Multi-Casting Mesh AER: A Scalable Assembly Approach for Reconfigurable Neuromorphic Structured AER Systems. Application to ConvNets

C. Zamarreño-Ramos, A. Linares-Barranco, T. Serrano-Gotarredona, and B. Linares-Barranco

***Abstract*— This paper presents a modular, scalable approach to assembling hierarchically structured neuromorphic AER (Address Event Representation) systems. The method consists of arranging modules in a 2D mesh, each communicating bidirectionally with all four neighbors. Address events include a module label. Each module includes an AER router which decides how to route address events. Two routing approaches have been proposed, analyzed and tested, using either destination or source module labels. Our analyses reveal that depending on traffic conditions and network topologies either one or the other approach may result in better performance. Experimental results are given after testing the approach using high-end Virtex-6 FPGAs. The approach is proposed for both single and multiple FPGAs, in which case a special bidirectional parallel-serial AER link with flow control is exploited, using the FPGA Rocket-I/O interfaces. Extensive test results are provided exploiting convolution modules of 64×64 pixels with kernels with sizes up to 11×11, which process real sensory data from a DVS (Dynamic Vision Sensor) retina. One single Virtex-6 FPGA can hold up to 64 of these convolution modules, which is equivalent to a neural network with 262×10^3 neurons and almost 32 million synapses.**

I. Introduction

AER (Address-Event-Representation) is now a popular "virtual wiring" technique for interconnecting spiking neuromorphic systems [1]–[33]. The high-speed available for digital inter-chip communications is exploited in AER to time-multiplex numerous synaptic connections between neurons, which only need to be active during a spike (also called event) transmission. In AER, whenever a spiking neuron in a chip (or module[1]) generates a spike, its "address" (or any given ID) is written on a high speed digital bus and sent to the receiving neuron(s) in one (or more) receiver module(s). In general, AER processing modules require at least one AER input port and one AER output port. As neuromorphic systems scale up in size, complexity, and functionality, researchers have been developing more complex and smarter AER "variations" to maintain the efficiency, reconfigurability and reliability of the ever growing target systems they want to build.

Authors are with the Instituto de Microelectrónica de Sevilla (IMSE-CNM-CSIC). Av. Américo Vespucio s/n, 41092 Sevilla, Spain.

[1]Throughout the paper we will be referring to generic AER modules, where a module can be one chip, several chips grouped into a PCB, several PCBs, or even a part of a chip or FPGA. One chip/FPGA could therefore hold several AER modules as well as AER buses.

In the following Section we review a set of approaches for large scale reconfigurable AER systems that have been proposed by different researchers, ranging from the sharing of a single AER bus by all AER modules, to the use of multiple independent buses or mesh type arrangements of modules which exploit communication techniques from the NoC (Network-on-Chip) research community.

II. Review of AER approaches for large scale systems

Table I summarizes a set of AER assembly techniques proposed in literature for tackling the growth of AER based processing systems. The simplest form of a generic AER concept for use in a large scale multi-module spiking neuromorphic system is illustrated in Fig. 1(a). Let us call it **"Flat-AER"**. Each module can contain, for example, an array of neurons. Each neuron is assigned a unique global address, which identifies the module it belongs to and its position inside the module. This way, the address space of all modules' input and output AER ports is the same. All modules share a single external AER bus [3]–[9]. Connectivity between neurons is configurable and set by a look-up-table in the external programmable "Mapper". Multi fan-out can be programmed in the mapper by repeating multiple destination addresses for each incoming address. Similarly, synaptic weighting can also be implemented by programming destination address repetitions. However, event repetition (for either fan-out or weighting) severely penalizes the AER bus communication bandwidth. To overcome this, some reported neuron chips allow for an additional synaptic weight parameter to be programmed into the mapper together with the event address [7], [10]. Alternatively, some other reported neuron chips include a built-in mechanism which will implement a given fan-out and synaptic weighting from a single input event (as in pre-wired diffusive networks [11], [12] or more elaborate computational hardware [13], [18], [22], [24]). In this case, the mapper would only need to repeat an event if it is destined for neurons belonging to different modules. Normally, event addresses represent neurons. However, in some reported neuron chips that include a number of physical synapses per neuron [6], the input address can represent one specific synaptic input. In this case, the address spaces at the mapper input and output would be different, as they represent different elements.

Flat-AER is simple and easy to build, configure, and use. It requires a mapper memory with as many positions as there are

neurons in the system. However, the main limitation of *flat-AER* is its communication bandwidth. Since every single event produced by any neuron has to travel through the single AER bus and Mapper, the system's maximum total event traffic is limited by the bus bandwidth. If N_{tot} is the total number of neurons, f_n is the mean spike rate per neuron and F_{out} the average fan-out per neuron (spike repetitions introduced by the mapper to emulate the projection fields and/or synaptic weighting), the event arrival rate λ at the Mapper output channel is [34] $\lambda = N_{tot} f_n F_{out}$. If E_{flat} is the physical channel bandwidth, then the channel service rate is $\mu = E_{flat}$ and the average time an event waits to be serviced is (assuming an M/M/1 queue model [35]) $\bar{t_q} = 1/(\mu - \lambda)$. Consequently, the absolute maximum communication bandwidth (number of events per unit time) Ev this approach can handle is obtained when $\lambda = E_{flat}$ and is

$$Ev_{max} = N_{tot} f_n \Big|_{max} = \frac{E_{flat}}{F_{out}} \tag{1}$$

Given Ev_{max}, it is possible to estimate the maximum allowable number of neurons for such communication architecture

$$N_{tot\_max} = \frac{Ev_{max}}{f_n} \tag{2}$$

Note that, in general and specially for rate encoded weights, $F_{out} = n_{RF} \times n_W$ (n_{RF} is the projection field size and n_W is the synaptic weight dynamic range) can become significantly large. For example, if the projection field has a size of $n_{RF} = 11 \times 11$ neurons and weights can have integer values ranging from $n_W = 1$ to 32, then $F_{out} = 3872$.

Reported AER-bus bandwidths are presently below $100Meps$ (mega events per second) for point-to-point links [3]–[7], [11]–[13], [18], [22], [24]–[28], [36]–[38], although cases have been reported of high density channels and multiplexing techniques being used to achieve higher event rates [8], [33]. *Flat-AER* therefore allows for a total communication bandwidth in the range of $10^8 eps$ down to $10^4 eps$, depending on fan-out.

Having several modules sharing the same physical lines degrades speed proportionally to the number of modules [26]. This can be overcome by using *Broadcast-Mesh-AER*.

"Broadcast-Mesh-AER" uses multiple point-to-point AER buses [25]–[27]. Fig. 1(b) shows its corresponding 1-D version. Each neuron in each module also has a unique global flat address. Each module has an AER input event path and an AER output event path, each with an AER input port and an AER output port. AER input events received at the input **AERi** port are sent to the module neuron array but are also passed through to the next module, via output port **AERi'**. This way, input events "hop" from module to module via independent AER point-to-point links. Consequently, the speed at each AER link is optimum and events are copied more efficiently in a pipeline fashion. Output events generated in each module also "hop" from module to module through **AERo** and **AERo'** ports until they reach the Mapper. In this scheme all input events coming from the Mapper are broadcast to all modules, and each module checks if the event is destined to the local neural array [25], [26]. However, overall network connectivity information is contained in the global mapper, and can be totally reconfigured by reprogramming the mapper (as in the *Flat-AER* approach).

One major claim of this approach is that the channel bandwidth of each point-to-point link E_{pp} improves proportionally with the number of chips N_{ch} in the network, with respect to the *Flat-AER* case where N_{ch} chips share the same AER bus[2] as in Fig. 1(a). The channel bandwidth of a point-to-point link E_{pp} is constant, while that of a multiple fan-out link E_{flat} degrades with the number of destination chips N_{ch}. More precisely, for typical PCBs, the bandwidth improvement of a point-to-point link is

$$E_{pp} \approx 2\,(N_{ch} - 1)\,E_{flat} \tag{3}$$

The network now has N_{ch} point-to-point links, which allows for a total communication bandwidth of $E_{pp} \times N_{ch}$. However, each event has to be copied to each link, so the maximum event rate is

$$Ev_{max} = \frac{E_{pp} N_{ch}}{F_{out} N_{ch}} = \frac{E_{pp}}{F_{out}} \tag{4}$$

Bandwidth is thus improved by improving E_{pp} with respect to E_{flat} proportionally to the number of chips [26].

Both *Flat-AER* and *Broadcast-Mesh-AER* use a common global flat address space and, in principle, allow for any arbitrary interconnect topology. However, practical neuromorphic systems have a pre-established hierarchical structure, depending on the functionality they implement. This has been exploited by other researchers to assemble scalable multi-module systems with independent AER-links, where each link is a physical plugged-in point-to-point bus-wire [28]. This is illustrated in Fig. 1(c), where AER splitters (blocks labeled "S" in Fig. 1(c)) and mergers (blocks labeled "M" in Fig. 1(c)) are also used for branching or de-branching links. A splitter block receives one input AER channel and replicates the traffic for n different output channels, while a merger block multiplexes n input AER channels into a single output channel. All physical links are point-to-point. In this **"Pre-Structured AER"** approach the address space is local to the neurons writing to or reading from an AER link. Optionally, local mappers can be inserted in a link to adapt address spaces from an output to an input (for example, to perform subsampling, address rotations, bit reallocations, etc.). In this approach no global Mapper is required, as the connectivity is pre-wired, and events do not need to travel through all the links. The number of links scales with the number of modules, so communication bandwidth saturation is much less likely to occur as systems scale up. However, system reconfiguration is laborious as it has to be done manually by re-plugging bus-wires, splitters, mergers, mappers, and processing modules.

In this case each point-to-point channel receives events from only a small fraction of the modules/chips. In general, we can define an effective number of independent channels M_{eff} as a fraction of the total number of chip modules $M_{eff} = \alpha N_{ch}$. The network maximum communication bandwidth would then be

[2]For a more detailed explanation see [26].

TABLE I

MULTI-MODULE AER ADDRESSING SCHEMES

		Flat AER [7]	Broadcast Mesh AER [25]–[27]	Pre-Structured AER [28]	Hierarchical Fractal AER [29]	Router Mesh AER [30], [31]	Cross-Point Interc. [32]	Multi-Casting Mesh AER
Addressing	Address space	flat	flat	local	local	flat	local	flat
	Broadcast to all modules	Yes	Yes	No	No	No	No	No
	Local Routing Tables to define global network	No	No	No	Yes	Yes	Yes	Yes
	Single global mapper	Yes	Yes	No	No	No	No	No
Module Props.	isolated neurons	Yes	No	No	Yes	No	No	No
	neurons with synapses	No	Yes	Yes	No	Yes	Yes	Yes
	events with synaptic weighting	Yes	No	No	Yes	No	No	No
	projection fields with synaptic weighting	No	Yes	Yes	No	Yes	Yes	Yes
	physical synapses	No	Yes	No	Yes	No	Yes	No

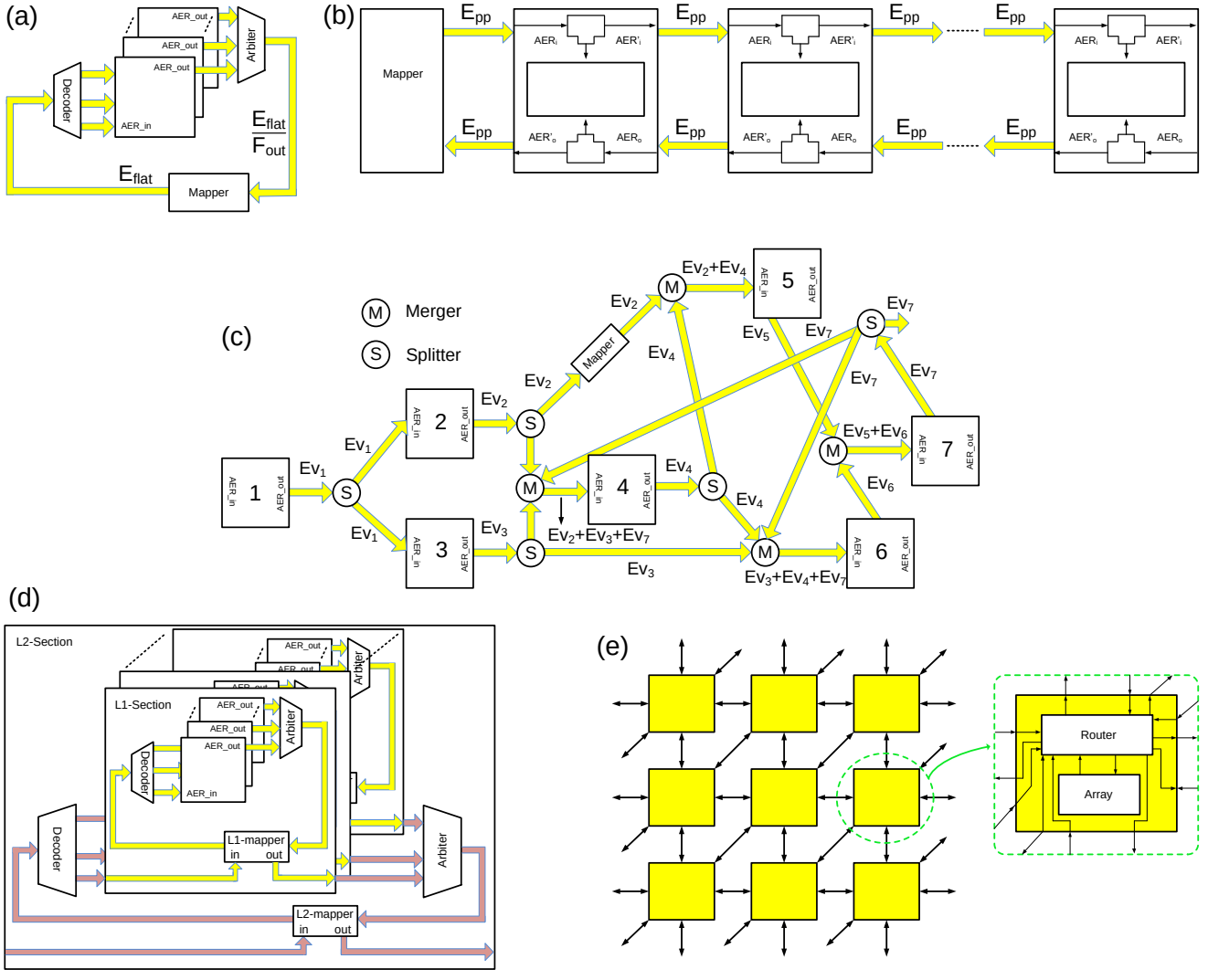


Fig. 1. Illustration of different multi-module AER assembly options: (a) Flat-AER, (b) Broadcast-Grid-AER, (c) Pre-Structured-AER, (d) Hierarchical-Fractal-AER, (e) Router-Mesh-AER.

$$Ev_{max} = M_{eff} E_{pp} = \alpha N_{ch} E_{pp} \tag{5}$$

Parameter F_{out} does not appear anywhere, since the projection fields and synaptic weighting are now implemented inside each event-processing module. As an illustrative example, Fig. 1(c) has $N_{ch} = 7$ modules, each generating an output event rate Ev_i. The distribution of splitters and mergers determine potential bottlenecks as

$$\begin{gathered} Ev_{1_{max}} \leq E_{pp} \ , \ \ Ev_{2_{max}} + Ev_{4_{max}} \leq E_{pp} \\ Ev_{2_{max}} + Ev_{3_{max}} + Ev_{7_{max}} \leq E_{pp} \\ Ev_{5_{max}} + Ev_{6_{max}} \leq E_{pp} \\ Ev_{3_{max}} + Ev_{4_{max}} + Ev_{7_{max}} \leq E_{pp} \end{gathered} \tag{6}$$

under these constraints, the absolute maximum capacity of this particular network can be obtained by applying constraint optimization to eqs. (6), resulting in

$$Ev_{max} = \sum Ev_{i_{max}} = \tag{7}$$

$$= \left(1 + \frac{1}{2} + \frac{1}{4} + \frac{1}{2} + \frac{1}{2} + \frac{1}{2} + \frac{1}{4}\right) E_{pp} = 4E_{pp}$$

Thus, in this case $M_{eff} = 4$ and $\alpha = M_{eff}/N_{ch} = 0.57$. In this approach the optimum point-to-point bandwidth E_{pp} is therefore further improved, proportionally to the number of chips (by a factor $\alpha N_{ch} = M_{eff} = 4$, in this case).

