Expandable Networks for Neuromorphic Chips

Paul A. Merolla, John V. Arthur, Bertram E. Shi, and Kwabena A. Boahen

***Abstract*— We have developed a grid network that broadcasts spikes (binary events) in a multi-chip neuromorphic system by relaying them from chip to chip. The grid is expandable because, unlike a bus, its capacity does not decrease as more chips are added. The multiple relays do not increase latency because the grid's cycle-time is shorter than the bus. We describe an asynchronous relay implementation that automatically assigns chip-addresses to indicate the source of spikes, encoded as word-serial address-events. This design, which is integrated on each chip, connects neurons at corresponding locations on each of the chips (pointwise connectivity) and supports oblivious, targeted, and excluded delivery of spikes. Results from two chips fabricated in 0.25-μm technology are presented, showing word-rates up to 45.4 M events/s.**

I. Address-event Communication

Neuromorphic engineers attempt to capture the computational power and efficiency of biological neural systems in their hybrid analog–digital VLSI systems. These neuromorphic systems employ a similar design strategy as biology: local computations are performed in analog, and the results are communicated using all-or-none binary events (spikes). A typical neuromorphic chip consists of a 2-D array of bio-inspired circuits (core), which may include models of synapses [1], spatiotemporal filtering [2], and spike generation [3]; the array is surrounded by a transceiver that handles spike communication to and from the core through an off-chip link. Techniques to *morph* neural wetware into VLSI hardware have been described previously for analog circuits [4], [5] and spike-based communication [6].

Neuromorphic chips routinely require tens of thousands of axonal connections to propagate spikes to and from the core — far too many to be implemented using dedicated wires. The **address-event** link, originally introduced by Mahowald and Sivilotti, implements these axonal connections using a time-division-multiple-access (TDMA) link [7], [8]. Transceivers multiplex–demultiplex spikes over a few high-speed wires for efferent–afferent (outgoing–incoming) axons, respectively. For each event, the axon's identity is encoded with a unique binary word, an address event.

Manuscript received December; revised ????. This work was made possible by funding from the Whitaker Foundation, the NSF's BITS and CAREER programs (EIA-0130822, ECS00-93851), and the Hong Kong Research Grants Council (HKUST6200/03E).

P. Merolla and J. Arthur are with the University of Pennsylvania, Dept of Bioengineering, 3320 Smith Walk, Philadelphia, PA 19104-6392 USA (e-mail: {pmerolla,jarthur}@seas.upenn.edu). K. Boahen was also with the University of Pennsylvania at the time of this work, and is currently affiliated with Stanford University (e-mail: boahen@stanford.edu). B. Shi is with the Dept. of Electrical Engineering, Hong Kong University of Science and Technology, Clear Water Bay Kowloon, HONG KONG SAR (email: eebert@ee.ust.hk).

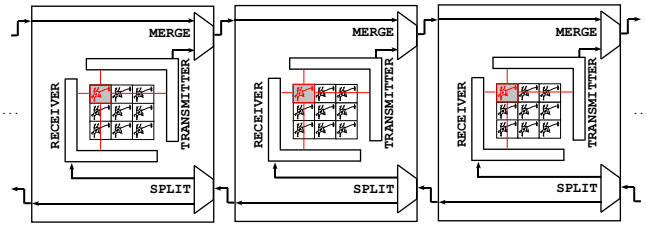


Fig. 1. Spikes are broadcast in a 1-D grid to multiple neuron transceiver arrays. Our grid can implement pointwise all-to-all connectivity (shown), and supports targeted and excluded delivery of spikes.

The address-event link is implemented with asynchronous logic because, like real axons, communication is event-driven, not clock-driven. Event timing is implicitly preserved as long as the link operates near real-time (i.e., all events reach their destination within the timing precision of an axon, which is typically around 1ms). A synchronous implementation is also possible, except that a global clock can couple correlated noise into sensitive analog circuits and artificially synchronize spikes. For these reasons, synchronous communication is avoided in neuromorphic systems.

In this paper, we describe how multiple transceiver chips can be connected together via address-event links. Our system is designed around the following insight: If the connectivity between chips is dense, broadcasting an event is more efficient than targeting it (where copies are routed to specific destinations). For example, in a 1-D n-chip network, routing copies of an event to more than two locations generates more traffic than broadcasting it everywhere, as each copy travels a distance $n/2$ on average, while a broadcast travels a distance n.[1] However, a dedicated bus, which has been used in the past to create multi-chip neuromorphic systems [9], [10], is a poor choice for implementing a broadcast because adding chips decreases the communication capacity of the system (due to the increasing length of the bus).

