dig. Tech. Papers*, 2005, pp. 78–79.

[8] B. Wen and K. Boahen, "A 360-channel speech preprocessor that emulates the cochlear amplifier," in *ISSCC Dig. Tech. Papers*, 2006, pp. 556–557.

[9] M. Sivilotti, "Wiring considerations in analog VLSI systems with application to field-programmable networks," Ph.D. dissertation, Comput. Neural Syst., California Inst. Technol., Pasadena, CA, 1991.

[10] M. A. Mahowald, "VLSI analogs of neuronal visual processing: A synthesis of form and function," Ph.D. dissertation, Comput. Neural Syst., California Inst. Technol., Pasadena, CA, 1992.

[11] J. Lazzaro, J. Wawrzynek, M. Mahowald, M. Silvilotti, and D. Gillespie, "Silicon auditory processors as computer peripherals," *IEEE Trans. Neural Netw.*, vol. 4, pp. 523–528, May 1993.

[12] G. Cauwenberghs, N. Kumar, W. Himmelbauer, and A. G. Andreou, "An analog VLSI chip with asynchronous interface for auditory feature extraction," *IEEE Trans. Circuits Syst. Part II, Analog Digit. Signal Process.*, vol. 45, pp. 600–606, May 1998.

[13] K. Boahen, "Retinomorphic chips that see quadruple images," in *Proc. Int. Conf. Microelectron. Neural, Fuzzy Bio-Inspired Syst. (Microneuro99)*, Granada, Spain, pp. 12–20.

[14] K. Boahen, "A retinomorphic chip with parallel pathways: Encoding INCREASING, ON, DECREASING, and OFF visual signals," *Int. J. Analog Integr. Circuits Signal Process.*, vol. 30, pp. 121–135, Feb. 2002.

[15] A. J. Martin and M. Nyström, "Asynchronous techniques for system-on-chip design," *Proc. IEEE*, vol. 94, no. 6, pp. 1089–1120, Jun. 2006.

[16] J. Sparsø and S. B. Furber, *Principles of Asynchronous Circuit Design: A Systems Perspective*. Norwell, MA: Kluwer, 2001.

[17] A. Mortara, E. A. Vittoz, and P. Venier, "A communication scheme for analog VLSI perceptive systems," *IEEE J. Solid-State Circuits*, vol. 30, pp. 660–669, Jun. 1995.

[18] K. Boahen, "Retinomorphic vision systems," presented at the Microneuro'96: Fifth Int. Conf. Neural Netw. Fuzzy Syst., Laousanne, Switzerland, Feb. 1996.

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

[20] E. Culurciello, R. Etienne-Cummings, and K. A. Boahen, "A biomorphic digital image sensor," *IEEE J. Solid-State Circuits*, vol. 38, pp. 281–294, 2003.

[21] P. F. Ruedi, P. Heim, F. Kaess, E. Grenet, F. Heitger, P.-Y. Burgi, S. Gyger, and P. Nussbaum, "A 128 × 128, pixel 120-dB dynamic-range vision-sensor chip for image contrast and orientation extraction," *IEEE J. Solid-State Circuits*, vol. 38, pp. 2325–2333, 2003.

[22] S. Chen and A. Bermak, "Arbitrated time-to-first spike CMOS image sensor with on-chip histogram equalization," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 15, no. 3, pp. 346–357, Mar. 2007.

[23] M. Azadmehr, J. Abrahamsen, and P. Häfliger, "A foveated AER imager chip," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS2005)*, Kobe, Japan, pp. 2751–2754.

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

[25] J. A. Leñero-Bardallo, T. Serrano-Gotarredona, and B. Linares-Barranco, "A five-decade dynamic-range ambient-light-independent calibrated signed-spatial-contrast AER retina with 0.1 ms latency and optional time-to-first-spike mode," *IEEE Trans. Circuits Syst. I, Reg. Papers*, to be published.

[26] P. Lichtsteiner, C. Posch, and T. Delbruck, "A 128 × 128 120 dB 30 mW asynchronous vision sensor that responds to relative intensity change," *IEEE J. Solid-State Circuits*, vol. 43, pp. 566–576, Feb. 2008.

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

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

[29] V. Chan, S.-C. Liu, and A. van Schaik, "Aer EAR: A matched silicon cochlea pair with address event representation interface," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 54, pp. 48–59, Jan. 2007.

[30] E. Chicca, A. M. Whatley, P. Lichtsteiner, V. Dante, T. Delbruck, P. Del Giudice, R. J. Douglas, and G. Indiveri, "A multichip pulse-based neuromorphic infrastructure and its application to a model of orientation selectivity," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 54, no. 5, pp. 981–993, May 2007.

[31] M. Oster, Y. Wang, R. Douglas, and S.-C. Liu, "Quantification of a spike-based winner-take-all VLSI network," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 55, no. 10, pp. 3160–3169, Nov. 2008.

[32] T. Teixeira, A. G. Andreou, and E. Culurciello, "Event-based imaging with active illumination in sensor networks," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS2005)*, Kobe, Japan, pp. 644–647.