Joshi et al. recently suggested a **"Hierarchical-Fractal-AER"** approach [29], illustrated in Fig. 1(d), which extends the basic Flat-AER concept of Fig. 1(a) in a hierarchical fashion. It exploits the assumption that nearby neurons are more heavily interconnected than more distant ones. Address space is expanded as events need to climb up in the hierarchy. This way, more intense local traffic is transferred very fast in parallel at the numerous lower level modules, while longer range but sparser traffic needs to traverse levels of hierarchy and is slower. Disregarding the traffic at the higher hierarchies, if M_{L1} is the number of lowest level L_1 parallel sections and N_{L1} the number of chips per L_1 section ($N_{ch} = M_{L1} \times N_{L1}$), then the absolute maximum network bandwidth would be

$$Ev_{max} = M_{L1} \frac{E_{hier}}{F_{out}} \tag{8}$$

with $E_{pp} \approx 2\left((N_{ch}/M_{L1}) - 1\right) E_{hier}$. This way, maximum network bandwidth can be expressed as

$$Ev_{max} \approx \frac{1}{2} \frac{N_{ch}}{N_{L1}^2} \frac{E_{pp}}{F_{out}} = \frac{1}{2} \frac{M_{L1}}{N_{L1}} \frac{E_{pp}}{F_{out}} \tag{9}$$

Depending on the ratio M_{L1}/N_{L1} a considerable improvement can be achieved with respect to *Flat-AER*.

Another approach, illustrated in Fig. 1(e), is what we call here **"Router-Mesh-AER"** [30]. Here again neurons have a global flat address which identifies their module and their address within the module, but there is no external Mapper through which all events have to pass. Instead, the mapping table is contained within a "*Router*" in each module. The router also decides the ports through which an event is sent to reach its destination module. Events are therefore not broadcast to all modules, but optimum "hop" paths are established in a multicast fashion. The main problem is that a very large mapping table needs to be programmed in each module. However, to simplify these tables, optimizations can be computed for each module router depending on the system topology, and default routing paths can be established for event addresses not listed in the tables [30]. This mesh approach has been traditionally used in NoC (Network on Chip) [39] topologies to assemble high performance multi-core processor systems for high performance computing applications [40]–[42]. The maximum network bandwidth for mesh type approaches will be computed in the next Section.

Another approach currently being developed, but at wafer scale [32], exploits massive programmable cross-point interconnects to reconfigure the network topology. We have included this method in Table I for comparison, although we are focusing more on multi-chip systems. Nonetheless, this wafer scale approach also includes off-wafer re-routing (and event re-timing) procedures for longer range and delay-controlled interconnects [33].

In all of the previously listed methods that use a global flat address space, each event includes a module ID that corresponds to the module where the event was generated. Let us call this "*Source-driven*" coding. However, it is also possible to label the event with the ID of the destination module instead. Let us call this "*Destination-driven"* coding.

In this paper we will consider a type of Pre-Structured-AER approach. However, instead of having manually pluggable links, modules are arranged in a 2D-Mesh while inter-module links are configured through in-module routers. We call this **"Multi-Casting-Mesh-AER"**. The approach is similar to the "*Router-Mesh-AER*" except that the module router tables only contain information on the module-to-module links, instead of the full inter-neuron connectivity. We will analyze both '*source-driven*' and '*destination-driven*' codings and will show experimental results from example vision processing systems implemented on FPGA prototyping boards, based on Convolutional Neural Networks (ConvNets) [43]–[47].

Fig. 2 shows the 2D network topology we used for *Multi-Casting-Mesh-AER*. The modules communicate bidirectionally and orthogonally with their neighbors through point-to-point AER links. The number of links in the network (and, thus, the network bandwidth) depends on the number of links per module. For the case shown in Fig. 2, each node has 4 links. If the mesh has $N_{ch} = M_1 M_2$ modules, the total number of inter-module links is[3] $N_l = 2[(M_1 - 1)M_2 + (M_2 - 1)M_1] = 4M_1M_2 - 2(M_1 + M_2) \approx 4N_{ch}$. Each event needs to hop through a number of links to travel from its source module to its destination module. Let us call n_h the average number of hops per traveling event. This average number n_h

[3]For the specific *Router-Mesh-AER* shown in Fig. 1(e) there are 6 inter-module links, thus $N_l = 2[(M_1-1)M_2 + (M_2-1)M_1 + (M_1-1)(M_2-1)] = 6M_1M_2 - 4(M_1+M_2) + 2 \approx 6N_{ch}$. In a 2D mesh, up to 8 links per module are possible. In a 3D mesh, up to 26 links per module are possible.

is application specific, but would usually be in the order of a fraction of M_1 or M_2. The network absolute maximum bandwidth is

$$Ev_{max} = \frac{N_l}{n_h} \frac{E_{pp}}{F_{Mout}} \tag{10}$$

where F_{Mout} is the module fan-out, representing the number of routing paths a single event is sent through in order to reach all its destination nodes.

Table II summarizes how the absolute maximum communication bandwidth Ev_{max} scales with number of chips N_{ch} for the different multi-chip approaches. From eq. (2), the maximum allowable number of neurons N_{tot_max} would also scale proportionally to Ev_{max} (for a fixed f_n). We can see that for *Flat-AER*, Ev_{max} (and N_{tot_max}) scales down with N_{ch}, while for *Broadcast-Mesh-AER*, Ev_{max} (and N_{tot_max}) stays constant. *Hierarchical-Fractal-AER* can be made to scale efficiently depending on the relative choice of M_{L1} and N_{L1}. For example, if N_{L1} is fixed, then Ev_{max} would scale linearly with N_{ch}. All three approaches (*Flat, Broadcast-Mesh, Hierarchical-Fractal*) are penalized by neuron fan-out F_{out}. *Pre-Structured-AER* is the most efficient, as it scales linearly with N_{ch} and has no fan-out penalty. However, it is not practical from a reconfigurability point of view. The 2D mesh-based approaches scale linearly with N_{ch}, although term n_h might have (square-root or log-type) dependence on N_{ch} depending on the specific application. However, the module fan-out F_{Mout} penalty is usually relatively small.

III. Routing in Multi-Casting-Mesh-AER

In Fig. 2, each module in the mesh is identified by a 2D index (*xNODE*, *yNODE*). From now on let us call each module in this 2D mesh an *AER-node*. The internal structure of an *AER-node* is shown in Fig. 3. It contains a *Router*, a local *Event Processor* (or neuron/synapse array), and a *Configuration Processor* (to set configurable parameters in the *Event Processor* or *Router*). The *Router* receives external events from the four neighbors and, based on its programmed routing tables, decides whether to send them to the local processors or to other neighbors. For events generated by the internal *Event Processor*, the *Router* adds the corresponding node index and sends them through the programmed ports. Thus, the *Router* introduces a network layer between the processing units' logic layer and the physical layer implementation. A heading bit distinguishes between *configuration commands* to be handled by the *Configuration Processor*, and *data events* to be handled by the *Event Processor*. The *Configuration Processor* can also receive commands through an SPI (Serial Peripheral Interface) connection. Other heading information identifies the node 2D index (*xNODE*, *yNODE*) coded in the event. This index identifies either the source node sending the event to the mesh (in the case of a *Source-Driven* addressing scheme), or the destination node in the mesh to which the event is being sent (in the case of a *Destination-Driven* addressing scheme). Each addressing mode has pros and cons which are analyzed throughout the paper. For both cases, Fig. 4 shows the proposed 32-bit event format containing two fields:

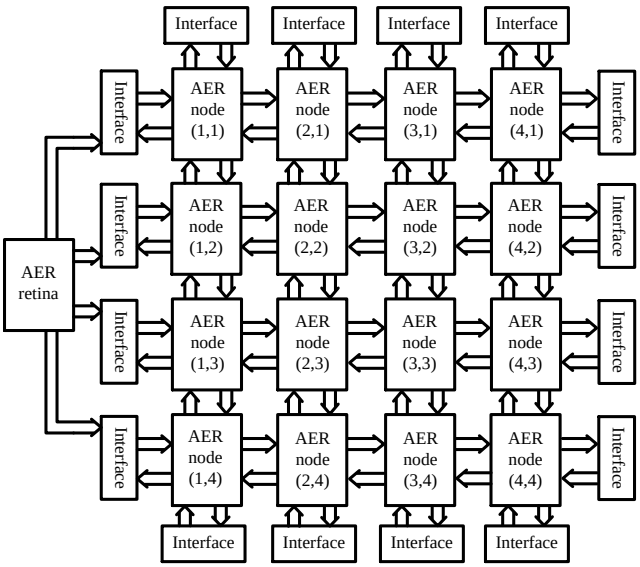


Fig. 2. 2D Network Topology for Multi-Casting-Mesh-AER. Each node is identified by the address field (*xNODE*, *yNODE*).

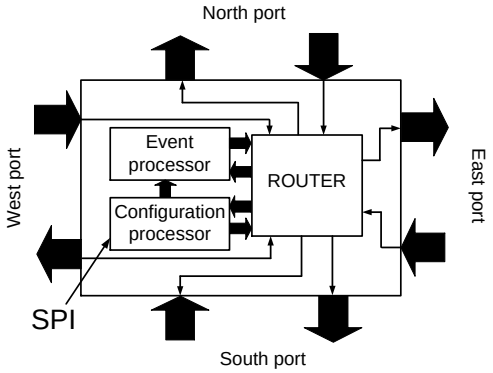
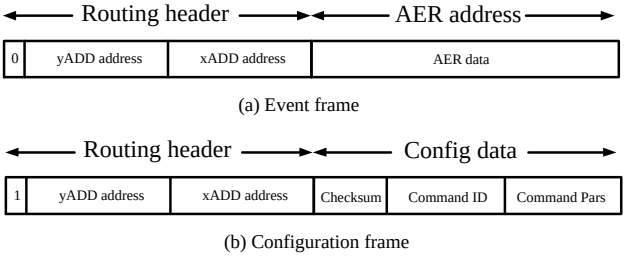


Fig. 3. Example of AER node for a multi-node multi-link AER system.



(a) Event frame

(b) Configuration frame

Fig. 4. Events format with headers added for routing purposes.

TABLE II
SCALING OF COMMUNICATION BANDWIDTHS IN MULTI-CHIP AER SCHEMES

	Flat AER [7]	Broadcast Mesh AER [25]–[27]	Pre-Structured AER [28]	Hierarchical Fractal AER [29]	Router Mesh AER [30]	Pre-Struc. Mesh AER
Ev_{max}	$\frac{E_{pp}}{2N_{ch}F_{out}}$	$\frac{E_{pp}}{F_{out}}$	$\alpha N_{ch}E_{pp}$	$\frac{N_{ch}E_{pp}}{2N_{L1}^2F_{out}}$	$\frac{6N_{ch}E_{pp}}{n_hF_{Mout}}$	$\frac{4N_{ch}E_{pp}}{n_hF_{Mout}}$

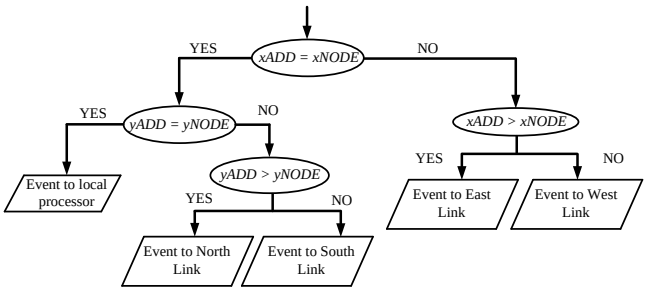


Fig. 5. Destination-driven routing algorithm for handling incoming events. (*xNODE*, *yNODE*) is the local node address and (*xADD*, *yADD*) is the event destination address.

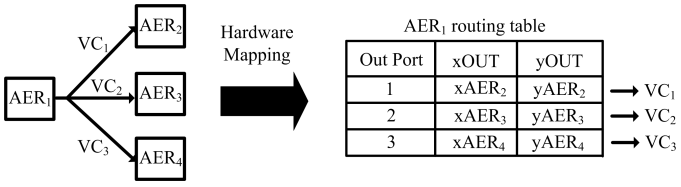


Fig. 6. Routing table for cloning output events in the destination driven algorithm. Left: Example logic diagram (schematics) showing the logic (virtual) connections between a source module AER_1 and three destination nodes AER_{2-4}. Right: Routing table showing the output event header to be added ($xOUT$,$yOUT$) and the port through which it is to be sent. VC_i (i=1,2,3) represents a virtual connection between the source node AER_1 and the destination nodes AER_{i-1}. ($xAER_i$,$yAER_i$) (i=2,3,4) is the network address of node AER_i.

- **Routing header:** the most significant bit is used to distinguish between data event (first bit is '0') or configuration command (first bit is '1'). The next 8 bits are used to code the destination or source node ID. Coordinates *xADD* and *yADD* are represented using 4 bits for each one.
- **Upper layer data:** the remaining 23 bits contain the event/command data. If it is a configuration command, it contains a command description, for example, a checksum, a command identifier and command parameters.

A. Destination-Driven Routing Algorithm

In this algorithm, the destination node address is written in the routing header. When the event arrives at a network node, the router analyzes the addressing header and decides the output port to which the event is to be forwarded. If the destination address corresponds to the node address, the event is sent to the local processor. If this is not the case, the event is routed in accordance with the algorithm represented in Fig. 5: it compares the event destination address *xADD* and *yADD* with the present node address *xNODE* and *yNODE* to decide the output port to which the event is to be forwarded. Using the geographical information contained in the destination address, the event is routed to the neighbor node with the shortest path to the destination in terms of the number of hops. As the router only has to compare two 4-bit digital words, the hardware required can be very simple and the routing operation can be performed on the fly. The algorithm in Fig. 5 gives priority to *xADD*. This is called dimension-ordered routing and tends to concentrate the traffic in one dimension (horizontal, in this case). To avoid this, routers which give priority to *yADD* can be alternated with those priming *xADD*, balancing the situation. However, this may yield to deadlock situations [48] and should be analyzed carefully for each case. On the other hand, dimension-ordered routing (with bi-directional links) is known to be deadlock-free [49], [50].

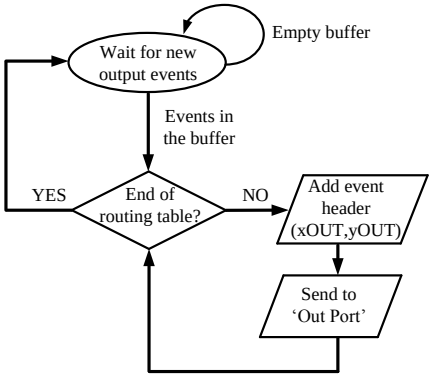


Fig. 7. Output event management in the destination-driven routing algorithm. Values for "xOUT", "yOUT" and "Out Port" are read from the routing table, as in the Fig. 6 example.

Besides managing the traffic coming from the neighboring chips, the router also inserts headers for the new events created in the local processor. The local router clones each newly created event as many times as the number of destination nodes (or virtual connections VC_i) there are for that event. For each clone, or virtual connection VC_i, the router adds the destination node address and sends it to one of the local output ports. This is organized in a routing table which is read for every new event. Every entry in this table has an output destination address and an output port that transmits the event. The routing table organization is illustrated in Fig. 6, for the case of one node AER_1 having three virtual connections $VC_1 - VC_3$ to three other nodes AER_2-AER_4. Fig. 7 shows the flow diagram for the output event management in the destination-driven routing algorithm.