We present an implementation of a grid network that broadcasts events to corresponding locations on all the chips (pointwise axonal projections) (Fig. 1).[2] The grid (or mesh) architecture, which was first used in the context of neuromorphic systems by [11], implements a broadcast by relaying events between nearest neighbor chips; it offers an expandable solution without sacrificing latency. In addition to pointwise all-to-all connectivity (oblivious delivery), our implementation supports targeted and excluded delivery, and can easily be

[1] This argument is also applicable to multi-dimensional networks. In a 2-D network, where the average distance is $\sqrt{n}$, the threshold is a fanout of $\sqrt{n}$.

[2] Axons are not restricted to terminate on a single neuron in the target layer, but like the cortex, they are able to arborize on a localized region around the target; this fanout is implemented using on-chip charge diffusion networks [5] whose spatial extent can be adjusted independently for each chip.

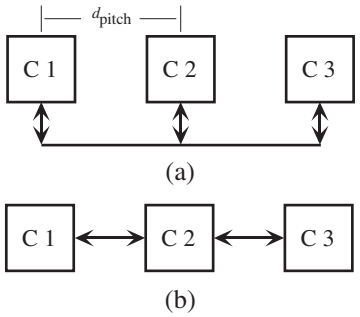


Fig. 2. Three-chip systems that use (a) a bus and (b) a grid.

extended to support arbitrary connectivity with the inclusion of a look-up table (described in Section VII). Communication in our system uses a novel address-event packet protocol that is able to keep track of an event's source without increasing the width of the link.

This paper is segmented into seven sections. In Section II, we compare the bus with the grid. After convincing the reader that grids are expandable and deliver their events on time, we then introduce a communication controller in Section III, called a $\top$, for implementing a grid. Section IV describes the signaling protocols that we use to build a $\top$; we present the implementation in Section V. Section VI presents results from two chips that use the circuits developed in this paper. Finally, Section VII concludes the paper with suggestions for future work.

II. Multi-chip Networks

Consider a network that broadcasts events to n chips. The network in Fig. 2(a) uses a bus for inter-chip communication, and the time spent signaling (per event) is: $T_{\mathsf{bus}} = 8(d_{\mathsf{pitch}}/v_{\mathsf{pcb}})(n-1)$, where d_{pitch} is the distance between chips, v_{pcb} is the propagation velocity, and n is the number of chips in the system. A factor of 8 is included because each communication has a request–acknowledge handshake, which is 4 transitions, and each transition must make a round-trip before reaching steady-state in a source-terminated link. A typical time is: $T_{\mathsf{bus}} = 3.2(n-1)$, (in nanoseconds) assuming 2 inches for d_{pitch}, which corresponds to 400 picoseconds in a typical PCB [12]. The obvious drawback to using a bus is that capacity decreases as n increases. More fundamentally, the capacity of the bus is limited by the time it takes a signal to propagate across its length (set by the speed of light in a PCB). Therefore, bus architectures are not an expandable solution because more chips translate to longer buses.

The network in Fig. 2(b) implements inter-chip communication through a 1-D grid, and offers an expandable solution. In this configuration, events propagate between nearest-neighbor chips such that multiple links can be active simultaneously. Communication between chips takes: $4(d_{\mathsf{pitch}}/v_{\mathsf{pcb}})$, or $T_{\mathsf{grid}} = 1.6$ nanoseconds.[3] Capacity is independent of n for grid communication because we have segmented the network into n nearest-neighbor links: that is, in the amount of time it takes a bus to broadcast one event to n chips, a grid can broadcast n events to n chips. The system is expandable because adding chips does not increase the cycle-time of inter-chip communication.[4]

TABLE I
Network Design

	Neuromorphic System	Supercomputer
Traffic	Poisson	Bursty
Distribution	Broadcast	Targeted
Implementation	Asynchronous	Synchronous