[33] [Online]. Available: http://jaer.wiki.sourceforge.net

[34] T. Delbrück, "Frame-free dynamic digital vision," in *Proc. Int. Symp. Secure-Life Electron., Adv. Electron. Quality Life Soc.*, Mar. 6–7, 2008, pp. 21–26.

[35] R. J. Vogelstein, U. Mallik, E. Culurciello, G. Cauwenberghs, and R. Etienne-Cummings, "A multi-chip neuromorphic system for spike-based visual information processing," *Neural Comput.*, vol. 19, no. 9, pp. 2281–300, 2007.

[36] D. H. Hubel and T. N. Wiesel, "Receptive fields of single neurones in the Cat's striate cortex," *J. Physiol.*, vol. 148, pp. 574–591, 1959.

[37] E. T. Rolls and G. Deco, *Computational Neuroscience of Vision*. New York: Oxford Univ. Press, 2002.

[38] P. Vernier, A. Mortara, X. Arreguit, and E. A. Vittoz, "An integrated cortical layer for orientation enhancement," *IEEE J. Solid-State Circuits*, vol. 32, pp. 177–186, Feb. 1997.

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

[40] 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.

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

[42] R. Serrano-Gotarredona, T. Serrano-Gotarredona, A. Acosta-Jiménez, C. Serrano-Gotarredona, J. A. Pérez-Carrasco, B. Linares-Barranco, A. Linares-Barranco, G. Jiménez-Moreno, and A. Civit-Ballcels, "On real-time AER 2-D convolution hardware for neuromorphic spike-based cortical processing," *IEEE Trans. Neural Netw.*, vol. 19, no. 7, pp. 1196–1219, Jul. 2008.

[43] 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.

[44] 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.

[45] 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.

[46] 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.

[47] 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.

[48] 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, 1997.

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

[50] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, "Backpropagation applied to handwritten zip code recognition," *Neural Comput.*, vol. 1, no. 4, pp. 541–551, 1989.

[51] K. Chellapilla, M. Shilman, and P. Simard, "Optimally combining a cascade of classifiers," presented at the Int. Symp.. Document Recog. Retrieval, San Jose, CA, 2006.

[52] R. Vaillant, C. Monrocq, and Y. LeCun, "Original approach for the localisation of objects in images," *IEE Proc. Vis., Image, Signal Process.*, vol. 141, no. 4, pp. 245–250, Aug. 1994.

[53] M. Osadchy, Y. LeCun, and M. Miller, "Synergistic face detection and pose estimation with energy-based models," *J. Mach. Learn. Res.*, vol. 8, pp. 1197–1215, May 2007.

[54] C. Garcia and M. Delakis, "Convolutional face finder: A neural architecture for fast and robust face detection," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 26, no. 11, pp. 1408–1423, 2004.

[55] F. Nasse, C. Thurau, and G. A. Fink, "Face detection using gpu-based convolutional neural networks," *Lecture Notes in Computer Science, Computer Analysis of Images and Patterns*, vol. 5702/2009, pp. 83–90, 2009.

[56] A. Frome, G. Cheung, A. Abdulkader, M. Zennaro, B. Wu, A. Bissacco, H. Adam, H. Neven, and L. Vincent, "Large-scale privacy protection in Google street view," in *Proc. Int. Conf. Comp. Vis. (ICCV'09)*, 2009, pp. 2373–2380.

[57] T. Serre, L. Wolf, S. Bileschi, M. Riesenhuber, and T. Poggio, "Robust object recognition with cortex-like mechanisms," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 29, no. 3, pp. 411–426, Mar. 2007.

[58] V. Jain and H. S. Seung, "Natural image denoising with convolutional networks," in *Advances in Neural Information Processing Systems*. Cambridge, MA: MIT Press, 2008, vol. 21.

[59] S. Nowlan and J. Platt, "A convolutional neural network hand tracker," in *Advances in Neural Information Processing Systems*, G. Tesauro, D. Touretzky, and T. Leen, Eds. Cambridge, MA: MIT Press, 1995, vol. 7, pp. 901–908.

[60] R. Hadsell, P. Sermanet, M. Scoffier, A. Erkan, K. Kavackuoglu, U. Muller, and Y. LeCun, "Learning long-range vision for autonomous off-road driving," *J. Field Robot.*, vol. 26, no. 2, pp. 120–144, Feb. 2009.

[61] F. Ning, D. Delhomme, Y. LeCun, F. Piano, L. Bottou, and P. Barbano, "Toward automatic phenotyping of developing embryos from videos," *IEEE Trans. Image Process.*, vol. 14, no. 9, pp. 1360–1371, Sep. 2005.

[62] V. Jain, J. F. Murray, F. Roth, S. Turaga, V. Zhigulin, K. Briggman, M. Helmstaedter, W. Denk, and H. S. Seung, "Supervised learning of image restoration with convolutional networks," in *Proc. Int. Conf. Comp. Vis. (ICCV'07)*, pp. 1–8.