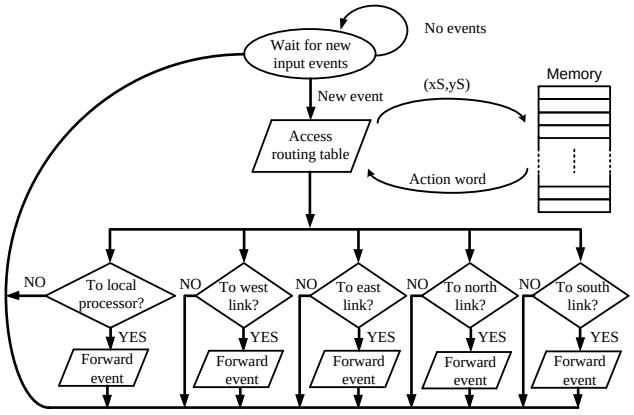


Fig. 8. Source-driven routing algorithm.

B. Source-Driven Routing Algorithm

In the source-driven option, the router receives information about the source address that generated the event. Hence, all the nodes must store information about all the possible source addresses that they can receive. When a new event is received, the router searches for the source address in a local user configurable connection memory. This memory codes all the operations that should be performed when a source address is received: forwarding the event to one or more output ports, routing it to the local processor, or both at the same time. The input routing algorithm for the source-driven solution is shown in Fig. 8.

The user can program connectivity maps by setting the elements of the connection memory. Each node locally stores its connection memory with its own routing actions for all the possible source addresses. Each position of this memory corresponds to each of the possible 8 bit source addresses and stores a 5-bit digital word. When one of those bits is at high level, it indicates that the event must be transmitted through the interface associated with that bit position.

This feature allows replication of events at intermediate network nodes. This is done by activating several bits in the connection memory position of the received source address. The event will be transmitted through the selected output interfaces. The programmed tasks can be performed in parallel because they do not require any shared resources. Output virtual connections from the same node can share the same initial segments of a route and clone events closer to the destination nodes, as opposed to the destination-driven case where the full route is cloned.

Local processor output event stream management is greatly simplified in the source-driven option. The source address is added to all the events that must be transmitted through the network. The only configuration parameter is the output port or ports that must transmit this information to get their final destination. By sending the same event through different output ports the user can balance the network traffic load and improve overall latency, because this reduces the event rate on critical physical links which may otherwise get saturated.

C. Comparison between both algorithms

Any node interconnection map can be implemented using either of the proposed routing algorithms. However, each solution offers certain advantages with respect to the other in terms of connectivity features. We will focus the different algorithms impact on parameters such as latency, network event traffic (the number of events transmitted through the network) and the hardware resources needed for the router implementation.

In terms of hardware complexity, the source-driven implementation needs a more sophisticated routing algorithm. This increased complexity leads to longer delays in the router event processing, increasing the latency associated with event transmission. The destination-driven router takes this decision on the fly, taking into account only the node address and the information contained in the incoming event. The latency penalty caused by the source-driven router is strongly dependent on the shared connection memory implementation and its arbitration mechanism. This memory block is large and results in an important area overhead.

The source driven algorithm provides the system designer with more freedom to balance event traffic and design routes through the networks. For any source-to-destination route, the designer can insert detours, de-branchings, and local event clonings at any intermediate node of the route to balance and optimize overall traffic. On the other hand, the destination driven algorithm creates pre-determined routes along the network, and the designer can only change the output ports of the source module of a route. Also, for the destination-driven case, the events that have to reach several modules necessarily have to be cloned at the source module. However, in the source-driven case, multiple module destination events can be cloned at intermediate route points. This alleviates overall traffic and makes the average effective module fanout (F_{Mout} in eq. (10)) smaller.

D. Deadlock in NoC type systems

Deadlock in Network-on-chip (NoC) or Network-on-Board (NoB) mesh-type systems is an issue of primary concern among researchers and developers. A deadlock situation can happen if routing paths form closed loops [48]. The result is that all sender ports in the loop are requesting to send an event, while at the same time all receiver ports in the loop cannot acknowledge because they cannot take a new event before their corresponding sender port drops an event. This is a very well known and studied problem in mesh type communicating structures. In order to avoid such situations one solution is to route the paths in such a way that no closed loops are formed. In the example systems we provide in this paper we did not encounter any deadlock situation because the ConvNet examples provided are all feed-forward systems. However, in general one may encounter situations where feedback paths need to be implemented, thus increasing chances of forming closed loops. Consequently, when assigning routing paths and output ports within the routers, care must be taken to avoid closed loops, thus eliminating the possibilities for

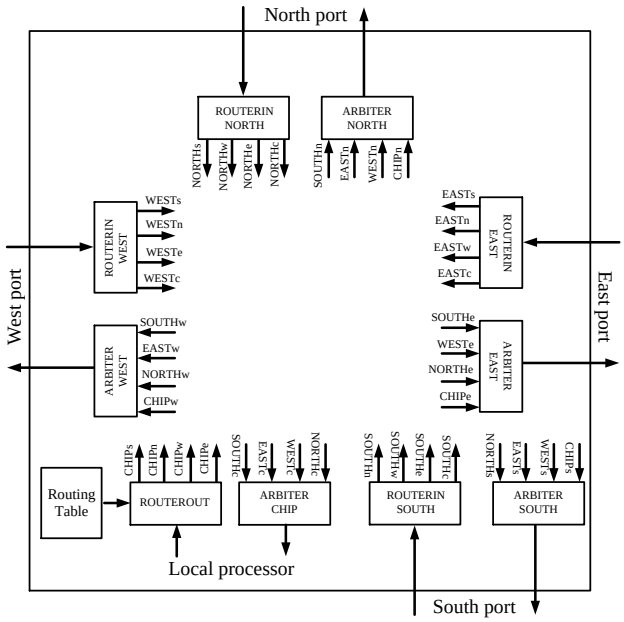


Fig. 9. Destination-driven router block diagram.

deadlock situations. Since the routers we are using have bi-directional data paths between nearest neighbours and events can be routed to four output ports per module, there is a lot of flexibility for avoiding closed loops.

IV. Router Design Details

The routing algorithms described above should be implemented efficiently and with a minimum hardware cost. This Section describes hardware implementations focusing on design issues that must be faced to reduce routing processing times, while keeping lightweight implementations.

A. Destination-Driven Router

Fig. 9 shows a block diagram of the destination-driven router circuit. The traffic through any of the input channels is processed by a *ROUTERIN* block that implements the destination-driven routing algorithm. A highly parallel hardware architecture has been chosen to reduce routing processing times. The goal is to separate the event streams that do not need to use the same channel. For example, a stream that is being transmitted from the west to the east port, never interferes with another stream that is being transmitted from the north to the south port. This is only possible if the shared routing resources are reduced to a minimum by replicating them in the architecture.

Routing is defined by user provided parameters. The first parameter is the node address which is used to identify the node in the network topology. The second parameter is the routing table that contains the information needed to communicate with the node's target destinations. Every entry in this routing table is a 10 bit word in which the 8 least significant bits are used to code the destination address and the 2 most significant bits are used to specify the corresponding output ports (as in Fig. 6).

The basic building blocks of the destination-driven router architecture are:

- **ROUTERIN:** this block receives the input stream coming from an input channel and implements the routing algorithm described in Fig. 5. It decides the output interface to which the event is to be forwarded, by comparing the input event address (*xADD*, *yADD*) with the user-specified local node address (*xNODE*, *yNODE*). The AER handshaking protocol is also used to transfer events between different blocks internally. Handshaking is used here for flow control purposes as the individual processor operation is stopped when a network communication link cannot transmit events. In these overflow situations, the router does not send the acknowledge back until there are hardware resources free to process the event. Each *ROUTERIN* block has four independent output interfaces connected to the output channel access arbiters or to the local processor arbiter.
- **ARBITER:** this block manages access to an output channel for the events coming from the *ROUTERIN* blocks or from the local processor. The *ARBITER* scans its four input AER interfaces to detect any new event. If an event is detected, it takes control of the output channel. If another event from any other interface arrives while the output channel is busy, the resource is not assigned until the current event releases the shared resource. If several input interfaces want to take control of the shared resource at the same time and the block is busy transmitting an event, the *ARBITER* gives lower priority to the last interface attended. Therefore, the arbitration mechanism prevents a fast input interface from monopolizing the shared resource. Note that all arbiters are synchronous circuits driven by the common FPGA clock.
- **ROUTEROUT:** this block implements the algorithm described in Fig. 7 to manage events coming from the local processor. This block reads the routing table row by row to add the proper header to the event, forwarding it through the specified output interface. For this purpose, the *ROUTEROUT* block has four output interfaces connected to the north, south, east and west arbiters. To improve router parallelism, the routing table is divided for every output port. This way, if an event has to be transmitted through different output ports, this can be done in parallel.

B. Source-Driven Router

Fig. 10 shows the block diagram of the source-driven router. The building blocks used are very similar to those described for the destination-driven router. However, the *ROUTERIN* and *ROUTEROUT* blocks have very different internal structures due to the differences in the routing algorithm. In the source-driven option, all the blocks have to read a shared connection memory containing the routing information. The implementation of an efficient shared access scheme for this memory can dramatically reduce the delay time associated with the routing event process. The router contains the following blocks:

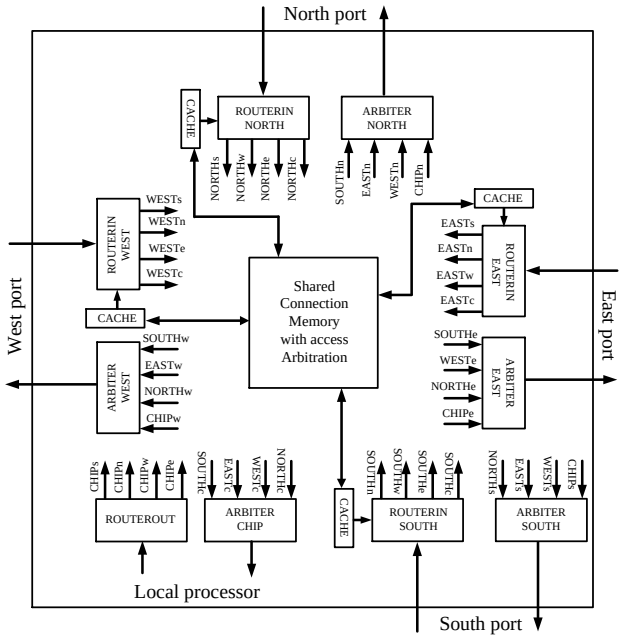


Fig. 10. Source-driven router block diagram.

- **ROUTERIN**: this block analyzes events coming from the input channel to extract the source address. This address is used to access a shared RAM connection memory of 256 positions containing the routing algorithm information. The word read from the connection memory codes the output port(s) that the event must be routed to. When this task is done, the behavior of this block is exactly the same as in the destination-driven router.
- **Local Cache:** every event that is routed in the source-driven solution needs a memory access and all the *ROUTERIN* blocks can read the memory at the same time. However, in a real network each input block will process a limited number of flows. This means that the addresses that every *ROUTERIN* block is going to read will be repeated very often and will only constitute a small part of the total address space. To speed up this process, the *ROUTERIN* block first consults a dedicated cache memory that stores the most common accessed address content. When the routing algorithm needs to access a shared memory position, the *ROUTERIN* block checks the cache memory and searches for this word. If it is found in the cache, the event is routed and there is no need to access the shared RAM. If the word is not found, the block reads the shared RAM and stores the word on its own cache.
- **ROUTEROUT** is greatly simplified because neither an event has to be replicated nor the same header has to be added to all the events. In this case, every time an event coming from the local processor is detected, the router node address is added and the event is sent to the corresponding output port(s).
- **Shared Connection Memory:** this block stores the configuration words that code the routing actions to be taken for each possible source address. All the *ROUTERIN* blocks need to read this memory when they receive a source address that they have not previously stored in their local cache. An arbitration mechanism is needed to avoid conflicts when several *ROUTERIN* blocks need to access the shared memory. This mechanism ensures that only one *ROUTERIN* block has control over the address bus of the shared memory.

C. FPGA Implementations Comparison

In order to compare the two router implementations in a multi-module system, we analyzed the impact of implementing different size networks on a Virtex-6 FPGA prototyping system. As unit-module we used a VHDL description of an event-driven programmable-kernel 2D AER-Convolution-processor for vision applications, capable of handling programmable kernels of size up to 11×11 on pixel arrays of size 64×64 [51]. Each VHDL ConvModule uses register RAM to store pixel states and kernel values. For each pixel state and each kernel weight we use an 8-bit register. The Virtex-6 could hold up to 64 of these Convolution modules, programmed with any arbitrary interconnection map. Fig. 11 illustrates the case of a 3×3 Convolution network. Inputs to the network were provided through 3 input ports, connected to an AER splitter receiving a unique external input AER flow and replicating it over the three inputs. Every network node included one of the previously described routers, the convolution block, and a dedicated configuration processor with an SPI. This interface was fed by a global configuration controller that received all the configuration data (like router tables, and convolution processor parameters) from a host computer. The rest of peripheral modules could connect one of their AER outputs to a multiplexer block connected in turn to the external FPGA. This way, such peripheral modules were able to send their outputs to any of the multiplexer inputs to allow external monitoring of the AER flow. Fig. 12 shows the FPGA occupation ratios for different network sizes (where the number of nodes is $N_n \times N_n$) in terms of occupied slices and memory resources. The destination-driven routing solution needed less hardware resources in terms of memory and slices than the source-driven implementation. Note that a 7×7 grid of these Convolution modules implemented a neural network with $7 \times 7 \times 64 \times 64 = 196k$ neurons (for $k = 1024$) and an equivalent number of $196k \times 11 \times 11 = 24.3$ million synapses. Since each convolution module only had to store one kernel of 11×11 8-bit words, the total physical RAM memory needed to store 7×7 kernels was 5929 bytes. The RAM required to hold all 8-bit neural states was 196KB. Consequently, the limiting factor in this particular case was the number of slices available in the FPGA and not its memory.

V. Network Extension to Multiple FPGAs

The examples illustrated in Section IV were synthesized on a single Virtex-6 FPGA. To allow modular scalability to arbitrary size multi-module networks, provisions for multi-FPGAs (or multi-chips, in case of ASICs) needed to be made. AER links inside the FPGA were made using parallel

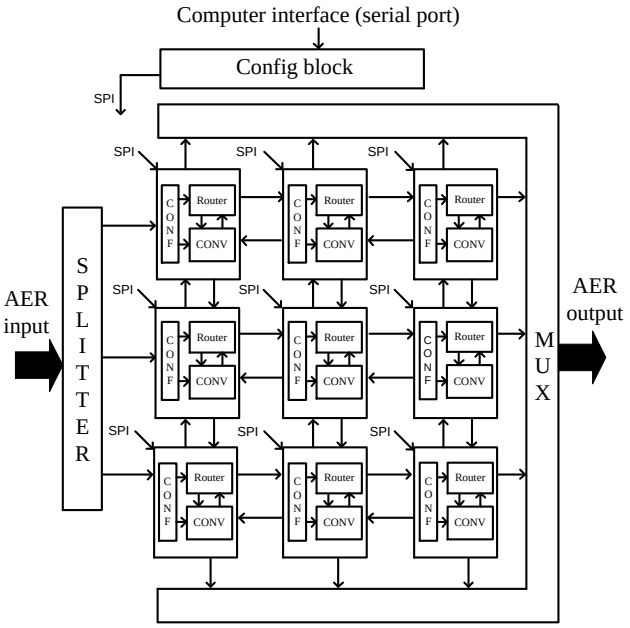


Fig. 11. Block diagram for the network on chip implementation.