We now show that a grid does not sacrifice latency even though events must be relayed from chip to chip. When the system is running at 95% of its capacity, for instance, queuing theory predicts that events wait an average of 20 slots before being serviced.[5] In a grid, this corresponds to 20 slots for each link, a total latency of $(n-1)20T_{\mathsf{grid}} = 32(n-1)$ nanoseconds. In a bus, the latency is $20T_{\mathsf{bus}} = 64(n-1)$ nanoseconds since a bus broadcasts to all the chips in T_{bus}. The grid's faster cycle-time compensates for the increased number of hops and results in a latency that is comparable to the bus, both of which scale with n. If we further consider that the grid is servicing more events than the bus for the same 95% loading (again due to its shorter cycle-time), we conclude that the grid is a superior choice. It is important to note that the 1ms timing precision required by neuromorphic systems is easily met by both grids and buses; throughput is ultimately limited by queue size.

The grid network architecture that we have chosen for our neuromorphic system is used in supercomputers — but supercomputers use targeted routing and synchronous signaling. In addition to these two differences (Table I), the traffic patterns are distinct. For example, a chip with 10,000 neurons each firing (independently) at an average rate of 100Hz will almost never exceed its expected load in 1ms by more than 20% ($p < 10^{-9}$). In contrast, when a processor experiences a cache miss, its load may increase from 100Mbits/s (average) to 1Gbit/s. Accordingly, we can design our network for the average load whereas supercomputer networks must be designed to handle the peak bandwidth (otherwise they incur large latencies).[6]

III. Multi-chip Communication

To broadcast spikes in a grid network, each event must visit all the chips in the system. We implement this broadcasting using an on-chip communication controller, which interfaces a local neuron array to nearest neighbor chips. This controller

[3]We use a factor of 4 instead of 8 because grid connections have a single termination point and only require one leg of the trip for signal transmission. Specific details on bus and grid implementation (e.g., using synchronous versus asynchronous, signaling protocols, or using different termination schemes) will affect the scaling factors of our calculations, but they are secondary to our expandability argument.

[4]Our expandability argument also pertains to multidimensional grids and buses. The multidimensional grids are expandable because their capacity is set by the distance between nearest neighbors, which is determined by how the network topology maps into physical space, whereas multidimensional buses are not expandable because their capacity decreases as chips are added (e.g., the capacity of a 2-D bus would be inversely proportional to $\sqrt{n}$ in the optimal configuration).

[5]This assumes a non-deterministic service time because in the high-load case, the burst lengths are variable. [13]

[6]Supercomputers often combine the loads of multiple processors using concentrators to make traffic less bursty, but not to the extent that neuromorphic chips do. Consult [14] for an example.

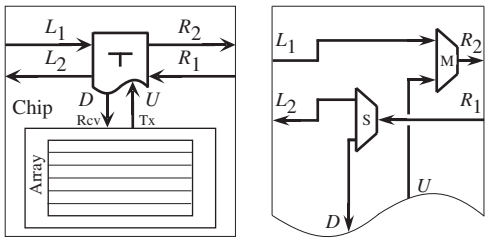


Fig. 3. (a) A $\top$ communicates to a left chip through L_1 and L_2, a right chip through R_1 and R_2 and the array through U and D. (b) The datapath for a $\top$.

has three bi-directional links: a left link and a right link to relay events between chips, and a bottom link to route events to and from the array. Because the three bi-directional links resemble a $\top$, we refer to the process as such. Fig. 3(a) illustrates a $\top$ with labels for each of its ports.

We adopt the following datapath for a $\top$, which is optimized for compactness (Fig. 3(b)). It collects events from transmitters in the rightward path (via merges, M) and distributes events to receivers in the leftward path (via splits, S). In this configuration, the rightmost chip in the grid must connect R_2 to R_1, thereby directing the transmitter stream toward the receivers. In this datapath, the loads are unbalanced, but that can be tolerated since the traffic in neural systems is non-bursty.

A. Packet Protocol

Spike events are relayed between chips using **packets**, which can have an arbitrary number of words, as specified by the following notation:

$$\mathbf{x} = \{x[1], x[2], \ldots, x[N]\}. \tag{1}$$

Here, $x[1]$ is the head word, $x[2]$ through $x[N-1]$ are embedded data, and $x[N]$ is the tail word. The temporal order of words within a packet corresponds to the order of the words within the brackets from left to right; hence $x[1]$ precedes $x[2]$.