[63] J. A. Pérez-Carrasco, B. Acha, C. Serrano, L. Camuñas-Mesa, T. Serrano-Gotarredona, and B. Linares-Barranco, "Fast vision through frame-less event-based sensing and convolutional processing. Application to texture recognition," *IEEE Trans. Neural Netw.*, vol. 21, no. 4, pp. 609–620, Apr. 2010.

[64] R. VanRullen and S. J. Thorpe, "Rate coding versus temporal order coding: What the retinal ganglion cells tell the visual cortex," *Neural Comput.*, vol. 13, no. 6, pp. 1255–1283, Jun. 2001.

[65] A. Delorme, J. Gautrais, R. Van Rullen, and S. S. Thorpe, "Spikenet: A simulator for modeling large networks of integrate and fire neurons," *Neurocomputing*, vol. 26–27, pp. 989–996, 1999.

[66] J. Lazzaro and J. Wawrzynek, "A multi-sender asynchronous extension to the AER protocol," in *Proc. 16th Conf. Adv. Res. VLSI (ARVLSI'95)*, pp. 158–158.

[67] S. R. Deiss, R. J. Douglas, and A. M. Whatley, "A pulse-coded communications infrastructure for neuromorphic systems," in *Pulsed Neural Networks*, W. Maass and C. M. Bishop, Eds. Cambridge, MA: MIT Press, 1999, ch. 6, pp. 157–178.

[68] R. Serrano-Gotarredona, M. Oster, P. Lichtsteiner, A. Linares-Barranco, R. Paz-Vicente, F. Gómez-Rodríguez, L. Camuñas-Mesa, R. Berner, M. Rivas, T. Delbrück, S. C. Liu, R. Douglas, P. Häfliger, G. Jiménez-Moreno, A. Civit, T. Serrano-Gotarredona, A. Acosta-Jiménez, and B. Linares-Barranco, "Caviar: A 45 k neuron, 5 M synapse, 12 G connects/s AER hardware sensory-processing-learning-actuating system for high-Speed visual object recognition and tracking," *IEEE Trans. Neural Netw.*, vol. 20, no. 9, pp. 1417–1438, Sep. 2009.

[69] P. Merolla, J. Arthur, B. E. Shi, and K. Boahen, "Expandable networks for neuromorphic chips," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 54, no. 2, pp. 301–311, Feb. 2007.

[70] M. J. Bellido, A. J. Acosta, J. Luque, A. Barriga, and M. Valencia, "Evaluation of metastability transfer models: An application to an N-bistable CMOS synchronizer," *Int. J. Electron.*, vol. 79, no. 5, pp. 585–593, 1995.

[71] K. A. Boahen, "A burst-mode word-serial address-event link – Part I: Transmitter design," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 51, no. 7, pp. 1269–1280, Jul. 2004.

[72] A. Linares-Barranco, G. Jiménez-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.

[73] F. Gómez-Rodríguez, R. Paz, A. Linares-Barranco, M. Rivas, L. Miro, S. Vicente, G. Jimenez, and A. Civit, "AER tools for communications and debugging," in *Proc. IEEE Int. Symp. Circuits Syst. 2006 (ISCAS'06)*, pp. 3253–3256.

[74] 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," in *Proc. 2008 IEEE Int. Symp. Circuits Syst. (ISCAS08)*, pp. 2150–2153.

[75] S. R. Vangal, J. Howard, G. Ruhl, S. Dighe, H. Wilson, J. Tschanz, D. Finan, A. Singh, T. Jacob, S. Jain, V. Erraguntla, C. Roberts, Y. Hoskote, N. Borkar, and S. Borkar, "An 80-Tile 1.28 TFLOPS network-on-chip in 65 nm CMOS," in *Proc. IEEE Int. Solid-State Circuits Conf. (ICCSC07)*, pp. 98–99.

[76] C. Fabaret, C. Poulet, J. Y. Han, and Y. LeCun, "Cnp: An FPGA-based processor for convolutional networks," in *Int. Conf. Field Progr. Logic Appl.*, 2009.

[77] C. Farabet, B. Martini, P. Akselrod, S. Talay, Y. LeCun, and E. Culurciello, "Hardware accelerated convolutional neural networks for synthetic vision systems," in *Proc. 2010 IEEE Int. Symp. Circuits Syst. (ISCAS10)*, 2010, pp. 257–260.

[78] F. Mamalet, S. Roux, and C. Garcia, "Embedded facial image processing with convolutional neural networks," in *Proc. 2010 IEEE Int. Symp. Circuits Syst. (ISCAS10)*, 2010, pp. 261–263.

[79] J. M. Nageswaran, N. Dutt, Y. Wang, and T. Delbrück, "Computing spike-based convolutions on GPUs," in *Proc. 2009 IEEE Int. Symp. Circuits Syst. (ISCAS09)*, pp. 1917–1920.

[80] A. Linares-Barranco, R. Paz-Vicente, F. Gómez-Rodríguez, A. Jiménez, M. Rivas, G. Jiménez, and A. Civit, "On the AER convolution processors for FPGA," in *Proc. 2010 IEEE Int. Symp. Circuits Syst. (ISCAS10)*, 2010, pp. 4237–4240.