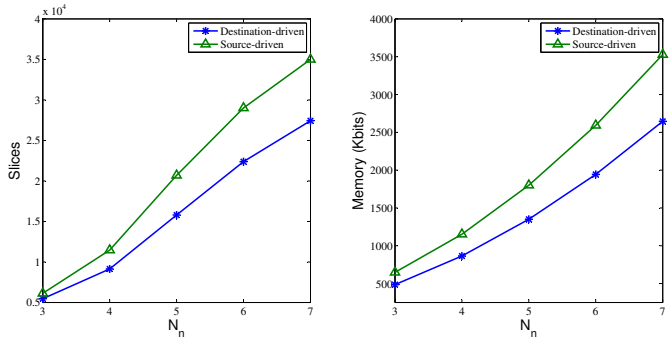


Fig. 12. Slice and memory occupation in an $N_n \times N_n$ node ConvNet implementation. The LX240 Virtex-6 FPGA used in the experiments have 37680 slices and 14976 Kb of block RAM.

AER buses. However, this was not realistic for a multi-FPGA realization because of the excessive number of resulting pins. Fortunately, high-end FPGAs include state-of-the-art serial links, like the Rocket I/O. In this Section we describe a way of extending each asynchronous bidirectional AER link to use available Rocket I/O serial interfaces, using neither dedicated handshaking lines (such as Ack and $Rqst$), nor an extra LVDS pair [38]. When the event rate transmitted in one direction exceeds the processing speed of the receiving module, a stop command is transmitted in the opposite direction. This way, flow control is implemented in both directions just by using the two required LVDS cable pairs for bidirectional serial transmission. For this purpose we used the 8b/10b encoding scheme, as this allows for 12 special characters, commonly called K-characters. We used these characters to implement idle commas (to keep the link synchronized during the absence of address events) and flow control commands. Note that this is standard industry practice.

Fig. 13 shows the full duplex serial Rocket I/O AER link

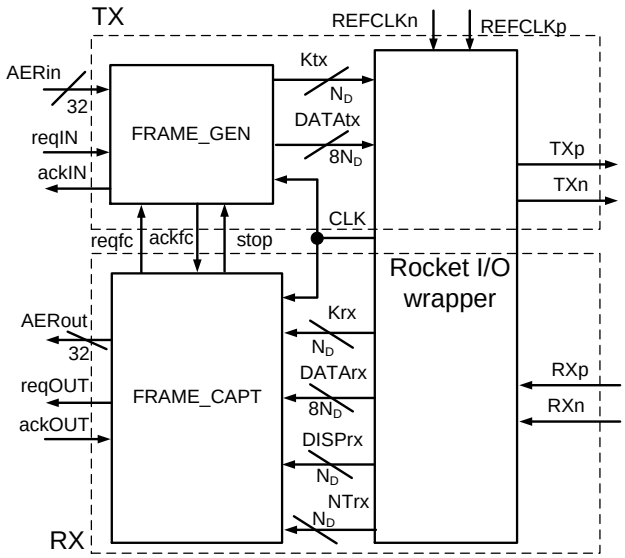


Fig. 13. Full-duplex Rocket I/O based parallel-serial AER link with flow control capability.

with flow control capability. The Xilinx CORE Generator tool provides a wrapper to interface with the FPGA dedicated hardware. It defines the signals that the user must generate in order to send and receive data over this link. The data width interface, N_D in Fig. 13, is user-configurable in 8, 16 or 32 bits. *TXp-TXn* and *RXp-RXn* represent the serial output interface and *REFCLKp* and *REFCLKn* the reference clock used by the Rocket I/O circuitry to generate the transmission frequency.

The *FRAME_GEN* block handshakes the input parallel AER stream and sends $8N_D$ bits at every rising edge of the master clock CLK provided by the Rocket I/O circuitry. If there is no user data to transmit, the interface sends a comma character represented by a K-character of the 8b/10b code. *FRAME_CAPT* receives the continuous data stream coming out from the channel, analyzes it, discards the commas and frames the parallel AER events, implementing the handshaking with the next processing block. Signals *Ktx* and *Krx* are activated when a K-character is transmitted or received, respectively. The receiver also uses the information provided by signals *DISPrx* and *NTrx* to detect possible transmission errors. *DISPrx* indicates a disparity error in the 8b/10b words received and *NTrx* is actived when the received word is not a valid code character.

Fig. 14 illustrates the full-duplex flow control mechanism implemented through signals *reqfc*, *ackfc* and *stop* in Fig. 13. The figure illustrates the case of ROUTER1 sending events to ROUTER2. The ROUTER2 RX2 *FRAME_CAPT* block (see Fig. 13) writes each incoming event in a FIFO memory, which is read by the router when there are new events to be processed. If the ROUTER1 TX1 transmission event rate is faster than the ROUTER2 RX2 handling capabilities, the number of elements stored in the FIFO will increase. If ROUTER1 TX1 keeps sending new events, the FIFO would overflow and information would be lost. The *FRAME_CAPT*

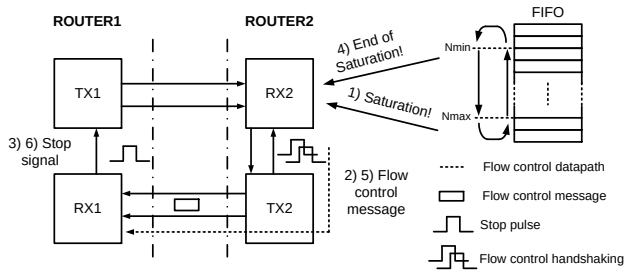


Fig. 14. Flow control mechanism in a full-duplex link.

block detects when the number of elements is greater than a user-defined threshold N_{max} (see '1)' in Fig. 14) and sends a flow control message using a K-character via the ROUTER2 transmitting channel TX2 (see '2)' in Fig. 14). This message request is made by activating *reqfc* and it will be processed with the highest priority by the ROUTER2 TX2 *FRAME_GEN* block. When the flow control message is sent, the ROUTER2 TX2 *FRAME_GEN* block acknowledges the transmission using *ackfc*.

When the ROUTER1 RX1 *FRAME_CAPT* block detects the flow control K-character, it automatically stops event transmission, asserting signal *stop* (see '3)' in Fig. 14). In an overflow situation, the AER acknowledge signal *ackIN* is not activated even when the request signal is asserted and the AER data flow is stopped. The ROUTER2 RX2 *FRAME_CAPT* block monitors the receiving FIFO until the number of elements falls below a second user-defined threshold N_{min} (see '4)' in Fig. 14). From this point on, the overflow situation is considered finished and the flow control message is sent to ROUTER1 RX1 (see '5)' in Fig. 14). When it is received, the *stop* signal is deasserted (see '6)' in Fig. 14) and the transmission flow is resumed.

The ROUTER1 sender keeps transmitting events while the flow control mechanism is in operation. To ensure that no events are lost during traffic peaks, the time needed to stop the transmitter when an overflow situation is detected must fulfill the inequality

$$T_{DET1} + T_{PROP} + T_{STOP} \leq (N_F - N_{max})\, T_{TX,EV} \quad (11)$$

where T_{DET1} is the time needed to detect the overflow situation and generate the flow control message, T_{PROP} is the channel propagation time and T_{STOP} is the time required to stop the transmitter. The maximum number of FIFO elements is represented by N_F and $T_{TX,EV}$ is the transmission event period that is causing the overflow.

It would be desirable for the transmission to begin as soon as possible after a flow control pause. For this purpose, the flow control mechanism has to ensure that there will be events stored in the FIFO waiting to be processed by the router after the recovery. To maximize the receiver event rate, we can impose the following restriction on the N_{min} value

$$T_{DET2} + T_{PROP} + T_{START} \leq N_{min} T_{RX,EV} \quad (12)$$

where T_{DET2} is the time needed to detect the end of an overflow situation, T_{START} is the time required to start the transmitter again and $T_{RX,EV}$ corresponds to the receiver event processing time (or the inverse of the event rate).

VI. System level design considerations

Several analysis and optimization methodologies for 2D mesh connected networks have been proposed in literature [52], [53]. The 2D communication layer is analyzed from a traffic management perspective using queuing theory to find network parameters such as latency, queue delays or queue occupation rates. All these parameters are strongly influenced by the application, because they depend on the network topology (physical and logical) and the traffic rates generated by the processors. On the other hand, optimization procedures for massively parallel architectures [54] have been successfully applied to neuromorphic systems which integrate millions of neurons [9]. Here we will rely on analysis techniques for 2D mesh networks, and use the results to suggest ways to optimize the implementations. The study will be centered on our convolution unit network implementations, but it can be easily extended to other neuron array schemes. We will analyze two points of view: hardware resource requirements and event traffic.

A. Hardware Resource Requirements

AER convolution modules and network circuits employ a certain amount of resources (logic area and memory) which must fit within the selected implementation platform. This resource consumption will be related to the number of AER modules and the neuron array sizes. In general, an N_{units} AER system requires an area of

$$A_{total} = N_{units}\,(A_{conv} + A_{router}) + A_{prog} \quad (13)$$

where A_{conv}, A_{router}, and A_{prog} are the areas used by one convolution processor, one router, and the overall SPI programming circuitry. Let every convolution module have a maximum kernel size of $N_{ker} \times N_{ker}$ and every weight be coded with W bytes. If convolution modules integrate an array of $N_{arr} \times N_{arr}$ neurons the state of which is represented by S bytes, the area taken up by a ConvModule is given by

$$A_{conv} = N_{arr}^2 A_{reg}(S) + N_{ker}^2 A_{reg}(W) + A_{logic} \quad (14)$$

where $A_{reg}(x)$ is the area of a register of x bytes, and A_{logic} is the area taken up by the additional logic of the convolution module implementation, which depends on N_{arr}, W and S.

For the routers, as we discussed previously, the simplicity of the destination driven algorithm needs less logic resources $A_{logic,dest}$ than the source driven $A_{logic,sour}$. Furthermore, the source driven approach needs extra memory to store the routing actions for all possible source addresses in the network (coded in 5 bits). This extra memory is implemented through local registers and uses an area $A_{reg}(x)$, where x is number of bytes. Since routing actions only require 5 bits, $x = (5/8)y$ where y is the number of memory positions. If N_{add} bits are used to code the network addresses, the router areas can be expressed as

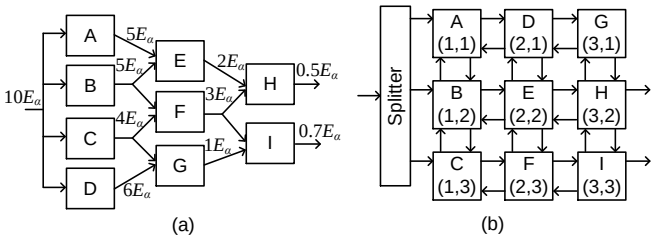


Fig. 15. Example AER system to study system level characterization methodology. (a) Logical description of the network and event rates for each logical (virtual) channel. Parameter E_α has units of event rate (events per second) and the traffic loads at the different nodes are expressed in terms of E_α. By sweeping E_α we can analyze the impact of traffic scaling and saturation in the network. (b) Physical implementation on the structured-grid-AER infrastructure and corresponding mapping of the logical modules (A to I) on the physical 2D grid.

$$A_{router,dest} = A_{rlogic,dest} \tag{15}$$

$$A_{router,sour} = A_{rlogic,sour} + A_{reg}(\frac{5}{8} \times 2^{N_{add}}) \tag{16}$$

The total number of neurons which can be integrated in the system is $N_{units}N_{arr}^2$ and the maximum number of kernel weights is $N_{units}N_{ker}^2$. Taking into consideration the previous resource analysis, the total RAM memory bytes M needed for both routing algorithms can be written as

$$M_{dest} = N_{units}\left(N_{ker}^2 W + N_{arr}^2 S\right) \tag{17}$$

$$M_{sour} = N_{units}\left(N_{ker}^2 W + N_{arr}^2 S + \frac{5}{8} \times 2^{N_{add}}\right) \tag{18}$$

B. Event Traffic Estimation

One of the most important parameters of any AER communication scheme is the event transmission latency between processing blocks. The AER mesh architecture used in this paper can be studied using an analytical model for NoC performance analysis [52], where routers are modeled as a collection of FIFO buffers with five input/output channels (north, south, east, west and local interfaces). This model computes the network queue occupation at every interface of each router assuming that event rates for all network channels are known parameters. These rates are strongly dependent on the specific application and can be easily estimated through a behavioral level simulation.

For example, Fig. 15(a) shows the logic network topology of a specific pre-structured AER system. This network can be simulated behaviorally to obtain the average event rates at each connection (virtual channel). The event rates at each connection are expressed in term of a reference event rate E_α, which can be swept to study changing traffic conditions. To map the logic network onto the physical 2D mesh of nodes, the first step is the "placement" of modules, which is illustrated in Fig. 15(b). The second step is to assign a route (or list of nodes) for events going from a source node s to a destination node d. Let us call this list of route nodes Π_{sd}. Each route corresponds to a virtual connection in the logic network. Once the module placement and route lists Π_{sd} are established we know the event rates at the input and output router channels.

Let l_{ijr} be the event rate at input channel i routed to output channel j in router r. For each router, we can define a 5×5 forwarding probability matrix where element f_{ijr} corresponds to the probability that an event which arrives at interface i will leave the router through interface j. These probabilities can be computed for every router in the network as

$$f_{ijr} = \frac{l_{ijr}}{\lambda_{rj}} \quad i,j \in [1,5], \quad r \in [1, N_{unit}] \tag{19}$$
$$\lambda_{rj} = \sum_{k=1}^{5} l_{ikr}$$

In the network the events from different routes have to share common resources to reach their final destination. If two events want to use the same resource, arbiters grant access and make some events wait in their queues until the resource becomes available. The forwarding matrix can be used to compute the contention probabilities c_{ijr} for each router, i.e., the probability that channels i and j compete for the same output, as:

$$c_{ijr} = \sum_{k=1}^{5} f_{ikr} f_{jkr} \quad \forall i \neq j \qquad c_{ij} = 1 \quad \forall i = j \tag{20}$$

The router forwarding matrix $F_r = [f_{rij}]_{5\times5}$ and the contention matrix $C_r = [c_{ijr}]_{5\times5}$ describe the routers' traffic management. It can be demonstrated [52] that the average number of events per queue $N_r = [N_{rj}]_{5\times1}$ at each router can be computed as:

$$N_r = (I - t_r \Lambda_r C_r)^{-1} \Lambda_r \bar{R_r} \tag{21}$$

where scalar t_r is the mean event processing time in the router and $\Lambda_r = [\lambda_{rj}]_{j=r}$ is a diagonal matrix made up of the total event rates through the 5 input interfaces. $\left[\bar{R_r}\right]_{5\times1}$ is the residual time matrix, which represents the amount of time that a new event has to wait in the queue until the event which occupied the shared resource at the moment the new event arrived finishes its processing. Solving eq. (21) and applying Little's theorem [55], we can compute the mean waiting time in channel j of router r as $W_{rj} = N_{rj}/\lambda_{rj}$. This way, the total latency of one node-to-node hop is $N_{rj} + t_r + t_{tx}$, where t_{tx} is the transmission time through the inter-node physical channel. The total latency of an event traveling from source node s to destination node d is therefore

$$L_{sd} = \sum_{(r,j)\in\Pi_{sd}} (W_{rj} + t_r + t_{tx}) \tag{22}$$

This analysis methodology will be applied to the example system of Fig. 15 in Section VIII where the network traffic will be estimated for a source driven and a destination driven

solution. Moreover, we will discuss how to use this analysis procedure to improve network performance by varying some of the implementation parameters. Note that, given a logical network together with virtual connection event rates, it is only necessary to establish a node "placement" and the route lists $\{\Pi_{sd}\}$. The other computations (from eqs. (19) to (22)) are quite straightforward, given parameters t_r, Λ_r, $\bar{R}_r$ and t_{tx}. The result is the route delays $\{L_{sd}\}$ from which the maximum can be identified as its main timing bottleneck

$$L_{max} = max\{L_{sd}\} \tag{23}$$

The designer must then adapt the node "placement" and route lists $\{\Pi_{sd}\}$ to minimize L_{max}.

VII. Experimental Results

In this Section we provide experimental results by implementing the above mentioned concepts on Virtex-6 hardware using Xilinx ML-605 development boards. First we show the characterization results of the Full-Duplex Rocket-I/O-Based AER parallel-serial interface described in Section V. An example of a multi-module AER processing system which consists of an array of Gabor filters implemented on a single FPGA is then described. After that, a second system, implementing a multi-layer feed-forward Convolutional Neural Network on a single FPGA, is described. Next, we check the maximum capacity of a single FPGA, and finally we provide results for a multi-layer feed-forward ConvNet for character recognition.