At the level of the neuron array, communication follows a word-serial address-event format where coincident spike events within a row are sent as a burst of addresses [15], [16]. A burst, which consists of a row address (y), followed by an arbitrary number of column addresses ($x[1]$ to $x[N]$), followed by an end-of-burst symbol (ϕ), can be ported into the packet notation:

$$\mathbf{b} = \{y, x[1], \ldots, x[N], \phi\}. \tag{2}$$

At the chip level, we wish to keep track of the event's source (i.e., which chip generated the event). This is accomplished simply by prepending a chip address. In our system, the same chip address is inserted at every $\top$, and its value is incremented or decremented as it is relayed from chip to chip (relative addressing). Inserting unique chip addresses is also possible, however, this requires manual configuration at each $\top$. Naturally, before packets can be delivered to the receiving array, the chip address must be stripped off so that the packet is in the appropriate burst format. We describe how the $\top$ performs these packet operations in Section III-B.

TABLE II
CHP LANGUAGE CONSTRUCTS

Operation	Notation	Explanation
Process	P_i	A composition of communications
Guarding	G_i	$\equiv B_i \rightarrow P_i$. Execute P_i if B_i is true
Sequential	$P_1; P_2$	P_1 ends before P_2 starts
Concurrent	$P_1 \| P_2$	There is no constraint
Repetition	$*[P_1; P_2]$	$\equiv P_1; P_2; P_1; P_2; \ldots$. Repeats forever
Selection	$[G_1 \| G_2]$	Execute P_i for which B_i is true
Arbitration	$[G_1 \mid G_2]$	Required if B_i not mutually exclusive
Input	$A?x$	Read data from port A to register x
Output	$A!x$	Write data from register x to port A
Probe	$\overline{A}$	Is communication pending on port A?
Packet	$p[j].i$	The ith bit of p's j-th word
Assignment	$y := x$	Copy data from x to y

The packet-based address-event protocol that we have proposed is an important innovation for neuromorphic systems. For one, packets are able to bundle system-level information along with spike events without increasing the width of the link (as opposed to previous neuromorphic implementations [11]). This opens the door for creating multi-dimensional grids simply by inserting additional head words each time the packet traverses to higher dimensions. Furthermore, the packets themselves are parsimonious since they only include the source information of events, and not general routing information.

B. $\top$ Specification

We use the Concurrent Hardware Processes (CHP) language [17] to describe how $\top$s modify both the structure and content of packets as they traverse the grid. A brief summary of the CHP notation is provided in Table II.

To make our CHP programs succinct, we invoke the operators INCR, INSERT, DECR, and DELETE; these operators modify packets according to the following functional descriptions:

- $\mathsf{INSERT}(\{x[1], \ldots, x[N]\}, y) = \{y, x[1], \ldots, x[N]\}$ prepends a new head word y,
- $\mathsf{DELETE}(\{x[1], \ldots, x[N]\}) = \{x[2], \ldots, x[N]\}$ deletes the current head word $x[1]$,
- $\mathsf{INCR}(\{x[1], \ldots, x[N]\}) = \{x[1]+1, x[2], \ldots, x[N]\}$ increments the head word $x[1]$,
- $\mathsf{DECR}(\{x[1], \ldots, x[N]\}) = \{x[1]-1, x[2], \ldots, x[N]\}$ decrements the head word $x[1]$.

Based on these operators, a $\top$ can be specified by two concurrent CHP processes:

$$*[[\overline{U} \rightarrow R_2!\mathsf{INSERT}(U?, 0) \mid \overline{L_1} \rightarrow R_2!\mathsf{INCR}(L_1?)]];$$
$$\| *[t := \mathsf{DECR}(R_1?); L_2!t \| D!\mathsf{DELETE}(t)]$$

where t is a local variable of type packet (Fig. 4). The first process (rightward path in Fig. 4) merges events from the local neuron array (U) with events from upstream chips (L_1). Prior to the merge, bursts generated by the local transmitter are prepended with a chip address of 0 so that they may be differentiated from bursts originating in upstream transmitters. Accordingly, chip addresses from upstream packets are incremented before they are relayed to neighbors (R_2).