A. Full-Duplex Parallel-Serial AER Interface

The ML-605 development board provides twenty independent full-duplex Rocket I/O serial ports, eight of which are available through an 8x PCIe connector. We used a dedicated board to adapt this connector to 16 independent SMA (SubMiniature version A coaxial RF connector) pairs, thus making it possible not only independently to test and characterize several transmitters and receivers, but also to interconnect them. Some extra test circuits were added inside the FPGA, such as an event generator and an event consumer/analyzer, both with independent programmable event rates. This allowed us to force overflow situations and test the flow control dynamics, while detecting errors between the sent and received events.

The timing characteristics of the serial link are given by its latency and maximum event rate. To characterize latency, two independent Full-Duplex AER serial links were interconnected (each with two high speed wires), and the delay between the '*reqIN*' (see Fig. 13) of the first one and the '*reqOUT*' of the second one was measured. This latency included the delay introduced by the 8b/10b encoders and decoders, phase alignment buffers, comma detection circuits, etc. To measure this latency, a very low event rate was programmed, so that consecutive events were sufficiently spaced in time. The measured latency was 232ns for a 2.5Gps bit rate, as shown in Fig. 16. The maximum event rate supported by the Rocket I/O could be characterized by analyzing the input AER handshaking (*reqIN*, *ackIN*) cycle duration which is highlighted in Fig. 16 as 20ns. This would result in 50Meps (mega events per second) maximum possible event rate for 32-bit events.

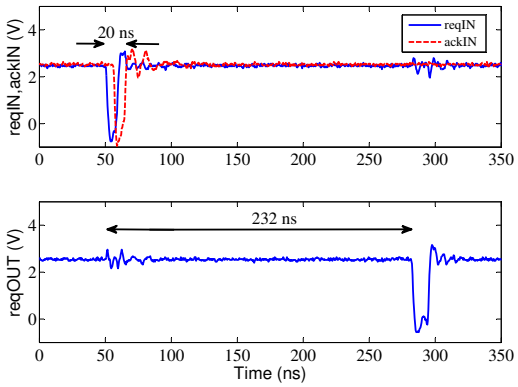


Fig. 16. Interface input (reqIN, ackIN) and output request (reqOUT) when the link operates with very sparse events to measure the input to output event latency.

Fig. 17 illustrates the flow control operation. The event generator in the transmitter was set intentionally to 26Meps event rate, while the event consumer in the receiver was set to have an event consumption rate of 20Meps. This led to overflow in the receiving FIFO. It can be seen how the transmitter was stopped when the flow control message was received and started over again when the overflow situation was overcome. Two different overflow behaviors were possible depending on the chosen N_{max} or N_{min} values. If these were optimally programmed, there was no stop in the output event flow, as shown in Fig. 17(a). On the other hand, pauses appeared if the receiver processed all the events stored in the FIFO before the flow control message arrived at the transmitter. This situation is illustrated in Fig. 17(b).

B. Routers with Parallel-Serial Interfaces

The routers described in Section III were complemented with four Full-Duplex Rocket-I/O-based AER parallel-serial interfaces (see Fig. 13), to test the performance for multi-FPGA event routing. Table III shows the Virtex-6 occupation statistics associated with both implementations. For the destination-driven router, clock frequency could be set up to 250MHz resulting in 2.5Gbps serial bit rate. However, for the source-driven router, clock frequency could only be set up to 200MHz because of the higher complexity, resulting in a 2Gbps bit rate. The latency introduced by the routers can be measured by looking at the delay between the *reqOUT* signal of the input full-duplex Rocket-I/O serial-to-parallel interface and the *reqIN* signal of the output full-duplex Rocket-I/O parallel-to-serial interface. This latency was 12.5ns for the destination-driven router and 20ns for the source-driven router, in non-overflow situations. In the case of the source-driven router the received address was also stored in local cache. Otherwise, the latency became 30ns.

The destination-driven router could handle a maximum 32-bit event rate of $E_{pp} = 27Meps$, which corresponds to 37ns for completion of the handshaking cycle. On the other hand,

TABLE III
ROUTER IMPLEMENTATION STATISTICS FOR THE DD (DESTINATION-DRIVEN) AND THE SD (SOURCE-DRIVEN) ROUTERS.

Resources	DD router	SD router	Total FPGA
Occupied slices	1121	1400	37680
Occupied RAMB18E1 blocks	0	1	832
Rocket I/O transceivers	4	4	20

the source-driven router could handle up to $E_{pp} = 17.5Meps$ maximum event rate, or 57ns handshaking cycle. Although event transmission in destination-driven routing is faster, it is also true that events have to be transmitted multiple times when destinations are multiple. In this case, if the multiple events are routed through the same port, there will be a considerable delay penalty as shown in Fig. 18(a). However, most of the time it is also possible to replicate events through (up to four) different ports, as shown in Fig. 18(b), to avoid this penalty.

C. Single-FPGA Implementation of Gabor Filter Array

As a first illustration of multi-module operation we implemented a 3×3 array of orientation extraction 2D-Gabor filters of different scales and angles on a single Virtex-6. The kernels are shown in Fig. 19. The implemented structure follows the diagram in Fig. 11 and received sensory input from an AER DVS retina [56], [57]. The convolution filters used a modified version of a previously reported VHDL ConvModule [51], with its random Poisson distributed readout mechanism replaced by a plain compare-and-fire mechanism. All filter outputs were routed to one of the multiplexer inputs and captured off-chip with an AER data logger [58]. Fig. 20 shows the sensor and the nine filter output events collected during the same period of 160ms, while the retina was observing two walking persons.

Event-driven convolution processors present the "*pseudo-simultaneity*" property [22], [24]: during a given time interval, the input flow of events (representing the input scene) is simultaneous to the output flow of events (representing the filtered input scene). This is because events are processed as they arrive with delays shorter than the average inter-event time. For the convolution modules we were using, event processing time for 11×11 kernels was about $3\mu s$. The input event flow provided by a 128×128 pixel DVS retina when observing people walking was in the range of 10-50keps (kilo

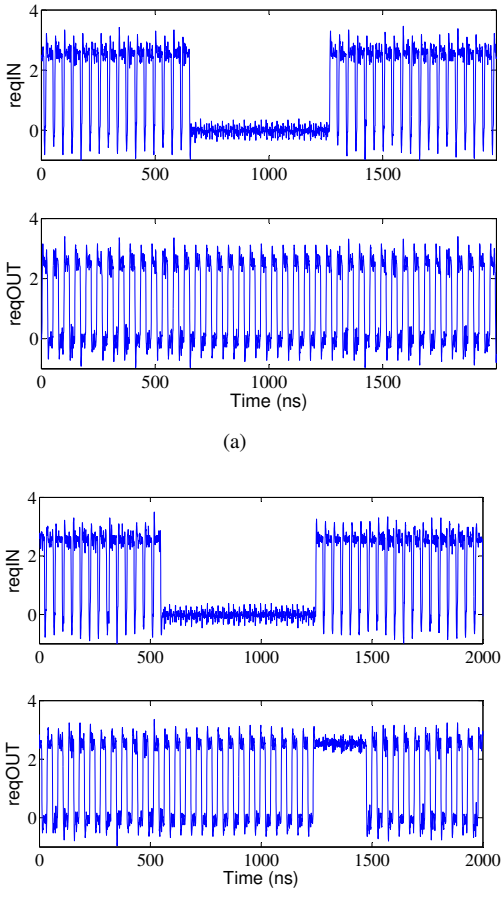


Fig. 17. Interface input (reqIN) and output request (reqOUT) in overflow with (a) optimum N_{max}-N_{min} election and (b) non-optimum N_{max}-N_{min} election leading to pauses in the output flow

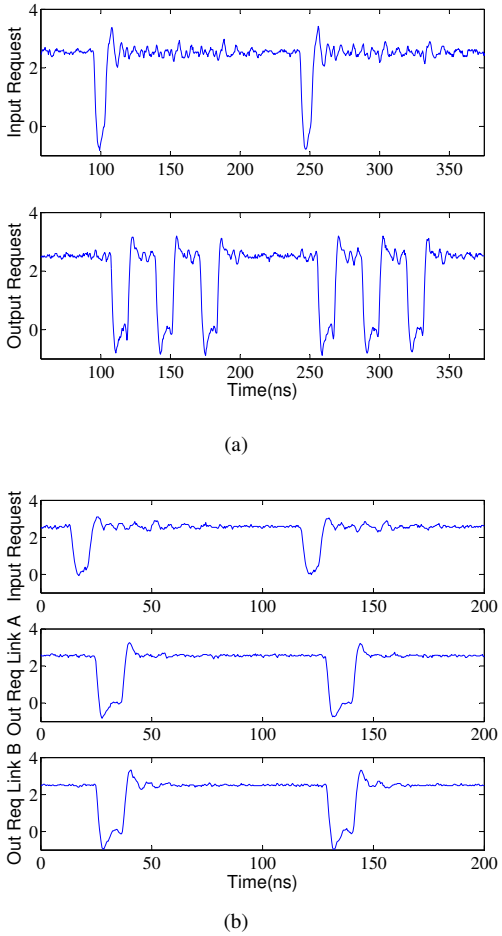


Fig. 18. (a) Output event replica when the same event must be transmitted to different destinations using the same output port. (b) Parallel transmission of events coming from the local processor that must be transmitted through different output ports A and B.

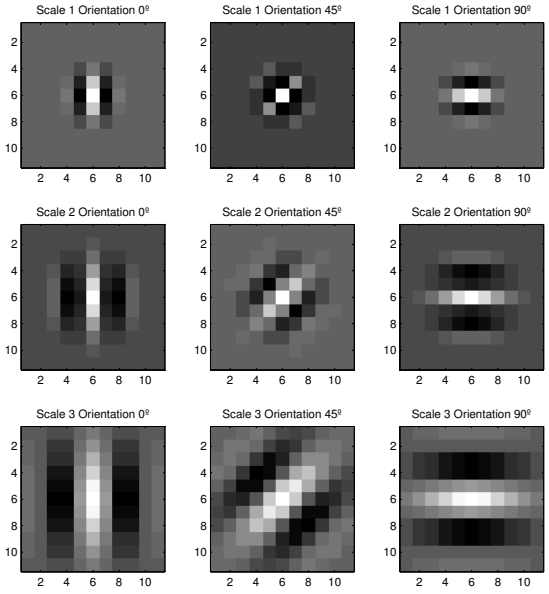
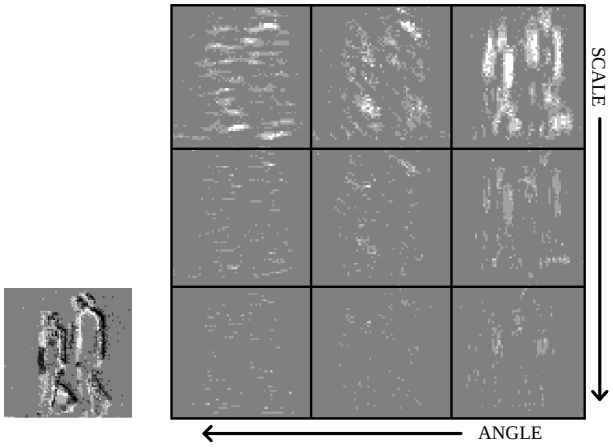


Fig. 19. Kernels in the bank of Gabor filters for the three chosen orientations and scales.

events per second). Consequently, a Gabor filter output event representing an angle at a given scale was available as soon as sufficient input events representing this feature were received, plus the extra $3\mu s$ for processing the last one. This is illustrated in Fig. 21, where a $-45°$ filter provides output events as soon as enough input events are received which are aligned in short $-45°$ edges. Stars represent input events and circles output events. The left most subfigure shows the input and output events collected during 40ms. As can be seen, there are $-45°$ oriented input segments present during these 40ms that are readily detected by output events during these same



(a) (b)

Fig. 20. Event-driven Gabor filtering illustration. Background gray represents zero activity pixels, brighter pixels are active pixels sending positively signed events, darker pixels are active pixels sending negatively signed events. (a) Input scene captured by the DVS retina, with pixel activity of both signs. (b) Results of the 3x3 bank of Gabor filters and sign rectification, so that only positive events result.

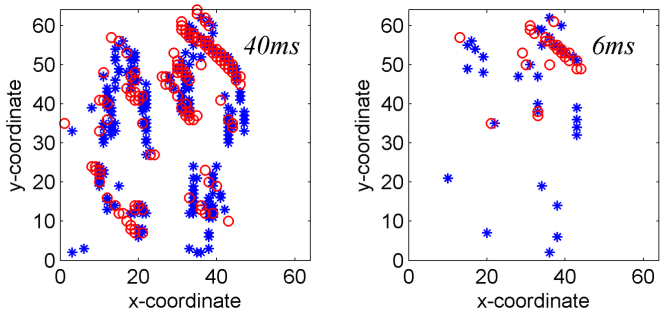


Fig. 21. Illustration of pseudo-simultaneity of event-driven convolutional filtering with a Gabor filter for detection of $-45°$ oriented edges. The two subfigures represent the x/y projection of events captured during 40ms and 6ms. Convolution module input events are represented by stars and output events by circles. One can see that input events representing $-45°$ edges are detected during the same 40ms or even 6ms they appear.

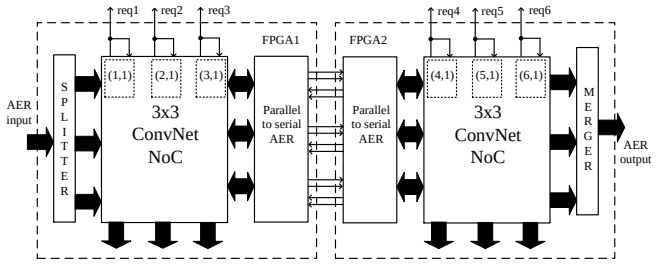


Fig. 22. Diagram of 3x6 Gabor Filter Array Implementation in two FPGAs

40ms. Furthermore, for the right most subfigure we can also see this but for a time interval of only 6ms. Consequently, in event-driven feature extraction, a given feature is detected as soon as enough representative input events are received. Thus, recognition delay would be determined mainly by the event statistics provided by the sensor, not the processing delay of the event-driven ConvNet.

D. Multi-FPGA Implementation of Gabor Filter Array

In order to illustrate the use and operation of the Full-Duplex Rocket-I/O-Based Parallel-Serial Interfaces described in Section V.A, we implemented a 3×6 array of Gabor filters in 2 FPGAs. The corresponding diagram is shown in Fig. 22. The retina events were fed through port 'AER input' and an in-FPGA splitter replicated them on three rows. Node routers were programmed so that the retina events would be copied horizontally from node to node inside each FPGA and also from FPGA1 to FPGA2. To transfer the output events produced by each node (or Gabor filter), the routers were also programmed to copy all output events horizontally from node to node and from FPGA1 to FPGA2. At the right end of all three rows there was an in-FPGA merger block that merged the three flows into a single output port, where an AER data logger board [58] was used to capture and timestamp events. The flow between the two FPGAs was fed through three Full-Duplex Rocket-I/O-Based Parallel-Serial Interfaces.

To analyze the impact of event hopping from node to node (either intra-FPGA or inter-FPGA) we programmed the same Gabor filter into all nodes in the first row. The Gabor filter

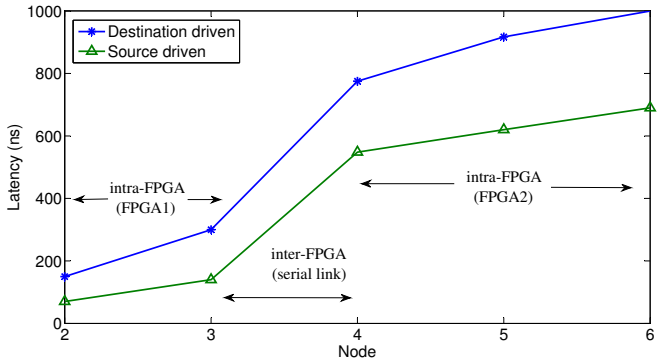


Fig. 23. Latency between nodes for the experiment described in Fig. 22 for the destination and source driven routing algorithms.

output flow was routed to the west and north channels at each node to observe the output and measure the latencies between the first output node $(1,1)$ and the other nodes $(j,1)$ with $j = 2...6$. These latencies could be measured by observing the delays between request signals req_j ($j = 2...6$) and req_1.

The measured delays of req_j with respect to req_1 are shown in Fig. 23, for destination and source driven algorithms. Every in-FPGA network hop added a latency of 150ns for the destination driven algorithm and 70ns for the source driven algorithm. For the inter-FPGA hops an additional 350ns was added to the routing delay, for both routing algorithms. Note that here the source-driven case presents lower latency than the destination-driven case. This is because of the massive retina event cloning for the destination-driven case, because each retina event has to be cloned for each destination Gabor filter. Therefore, event traffic is much higher in the destination-driven case for this particular arrangement.

E. Testing Single-FPGA Maximum Capacity

So far, in each FPGA we had programmed nine 64×64 pixel ConvModules to verify the operation of routers and interfaces. In order to test the maximum capacity of one Virtex-6 FPGA, we checked the maximum of Gabor filters it could hold, together with their routers, peripheral interfaces, SPI configuration circuitry, and input splitter. We used the destination-driven routers, as they are more efficient in terms of FPGA resources. We were able to have the FPGA hold a total of 64 Gabor filters[4], each with 64×64 pixels and kernels of size 11×11. The ConvModule array, together with the 1x8 input splitter circuit, the configuration infrastructure through the serial port and the output channels read-out circuitry occupied 32720 slices, representing 86% of the Virtex6 FPGA capacity. Internal memory occupation was 15% for the 36K RAM blocks and 18% for the 18K RAM blocks. We programmed a Gabor filter array by sweeping four scales and 16 angles. Fig. 24 shows the collected output events for the 64 filters in the same 160ms time window, while the input DVS retina was observing the same two persons walking, shown in Fig. 20(a). Note that, in this case, one single FPGA was emulating a system with $N_{neurons} = 64 \times 64 \times 64 = 2.62 \times 10^5$ neurons and $N_{synapses} = N_{neurons} \times 11 \times 11 = 3.17 \times 10^7$ synapses.

Since the network had a total of $N_l = 216$ inter-module links, each with $E_{pp} = 27Meps$ (as it is destination-driven), the mesh could communicate a total of up to $N_l E_{pp} = 5.8Geps$ of 32-bits each. However, this number needs to be divided by the average number of hops per event n_h and the average number of module fan-out F_{Mout}, to determine the effective (non-cloned) events traveling through the network. Both numbers n_h and F_{Mout} are problem specific, and can be optimized for each case. In this particular case average n_h was around 4, while the average F_{Mout} was about[5] 3. For the example in Section VII-F average n_h is about 3 and average F_{Mout} about 2.

F. Multi-Module Multi-Layer ConvNet Recognition Example

The previously described arrays of Gabor filters represent a one-layer neural system, where all modules (filters) received the same replica from the input sensor. The example illustrated in this Section is a multi-layer Convolutional Neural Network that performs a previously reported character recognition task which has been verified using an AER event-driven simulator with user-defined behaviorally-described event-processing modules [59], [60]. It is loosely based on Fukushima's neocognitron [61] or Serre's hierarchical network [62]. Here we used the same 64×64 convolution module as above (a modified version of the one in [51]) to assemble the 36-node Convolution Neural Network shown in Fig. 25. For this, a 2D-array of 6×6 AER-nodes was synthesized in a single FPGA. The heuristically chosen kernels [59], [60] are illustrated in Fig. 26. Kernels $k1$ to $k13$ performed feature extraction for the 1st layer. Kernels $Ker1$ to $Ker6$ were used for the 15 filters in the second layer. Convolution outputs were always half-wave rectified (events were assigned a positive sign). The layer 2 output virtual channels (labeled 19 to 41 in Fig. 25) were fed to four modules labeled $AGGR_i$ in Fig. 25. These were not ConvModules, but plain arrays of integrate-and-fire neurons. Each $AGGR_i$ module included an AER-merger at its input to merge the traffic from several virtual channels, while forcing their sign bit. For example, module $AGGR_i$ merged events from virtual channels $\{19, 20, 21, 23, 33\}$ and $\{25, 27, 32, 38, 40, 41\}$, while forcing a positive sign bit for the first set and a negative sign bit for the second set. Finally, the 4th layer performed 4 convolutions in parallel, all with the same kernel *KerC* in Fig. 26.

The system was stimulated with bursts of events representing three different versions of letters A, C, H, and M. Bursts had between 200 to 400 events and lasted from about 0.5 to 1ms. Fig. 27 shows the main timing properties in this set-up. An input stimulus burst lasts for a time T_{burst}. At one of the four output recognition channels (nodes '46' to '49' in Fig. 25) the first output event appears at time T_{first} and the output burst lasts until time T_{last}. Table IV summarizes the measured

[4]The corresponding VHDL description is available upon request.

[5]Assuming a $2r$ retina event rate and an r average filter output event rate, the total cloned event rate would be $E_{max} = \overline{n_h}(2r \times 64 + r \times 64)$, while the effective uncloned rate would be $E_{eff} = 2r + 64r$. This results in $E_{max}/E_{eff} = \overline{F_{Mout}}\,\overline{n_h} = 2.91\overline{n_h}$.

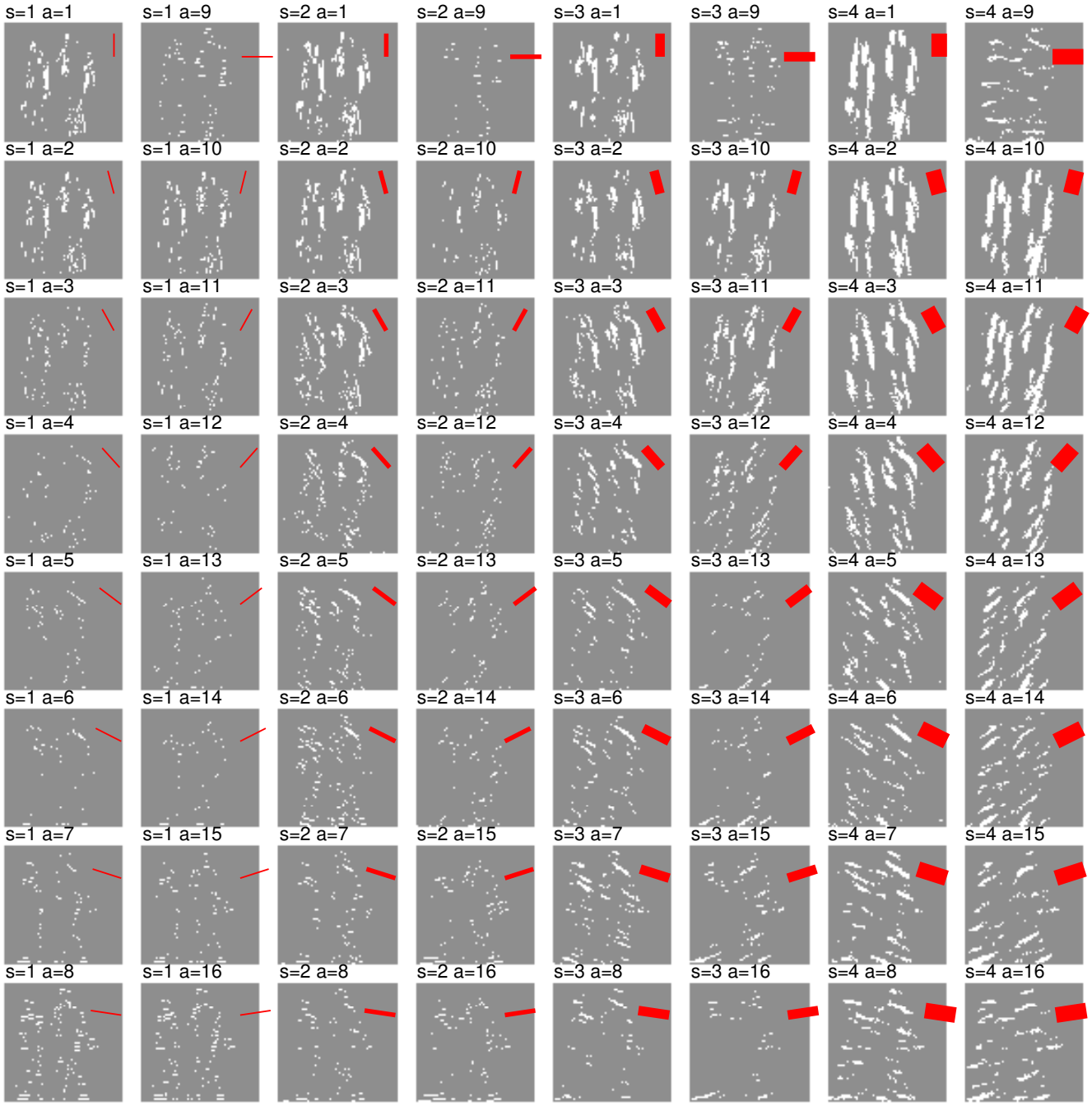


Fig. 24. Output captured from the 64 Gabor filter array. Four scales ($s = 1, ...4$) and 16 angles ($a = 1, ...16$) were swept. Gray scale represents the number of events integrated in every address position in a 160ms temporal window. Background gray is zero output, while bright pixels represent active pixels. The rotated red bar in each subfigure indicates angle and scale (thickness) of the corresponding Gabor filter.

timing results ($T_{burst}, T_{first}, T_{last}$) and also the number of events per output burst for each letter presentation. On average, correct recognition output spikes (which start at time T_{first}) appeared at about half-way through the input stimulus burst $0.5 \times T_{burst}$ and lasted until shortly after the input stimulus burst had finished.

VIII. Discussion

In this Section we will briefly illustrate the system level analysis methodology presented in Section VI with the simple example shown in Fig. 15. This example is not optimized to achieve best performance, but is merely intended to show how we can analyze the network using queuing theory and how this can help us in making design decisions. Fig. 15(a) shows the logical connections (virtual channels) between blocks (nodes) and the event rate at each channel. Average channel

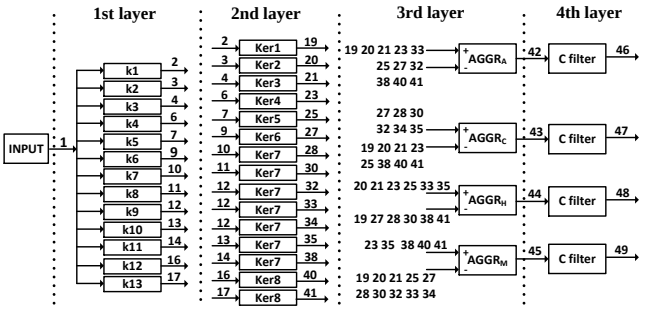


Fig. 25. Logical network topology for the four letter recognition system. Filters ki, $Keri$ and $KerC$ are kernels programmed in the event-driven convolution modules of layers 1, 2, and 4, respectively. Modules $AGGR_i$ are simple integrate-and-fire neurons which count events produced at every address for any of their input channels, producing output events when the count threshold is reached. Node numbers in the figure represent virtual AER channels. Modules $AGGR_i$ include AER-mergers at their inputs which force the sign bit of incoming events.

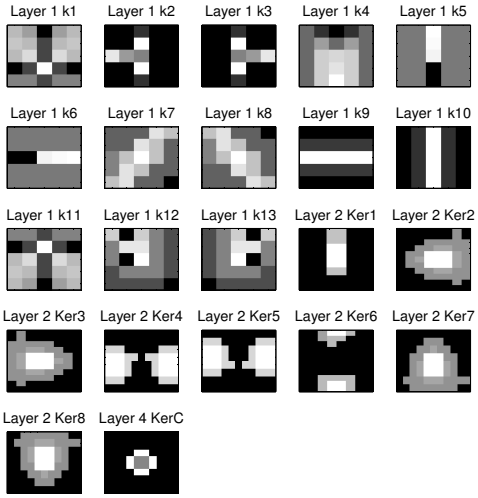


Fig. 26. Kernels used in the letter recognition system through all the network layers.

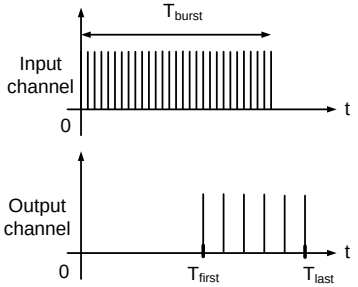


Fig. 27. Timing diagram of the letter recognition process. T_{burst} is the total duration of the input stream representing the input letter. The first output event in the recognition channel appears at instant T_{first}, while the last one is generated at T_{last}.

TABLE IV

TEST RESULTS FOR THE LETTER RECOGNITION SYSTEM.

Input Letter	T_{burst} (μs)	T_{first} (μs)	T_{last} (μs)	# of events A	# of events C	# of events H	# of events M
A1	897	366	941	38	2	0	0
A2	777	589	812	17	0	0	0
A3	867	367	899	39	1	0	0
C1	596	334	619	4	65	0	0
C2	656	373	690	4	87	0	0
C3	476	289	495	5	26	0	0
H1	897	331	928	5	0	68	8
H2	777	460	776	1	0	41	1
H3	746	424	778	1	0	44	1
M1	927	281	885	6	0	5	47
M2	897	584	892	0	0	8	21
M3	837	400	786	0	0	11	28

event rate is expressed in terms of a reference event rate E_α (which has units of events per second) to study traffic under different load conditions. Traffic information can be obtained from behavioral simulations. The logical topology can be mapped into the physical system as depicted in Fig. 15(b). The routing algorithm and tables determine the event routes through the network and this information can be used to estimate the total event rate through each physical channel. Using this information, the forwarding F_r and contention C_r matrices can be computed for each router using eqs. (19) and (20). The model is fed with all these matrices and with certain implementation parameters, such as the routers' service time t_r or the physical channel transmission times t_{tx}. Knowing all these parameters enables us to solve the traffic equation (eq. (21)) to estimate the mean number of packets at each network queue N_r. This number provides information about the network traffic distribution and makes it possible to compute the queues' mean waiting time W_{rj}. By applying eq. (22) to every route it is possible to obtain the latency associated with each route L_{sd}, and also the mean and worst case latency for the whole network.

Fig. 28(a) shows the resulting mean network latency versus reference event rate E_α for the example in Fig. 15. For low event rates, routers are fast enough to process events on the fly and queues are empty most of the time. As a result, mean latency is constant and depends only on the mean number of hops and routing times. In this situation, and for this particular example, the destination driven algorithm presents lower mean latency than the source driven algorithm because the routing algorithm is faster. Mean latency starts to increase exponentially with reference event rate $E_\alpha \approx 8 \times 10^5 eps$ because queues must handle more events and their waiting times rise. For this particular system, the saturation point is almost the same for both routing algorithms. However, Fig. 28(b) shows the same simulation for a 3×3 array of Gabor filters where the input flow must be forwarded to all the nodes in the network. In this case, the network saturates at $E_\alpha \approx 5.8 \times 10^5 eps$ for the destination driven algorithm, while for the source driven algorithm it saturates at $E_\alpha \approx 7.4 \times 10^5 eps$. In this case, source driven routing is more efficient for very high traffic. This is because in the destination driven routing

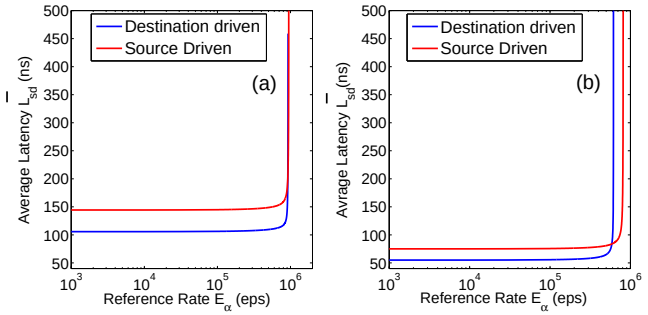


Fig. 28. Mean network latency for different reference event rates E_α for destination and source driven routing algorithms (a) for the example system in Fig. 15 and (b) for a 3×3 array of Gabor filters. Service times were $t_r = 50ns$ for destination-driven routers and $t_r = 70ns$ for source-driven routers.