The second process (leftward path in Fig. 4) delivers packets to the local receiver (D) while also relaying them to neighbors

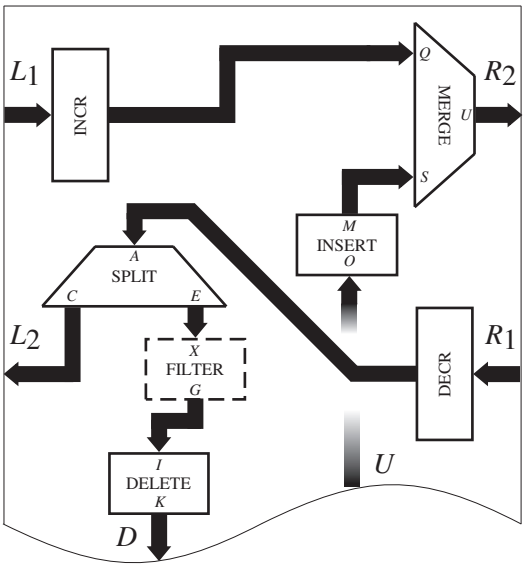


Fig. 4. The internal blocks of a $\top$ and their connections.

(L_2); the filter block will be described in Section III-C. Before a packet is delivered to the receiver, its chip address is stripped via a delete operation, returning the packet to its original word-serial format (i.e., the head word is now the row address). We also maintain relative addressing in this leftward path by decrementing chip addresses before the split; this will be useful when we selectively deliver packets based on their chip of origin (described next).

C. Selective Packet Delivery

The $\top$ that we have described thus far obliviously broadcasts events to each receiver, thereby indiscriminantly delivering them to corresponding locations on all the chips. However, by including a filter block between the split and delete (dashed block in Fig. 4), we are able to selectively deliver packets based on their chip of origin. We consider two modes: The first mode, which we call **targeted**, delivers packets if the incoming chip address is 0; this is useful for addressing a single chip. The second mode, which we call **excluded**, delivers packets if the incoming head word is not 0; this is useful for broadcasting to all but the target chip.

We implement targeted deliveries by modifying the second CHP process:

$$*[t := \mathsf{DECR}(R_1?); L_2!t \| [t[1].\mathsf{borr} \rightarrow D!\mathsf{DELETE}(t) \; [\!] \; \neg t[1].\mathsf{borr} \rightarrow \mathsf{skip}]]$$

where $t[1]$.borr is the most significant borrow-out bit from the decremented chip address. Here, the packet is delivered to the receiver (D) only when the borrow-out bit is true (i.e., when the incoming packet on R_1 has a chip address of 0). Similarly, the excluded mode can be implemented by delivering the packet when the borrow-out bit is false (not shown). In practice, our filter block is capable of performing both targeted and excluded deliveries, set by a user defined flag at the head of a packet; the details are described in Section V-D.

It is important to realize that the filter is not limited to targeted and excluded deliveries — in fact, the filter can use a programmable lookup table to decide which source chips should connect to which target chips, thus supporting arbitrary inter-chip pointwise connectivity. Although our current specification does not include this programmability, we are currently in the process of building such a system.

IV. Signaling Protocols

There is quite a bit of flexibility when implementing our CHP programs, and in particular, we can choose the signaling protocols for each port (using the process of hand-shaking expansion, which is described in the Appendix). We choose to use two protocols instead of one to satisfy a number of constraints [18]. We use **single-rail** signaling (Fig. 5) to be compatible with the address-event transceiver circuitry surrounding the neuron array, which is generated by the publicly available **ChipGen** software [19], and to be compatible with previous neuromorphic chips (e.g., a retina chip [20]). We use **dual-rail** signaling (Fig. 6) when we have to perform data manipulation, as the timing requirements for single-rail signaling are difficult to meet in these cases.

We use the following naming conventions: For single-rail signaling, port P's four control lines are named `(pr,pa,qr,qa)`, where `p` indicates the head–tail words and `q` indicates the data words; the appended letters `r` and `a` denote request (data validity) and acknowledgment (data acceptance), respectively. For dual-rail signaling, we indicate head, tail, and data lines by appending `h`, `e`, and `n..0`, respectively.[7] Each dual-rail bit uses two lines, appended with `t` (true) and `f` (false), to indicate the bit's state (see Fig. 6).

Blocks that are designed in different protocols are interfaced together through protocol converters. It is important to note that in most systems where there are no *a priori* constraints, using a single protocol throughout simplifies the design. In fact, the ChipGen software is currently being migrated to a delay-insensitive code so that protocol conversions in our system can be eliminated altogether. We expect that the single-rail standard that has dominated single chip neuromorphic systems will be phased out as large scale multi-chip systems become more popular.