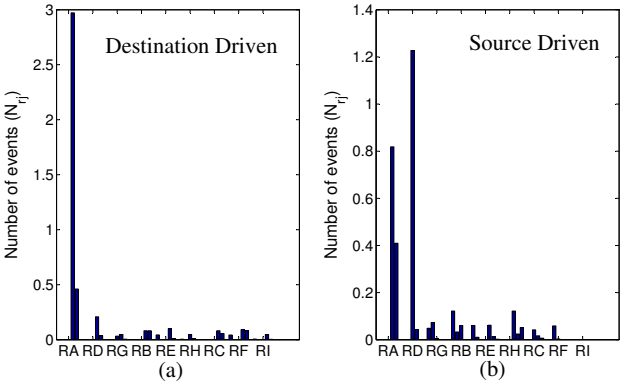


Fig. 29. Events in router queues for a reference event rate $E_\alpha = 9 \times 10^5 eps$. Vertical bars indicate number of events in the order north-east-south-west-local interfaces for every router R_i.

each input event to the network has to be cloned once per destination module, while in the source driven routing each node can perform local replication. In the source driven routing the number of actual events traveling over the physical links is therefore much less for this particular application.

For the example in Fig. 15, the number of events queued in all routers when the network saturates at $E_\alpha = 9 \times 10^5 eps$ can be obtained by solving eq. (21). Fig. 29 represents the 5 input interface queues for each router. This makes it possible to find the network bottlenecks in each case. For the destination driven system, the west interface of router A (RA in Fig. 29(a)) is the most loaded queue. As input events have to be replicated to reach nodes A and D, the input event rate at RA west input interface is artificially increased, overloading this channel. For the source driven algorithm represented in Fig. 29(b), the west interface of routers RA and RD represent the system bottleneck. Again, these two channels are the most overloaded channels due to the multiplexing of several AER streams.

Fig. 30(a) is an example of how traffic analysis methodology can be used to explore the network parameters design space. The traffic simulations were repeated varying the routing time t_r for the destination driven system. For longer service times,

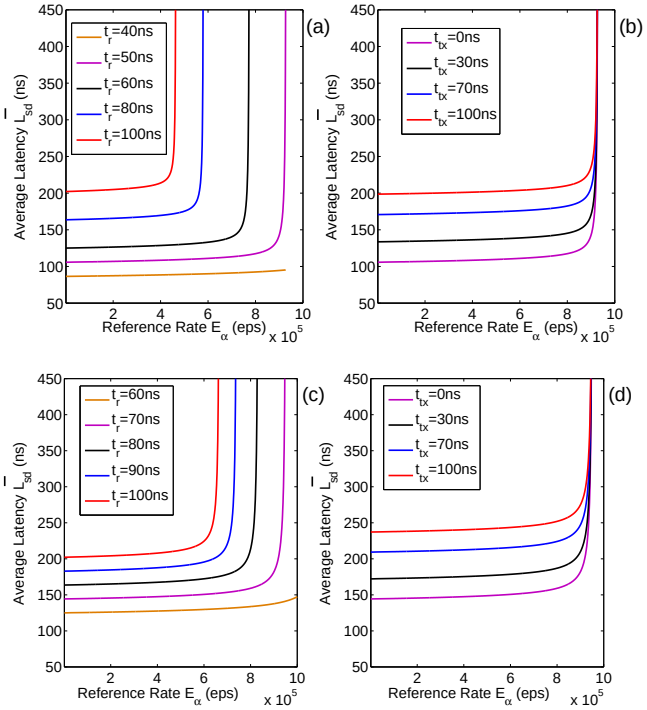


Fig. 30. (a) Latency curves for the destination driven case for different router service times t_r. (b) Latency curves for the destination driven case for different transmission times t_{tx}. (c) Latency curves for the source driven case for different router service times t_r. (d) Latency curves for the source driven case for different transmission times t_{tx}.

mean latency increases because each hop takes longer to route events. Moreover, the network becomes saturated for lower event rates if the routers' service time t_{tx} is increased, reducing the maximum achievable event rate. Fig. 30(b) performs the same simulation, this time varying the transmission time through the channel. In this case, the saturation point remains the same in all simulations, but overall latency increases for low event rates. Figs. 30(c-d) show the same but for source driven routing. As can be seen, the behavior is similar.

IX. Conclusions

We have presented a scalable method for assembling arbitrary modular AER neuromorphic systems, by arranging modules in a 2D mesh. Address events include a module label, and modules include a simple router that routes events depending on their module labels. The approach is generic for ASIC and FPGA based hardware implementations, but was tested on single and multiple Virtex-6 FPGAs. Experimental results are provided for AER-based vision processing applications, such as multiple Gabor filtering and character recognition based on convolutional type neural networks.

The approach is scalable and robust to communication bandwidth saturation. Analysis techniques to estimate resource usage and traffic bottlenecks have been given, which also allow the user to optimize mappings from logical descriptions to the physical implementation. Two routing approaches have been discussed, "destination-driven" and "source-driven". Either one can perform better depending on the traffic conditions

and the network connectivity. Latencies in event communications have been measured to be in the micro second range or below. Implemented test network examples, based on ConvNets, allow for neural systems with over 200,000 neurons, emulating over 32 million synapses (using kernel-based weight-sharing for RAM storage) in a single Virtex6 FPGA. The mesh network uses 4 bi-directional links per module and can communicate up to 5.8×10^9 events per second (excluding module fan-out and event-cloning). This number can be increased by also using diagonal links (resulting in up to 8 links per module), or up to 26 links per module in a 3D arrangement.

The presented approach is intended for the real-time processing of visual sensory data provided by spiking event-driven vision sensors, and does not include any on-line learning capability. Other researchers are developing hardware platforms for emulating fine neural dynamics with synaptic learning capabilities, aimed at studying complex brain functions. For example, the FACETS and BrainScaleS projects (see the summary flyer at [63]) are attempting to put 180,000 analog neurons and 50 million synapses with an average firing rate of 10^5eps per neuron (e.g. with an acceleration factor of about 10^4 with respect to biological time) together on a wafer. The SpiNNaker project is pursuing the development of an ARM-core mesh of processors, using SpiNNaker-chips each with about 20 ARM-cores [31]. A variety of neural and synaptic models can be programmed capable of operating in biological real time. Using realistic but low complexity neural models, an ARM core can handle about 2000 neurons with about 1000 synapses each. Putting 18 SpiNNaker chips on a PCB would, therefore, allow for a system with about 640,000 neurons and 640 million synapses. Other researchers are pursuing similar goals using FPGA based implementations, capable of hosting 1 million neurons in one Virtex-6 FPGA [9].

The results presented in this paper using the Virtex-6 platform, can be extrapolated to ASIC based designs by relying on performance figures from already fabricated ConvModule chips [13], [18], [22], [24]. For example, by re-using an earlier $0.35\mu m$ CMOS ConvChip pixel [24] in a $1cm^2$ 40nm die, it is realistic to consider achieving 1 million neurons per chip. Using a 10×10 mesh of these chips would provide a 10^8 neuron ConvNet, which is comparable (in terms of number of neurons and synapses) to 1% of the human brain.

ACKNOWLEDGMENT

This work was supported in part by Andalusian grant TIC-6091 (NANO-NEURO), and Spanish grants from the Ministry of Economy and Competitivity (former Ministry of Science and Innovation) TEC2009-106039-C04-01/02 (VULCANO) (with support from the European Regional Development Fund) and PRI-PIMCHI-2011-0768 (PNEUMA) coordinated with the European CHIST-ERA program. CZR was supported by an FPU scholarship. The authors would like to acknowledge the highly constructive feedback received from the reviewers, which helped to greatly improve the quality of the paper.

REFERENCES

[1] M. Sivilotti, "Wiring considerations in analog VLSI systems with application to field-programmable networks," *PhD, Computation and Neural Systems, Caltech, Pasadena California*, 1991.

[2] M. Mahowald, "VLSI Analogs of Neuronal Visual Processing: a Synthesis of Form and Function," *PhD, Computation and Neural Systems, Caltech, Pasadena, California*, 1992.

[3] A. Mortara, E. A. Vittoz, and P. Venier, "A Communication Scheme for Analog VLSI Perceptive Systems," *IEEE J. of Solid-State Circuits*, vol. 30, no. 6, pp. 660–669, June 1995.

[4] K. Boahen, "Point-to-Point Connectivity Between Neuromorphic Chips Using Address Events," *IEEE Trans. on Circ. and Syst. Part-II*, vol. 47, no. 5, pp. 416–434, May 2000.

[5] ——, "A Burst-Mode Word-Serial Address-Event Link-I,II,III," *IEEE Trans. on Circ. and Syst. Part-II*, vol. 51, no. 7, pp. 1269–1300, July 2004.

[6] E. Chicca, A. M. Whatley, P. Lichtsteiner, V. Dante, T. Delbruck, P. D. Giudice, R. J. Douglas, and G. Indiveri, "A Multichip Pulse-Based Neuromorphic Infrastructure and Its Application to a Model of Orientation Selectivity," *IEEE Trans. Circ. Syst. Part I*, vol. 54, no. 5, pp. 981–993, May 2007.

[7] R. Vogelstein, U. Mallik, J. Vogelstein, and G. Cauwenberghs, "Dynamically Reconfigurable Silicon Array of Spiking Neurons with Conductance-Based Synapses," *IEEE Trans. Neural Networks*, vol. 18, no. 1, pp. 253–265, Jan. 2007.

[8] C. Mayr, H. Eisenreich, S. Henker, and R. Schüffny, "Pulsed Multi-Layered Image Filtering: A VLSI Implementation," *International Journal of Applied Mathematics and Computer Sciences*, vol. 1, pp. 60–65, 2005.

[9] A. Cassidy, A. Andreou, and J. Georgiou, "Design of a one million neuron single fpga neuromorphic system for real-time multimodal scene analysis," *45th Annual Conf. on Inf. Sciences and Systems (CISS)*, pp. 1–6, 23-25 March 2011.

[10] Y. Wang and S.-C. Liu, "Programmable Synaptic Weights for an aVLSI Network of Spiking Neurons," *Proc. of the 2007 IEEE Int. Symp. on Circ. and Syst. (ISCAS)*, pp. 4531–4534, May 2006.

[11] P. Vernier, A. Mortara, X. Arreguit, and E. A. Vittoz, "An Integrated Cortical Layer for Orientation Enhancement," *IEEE J. Solid-State Circ.*, vol. 32, no. 2, pp. 177–186, Feb. 1997.

[12] T. Y. W. Choi, P. Merolla, J. Arthur, K. Boahen, and B. E. Shi, "Neuromorphic Implementation of Orientation Hypercolumns," *IEEE Trans. on Circ. and Systems (Part I)*, vol. 52, no. 6, pp. 1049–1060, June 2005.

[13] 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. Circ. and Syst., Part-I*, vol. 53, no. 12, pp. 2548–2566, Dec. 2006.

[14] V. Chan, C. Jin, and A. van Schaik, "An address-event vision sensor for multiple transient object detection," *IEEE Trans. on Biomedical Circuits and Systems*, pp. 278–288, Dec. 2007.

[15] Z. Fu, T. Delbrück, P. Lichsteiner, and E. Culurciello, "An address-event fall detector for assisted living applications," *IEEE Trans. on Biomedical Circuits and Systems*, pp. 88–96, June 2008.

[16] T. J. Hamilton, C. Jin, A. van Schaik, and J. Tapson, "An active 2-d silicon cochlea," *IEEE Trans. on Biomedical Circuits and Systems*, pp. 30–43, March 2008.

[17] S. A. Bamford, A. F. Murray, and D. J. Willshaw, "Spike-timing-dependent plasticity with weight dependence evoked from physical constraints," *IEEE Trans. on Biomedical Circuits and Systems*, in Press.

[18] R. Serrano-Gotarredona, T. Serrano-Gotarredona, A. Acosta-Jiménez, C. Serrano-Gotarredona, J. Pérez-Carrasco, B. Linares-Barranco, A. Linares-Barranco, G. Jiménez-Moreno, and A. Civit-Ballcels, "On Real-Time AER 2D Convolutions Hardware for Neuromorphic Spike Based Cortical Processing," *IEEE Trans. on Neural Networks*, vol. 19, no. 7, pp. 1196–1219, July 2008.

[19] B. Wen and K. Boahen, "A Silicon Cochlea with Active Coupling," *IEEE Trans. on Biomedical Circuits and Systems*, pp. 444–455, Dec. 2009.

[20] S. Mitra, S. Fusi, and G. Indiveri, "Real-Time Classification of Complex Patterns Using Spike-Based Learning in Neuromorphic VLSI," *IEEE Trans. on Biomedical Circuits and Systems*, pp. 32–42, Feb. 2009.

[21] R. Mill, S. Sheik, G. Indiveri, and S. L. Denham, "A Model of Stimulus-Specific Adaptation in Neuromorphic Analog VLSI," *IEEE Trans. on Biomedical Circuits and Systems*, pp. 413–419, Oct. 2011.

[22] L. Camuñas-Mesa, A. Acosta-Jiménez, C. Zamarreño-Ramos, T. Serrano-Gotarredona, and B. Linares-Barranco, "A 32 × 32 Convolution Processor Chip for Address Event Vision Sensors with 155ns Event Latency and 20Meps Throughput," *IEEE Trans. Circ. and Syst., Part-I*, vol. 58, no. 4, pp. 777–790, April 2011.

[23] D. G. Chen, D. Matolin, A. Bermak, and C. Posch, "Pulse-Modulation Imaging – Review and Performance Analysis," *IEEE Trans. on Biomedical Circuits and Systems*, pp. 64–82, Feb. 2011.

[24] L. Camuñas-Mesa, C. Zamarreño-Ramos, A. Linares-Barranco, A. Acosta-Jiménez, T. Serrano-Gotarredona, and B. Linares-Barranco, "An Event-Driven Multi-Kernel Convolution Processor Module for Event-Driven Vision Sensors," *IEEE J. of Solid-State Circ.*, Feb. 2012.

[25] J. Lin, P. Merolla, J. Arthur, and K. Boahen, "Programmable Connections in Neuromorphic Grids," *Proc. Int. Midwest Symp. on Circ. and Syst. (MWSCAS)*, pp. 80–84, Aug. 2006.

[26] P. Merolla, J. Arthur, B. Shi, and K. Boahen, "Expandable Networks for Neuromorphic Chips," *IEEE Trans. on Circ. and Syst., Part-I*, vol. 54, no. 2, pp. 301–311, Feb. 2007.

[27] S. A. Bamford, A. F. Murray, and D. J. Willshaw, "Large Developing Receptive Fields Using a Distributed and Locally Reprogrammable Address-Event Receiver," *IEEE Trans. Neural Networks*, vol. 21, no. 2, pp. 286–304, Feb. 2010.

[28] R. Serrano-Gotarredona, M. Oster, P. Lichtsteiner, A. Linares-Barranco, R. Paz-Vicente, F. Gómez-Rodríguez, L. Camunas-Mesa, R. Berner, M. Rivas-Perez, T. Delbrück, S.-C. Liu, R. Douglas, P. Häfliger, G. Jiménez-Moreno, A. Ballcels, T. Serrano-Gotarredona, A. Acosta-Jiménez, and B. Linares-Barranco, "CAVIAR: A 45k Neuron, 5M Synapse, 12G Connects/s AER Hardware Sensory-Processing-Learning-Actuating System for High-Speed Visual Object Recognition and Tracking," *IEEE Trans. Neural Networks*, vol. 20, no. 9, pp. 1417–1438, Sept. 2009.

[29] S. Joshi, S. Deiss, M. Arnold, Y. J. Park, and G. Cauwenberghs, "Scalable Event Routing in Hierarchical Neural Array Architecture with Global Synaptic Connectivity," *12th Int. Workshop on Cellular Nanoscale Networks and Their Applications (CNNA)*, Feb. 2010.

[30] M. Khan, D. Lester, L. Plana, A. Rast, X. Jin, E. Painkras, and S. Furber, "SpiNNaker: Mapping Neural Networks onto a Massively-Parallel Chip Multiprocessor," *IEEE Int. Joint Conf. on Neural Networks (IJCNN)*, pp. 2849–2856, June 2008.

[31] A. D. Rast, F. Galluppi, X. Jin, E. Painkras, and S. B. Furber, "The Leaky Integrate-and-Fire Neuron: A Platform for Synaptic Model Exploration on the spiNNaker Chip," *IEEE Int. Joint Conf. on Neural Networks (IJCNN)*, pp. 1–8, June 2010.

[32] J. Fieres, J. Schemmel, and K. Meier, "Realizing Biological Spiking Network Models in a Configurable Wafer-Scale Hardware System," *IEEE Int. Joint Conf. on Neural Networks (IJCNN)*, pp. 969–976, June 2008.

[33] S. Scholze, S. Schiefer, J. Partzsch, S. Hartmann, C. G. Mayr, S. Höppner, H. Eisenreich, S. Henker, B. Vogginger, and R. Schüffny, "VLSI Implementation of a 2.8 Gevent/s Packet Based AER Interface with Routing and Event Sorting Functionality," *Frontiers in Neuroscience (Frontiers in Neuromorphic Engineering)*, vol. 117, no. 5, pp. 1–13, 2011. Doi:10.3389/fnins.2011.00117.

[34] A. Cassidy, T. Murray, A. Andreou, and J. Georgiou, "Evaluating On-Chip Interconnects for Low Operating Frequency Silicon Neuron Arrays," *IEEE Int. Symp. on Circ. and Syst. (ISCAS)*, pp. 2437–2440, May 2011.

[35] D. Gross and C. M. Harris, "Fundamentals of queueing theory," *3rd ed. John Wiley & Sons, Inc*, 1998.

[36] C. Zamarreño-Ramos, T. Serrano-Gotarredona, and B. Linares-Barranco, "An Instant-Startup Jitter-Tolerant Manchester-Encoding Serializer/Deserializar Scheme for Event-Driven Bit-Serial LVDS Inter-Chip AER Links," *IEEE Trans. on Circ. and Syst., Part I*, in Press.

[37] H. Berge and P. Hafliger, "High-Speed Serial AER on FPGA," *Proc. IEEE Int. Symp. on Circ. and Syst. (ISCAS)*, pp. 857–860, May 2007.

[38] D. B. Fasnacht, A. M. Whatley, and G. Indiveri, "A Serial Communication Infrastructure for Multi-Chip Address Event Systems," *Proc. IEEE Int. Symp. on Circ. and Syst. (ISCAS)*, pp. 648–651, May 2008.

[39] L. Benini and G. D. Micheli, "Networks on Chips: A New SoC Paradigm," *IEEE Comput.*, vol. 35, no. 1, pp. 70–78, Jan. 2002.

[40] P. Salihundam, S. Jain, T. Jacob, S. Kumar, V. Erraguntla, Y. Hoskote, S. Vangal, G. Ruhl, and N. Borkar, "A 2 Tb/s 6×4 Mesh Network for a Single-Chip Cloud Computer With DVFS in 45 nm CMOS," *IEEE J. of Solid-State Circuits*, vol. 46, no. 4, pp. 757–766, April 2011.

[41] J. Howard, S. Dighe, S. Vangal, G. Ruhl, N. Borkar, S. Jain, V. Erraguntla, M. Konow, M. Riepen, M. Gries, G. Droege, T. Lund-Larsen, S. Steibl, S. Borkar, V. De, and R. V. D. Wijngaart, "A 48-Core IA-32 Processor in 45 nm CMOS Using On-Die Message-Passing and DVFS for Performance and Power Scaling," *IEEE J. of Solid-State Circuits*, vol. 46, no. 1, pp. 173–183, Jan. 2011.

[42] S. Sarkar, G. Kulkarni, P. Pande, and A. Kalyanaraman, "Network-on-Chip Hardware Accelerators for Biological Sequence Alignment," *IEEE Trans. on Computers*, vol. 59, no. 1, pp. 29–41, Jan. 2010.

[43] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based Learning Applied to Document Recognition," *Proc. of the IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov. 1998.

[44] 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.

[45] 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 Computation*, vol. 1, no. 4, pp. 541–551, 1989.

[46] R. Vaillant, C. Monrocq, and Y. LeCun, "Original Approach for the Localisation of Objects in Images," *IEE Proc on Vision, Image, and Signal Processing*, vol. 141, no. 4, pp. 245–250, August 1994.

[47] M. Osadchy, Y. LeCun, and M. Miller, "Synergistic Face Detection and Pose Estimation with Energy-Based Models," *Journal of Machine Learning Research*, vol. 8, pp. 1197–1215, May 2007.

[48] S. Murali, *Designing Reliable and Efficient Networks on Chips*. Springer, 2009.

[49] C. L. Seitz, W. C. Athas, C. M. Flaig, A. J. Martin, J. Seizovic, C. S. Steele, and W.-K. Su, "The architecture and programming of the Ametek Series 2010 multlcomputer," In *Proc. 3rd Conf. on Hypercube Concametlt Computers and Apphcatzorzs*, Volume I, (Pasadena, Calif., Jan. 19-20). ACM, New York, pp. 33-36, 1988.

[50] Intel Corporation, *A Touchstone DELTA System Description*, 1991.

[51] A. Linares-Barranco, R. Paz-Vicente, F. Gómez-Rodriguez, A. Jiménez, M. R. A., G. Jiménez, and A. Civit, "On the AER Convolution Processors for FPGA," *Proc. IEEE Int. Symp. on Circ. and Syst. (ISCAS)*, pp. 4237–4240, May 2010.

[52] U. Ogras, P. Bogdan, and R. Marculescu, "An Analytical Approach for Network-on-Chip Performance Analysis," *IEEE Tran. on Comp.-Aided Design of Int. Circ. and Syst.*, vol. 29, no. 12, pp. 2001–2013, Dec. 2010.

[53] J. Hu, U. Y. Ogras, and R. Marculescu, "System-Level Buffer Allocation for Application-Specific Networks-on-Chip Router Design," *IEEE Tran. on Comp.-Aided Design of Int. Circ. and Syst.*, vol. 25, no. 12, pp. 2919–2933, Dec. 2006.

[54] A. Cassidy and A. Andreou, "Beyond Amdahl's Law: An Objective Function That Links Multiprocessor Performance Gains to Delay and Energy," *IEEE Transactions on Computers*, 2011.

[55] F. Hillier and G. Lieberman, "Introduction to Operations Research," *6th Ed., New York, McGraw-Hill*, pp. 631–732, 1995.

[56] P. Lichtsteiner, C. Posch, and T.Delbruck, "A 128×128 120dB 30mW Asynchronous Vision Sensor that Responds to Relative Intensity Change," *IEEE J. Solid-State Circuits*, vol. 43, pp. 566–576, Feb. 2008.

[57] J. A. Leñero-Bardallo, T. Serrano-Gotarredona, and B. Linares-Barranco, "A 3.6μs Asynchronous Frame-Free Event-Driven Dynamic-Vision-Sensor," *IEEE J. Solid-State Circuits*, vol. 46, no. 6, pp. 1443–1455, June 2011.

[58] F. Gómez-Rodríguez, R. Paz-Vicente, A. Linares-Barranco, M. Rivas, L. Miró, S. Vicente, G. Jiménez, and A. Civit, "AER Tools for Communications and Debugging," *Proc. IEEE Int. Symp. on Circ. and Syst. (ISCAS)*, pp. 3253–3256, May 2006.

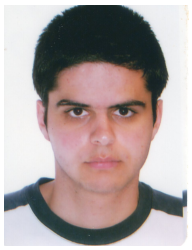
[59] J. Pérez-Carrasco, T. Serrano-Gotarredona, C. Serrano, B. Acha, and B. Linares-Barranco, "High-speed character recognition system based on a complex hierarchical AER architecture," *Proc. IEEE Int. Symp. on Circ. and Syst. (ISCAS)*, pp. 2150–2153, May 2008.

[60] J. Pérez-Carrasco, "Simulation tool for building and analyzing complex and hierarchically structured aer-based visual processing systems," Ph.D. dissertation, Univ. of Sevilla, Spain, 2011.

[61] K. Fukushima and N. Wak, "Handwritten Alphanumeric Character Recognition by the Neocognitron," *IEEE Tran. on Neural Networks*, vol. 2, no. 3, pp. 355–365, May 1991.

[62] T. Serre, L. Wolf, S. Bileschi, M. Riesenhuber, and T. Poggio, "Robust Object Recognition with Cortex-Like Mechanisms," *IEEE Tran. on Pattern Analysis and Machine Intelligence*, vol. 29, no. 3, pp. 411–426, March 2007.

[63] "www.facets-project.org."



Carlos Zamarreño-Ramos received his B. S. degree in Telecommunications Engineering in 2007, his M.Sc. degree in Microelectronics in 2009, and his PhD degree in December 2011 from the University of Seville, Sevilla, Spain. From 2007 until December 2011 he was a PhD student at the Instituto de Microelectrónica de Sevilla (IMSE-CNM-CSIC), Sevilla, Spain. During June-July 2010, he was with the Department of Electrical and Computer Engineering, Texas A&M University, College Station, as a Visiting Scholar. His research interests include very large scale integration (VLSI) circuit design applied to bio-inspired circuits and systems, bio-inspired signal processing, high-speed serial links, modular assembly of reconfigurable AER (Address Event Representation) processing systems, hardware implementations and simulation of spiking neural networks, implementations of event-driven AER vision processing systems, and power management. Since February 2012 he holds the position of Junior Analog Designer at Dialog Semiconductor GmbH, Germering, Germany.

Alejandro Linares-Barranco received the B.S. degree in computer engineering, the M.S. degree in industrial computer science, and the Ph.D. degree in computer science (specializing in computer interfaces for bio-inspired systems) from the University of Sevilla, Sevilla, Spain, in 1998, 2002, and 2003, respectively. From January 1998 to June 1998, he was Second Lieutenant in the Spanish Air Force working as System Administrator and Software Developer. From 1998 to 2000, he was a Member of the Technical Staff at the Sevilla Microelectronics Institute (IMSE-CNM-CSIC). From 2000 to 2001, he was a Development Engineer with the Research and Development Department, at SAINCO Company, Sevilla, working on VHDL-based field-programmable gate array (FPGA) systems for the INSONET European project on power line communications. Since 2001 to 2006, he was an Assistant Professor at the Computer Architecture and Technology Department of the University of Sevilla, Sevilla, Spain. In 2006 he was promoted to Associated Professor. His Lab (Robotics and Computers Technology) developed a set of AER-tools for debugging and connecting AER systems under the EU project CAVIAR. His research interests include VLSI and FPGA digital design, neuro-inspired chip-to-chip and chip-to-computer interfaces, spike based processing, motor control and vision for FPGAs, wireless sensor networks and embedded applications based on microcontrollers, bus emulation, and computer architectures. Since 2009 he has been a Review Committee Member of ISCAS. In 2010 he became member of the Technical Committee on Neural Systems and Applications (NSATC) of the IEEE Circuits and Systems Society. On 2011 he became Secretary of the NSATC.

Teresa Serrano-Gotarredona received the B.S. degree in Electronic Physics and the Ph.D degree in VLSI neural categorizers from the University of Sevilla, Sevilla, Spain, in 1992, and 1996, respectively, and the M.S. degree in Electrical and Computer Engineering from The Johns Hopkins University, Baltimore, MD, in 1997. She was an Assistant Professor in the Electronics and Electromagnetism Department, University of Sevilla from 1998 until September 2000. Since September 2000, she has been a Tenured Scientist at the Sevilla Microelectronics Institute (IMSE-CNM-CSIC), Sevilla, Spain, and in July 2008 she was promoted to Tenured Researcher. Since January 2006, she is also part-time Professor with the University of Sevilla. She was on a sabbatical stay at the Electrical Engineering Dept. of Texas A&M University during Spring 2002. Her research interests include analog circuit design of linear and nonlinear circuits, VLSI neural based pattern recognition systems, VLSI implementations of neural computing and sensory systems, transistor parameters mismatch characterization, Address-Event-Representation (AER) VLSI, RF circuit design, nanoscale memristor-type AER, and Real-Time Vision Sensing and Processing Chips. She is co-author of the book Adaptive Resonance Theory Microchips (Kluwer 1998).

Dr. Serrano-Gotarredona was corecipient of the 1997 IEEE Transactions on VLSI Systems Best Paper Award for the paper "A Real-Time Clustering Microchip Neural Engine". She was also a corecipient of the 2000 IEEE Circuit and Systems-Part I Darlington Award for the paper "A General Translinear Principle for Subthreshold MOS Transistors". She is presently Secretary of the IEEE CAS Society Sensory Systems Technical Committee, Academic Editor of the PLoS ONE Journal, and Associate Editor for IEEE Trans. on Circuits and Systems Part I.

Bernabé Linares-Barranco (F'10) received the B. S. degree in Electronic Physics, the M. S. degree in Microelectronics, and a first Ph.D. degree in high-frequency OTA-C oscillator design in June 1990 from the University of Sevilla, Sevilla, Spain, in 1986, 1987, and 1990, respectively, and a second Ph.D degree in analog neural network design from Texas A&M University, College-Station, USA, in 1991. Since September 1991, he has been a Tenured Scientist at the Sevilla Microelectronics Institute (IMSE-CNM-CSIC), which is one of the institutes of the "National Microelectronics Center" (CNM) of the "Spanish Research Council" (CSIC) of Spain. On January 2003, he was promoted to Tenured Researcher and, in January 2004, to Full Professor of Research. Since March 2004, he has also been part-time Professor with the University of Sevilla. From September 1996 to August 1997, he was on sabbatical stay at the Department of Electrical and Computer Engineering, Johns Hopkins University, Baltimore, MD, as a Postdoctoral Fellow. During Spring 2002, he was a Visiting Associate Professor at the Electrical Engineering Department, Texas A&M University. He is co-author of the book "Adaptive Resonance Theory Microchips" (Kluwer, 1998). He was also the coordinator of the EU-funded CAVIAR project. He has been involved with circuit design for telecommunication circuits, VLSI emulators of biological neurons, VLSI neural based pattern recognition systems, hearing aids, precision circuit design for instrumentation equipment, bio-inspired VLSI vision processing systems, transistor parameters mismatch characterization, Address-Event-Representation VLSI, RF circuit design, real-time AER vision sensing and processing chips, memristor circuits, and extending AER to the nanoscale.

Dr. Linares-Barranco was corecipient of the 1997 IEEE Transactions on VLSI Systems Best Paper Award for the paper "A Real-Time Clustering Microchip Neural Engine", and the 2000 IEEE CAS Darlington Award for the paper "A General Translinear Principle for Subthreshold MOS Transistors". He has organized several Special Sessions and post-conference Workshops for ISCAS and NIPS. From July 1997 to July 1999, he was an Associate Editor of the IEEE Transactions on Circuits and Systems-Part II, and from January 1998 to December 2009 he was an Associate Editor for IEEE Transactions on Neural Networks. He is Associate Editor of Frontiers in Neuromorphic Engineering since May 2010. He was the Chief Guest Editor of the 2003 IEEE Transactions on Neural Networks Special Issue on Neural Hardware Implementations. From June 2009 until May 2011 he was the Chair of the Sensory Systems Technical Committee of the IEEE CAS Society. In March 2011 he became Chair of the IEEE CAS Society Spain Chapter. He is an IEEE Fellow.