V. $\top$ Implementation

Here, we present the production rule set (PRS) and circuit descriptions for the main blocks required to build a $\top$; we provide a brief review on how these circuits were synthesized in the Appendix, along with the relevant hand-shaking expansions (HSEs). PRS syntax is described in Table III. The designs for the protocol converters have been omitted to save space.

[7]The head signal is superfluous since we know that a head occurs immediately after seeing a tail, however, we explicitly include it because it simplifies the design of the increment–decrement and filter circuits (otherwise we would need to reconstruct the head signal at each of these blocks).

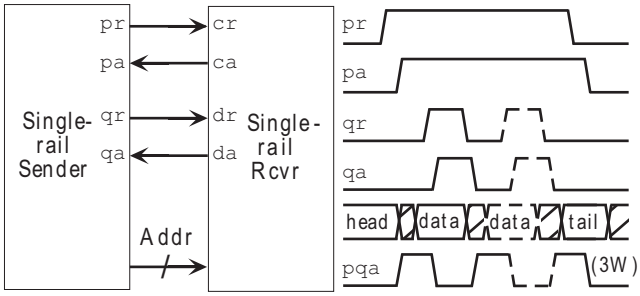


Fig. 5. Single-rail communication segregates the datapath from the control signals and requires that data validity precedes the requests. Our single-rail protocol uses a head–tail request-acknowledge pair (pr,pa) to signify head–tail words, and a data request-acknowledge pair (qr,qa) to signify data words. For inter-chip communication, we use three control lines (instead of four) by combining two acknowledges into a single line (pqa).

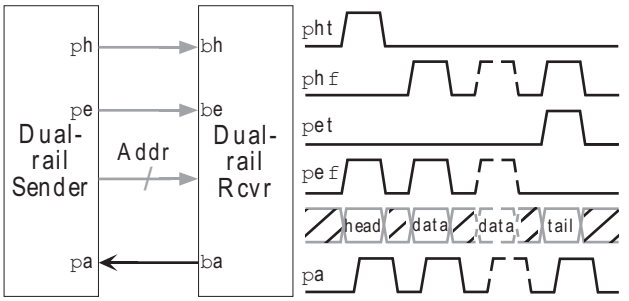


Fig. 6. Dual-rail communication, which uses two lines (true and false) to encode each bit, combines the data with the hand shake signals; the data is valid if line true or false is high, and neutral when both are low; the state where both lines are high is disallowed. A dual-rail sender initiates a communication by changing its data lines to a valid state, and a receiver acknowledges with pa. The position of the word within a packet is determined by the head (h) and tail (e) bits. Grey arrows represent dual-rail signals, which are 2 lines/bit, whereas black arrows represent single-rail signals.

A. Merge and Split

The MERGE block combines packets from two ports in an orderly fashion. We design the merge in single-rail instead of dual-rail (see [21] for a delay-insensitive implementation) because single-rail is more compact (two n-bit datapaths can be merged using 2n wires as opposed to 4n wires). However, as we described earlier, the primary reason we use the single-rail is because of compatibility.

The merge operation requires that we arbitrate between the Q and S ports (as specified in the CHP), since these communications can occur concurrently. Accordingly, we use a mutual exclusion element [22] to create the signals aqr and asr from packet-head requests qr and sr, respectively. We show the compiled PRS and circuit implementations in Fig. 7.

The PRS for n, sa, and ta have been omitted because they are the same form as for m, qa, and ra, respectively, using guards from the opposite side (n uses asr instead of aqr, and sa and ta use n instead of m).

The SPLIT block copies an arriving packet to two ports. This operation is implemented trivially in single-rail by using wires to connect the incoming and outgoing head and body requests, and C-elements to connect incoming and outgoing head and body acknowledges; we omit the PRS and circuits because of their simplicity. A dual-rail split, which is required when the FILTER block is included in the system, can be built by using wires to connect each incoming and outgoing datalines, and a C-element to connect incoming and outgoing acknowledges.

TABLE III
PRS AND HSE PRIMITIVES

Operation	Notation	Explanation
Signal	v	Voltage on a node
Complement	~v	Inversion of v
And	v & w	High if both are high
Or	v | w	Low if both are low
Set	v+	Drive v high
Clear	v-	Drive v low
Wait	[v]	Wait until v is high
Sequential	u -> v+	[u];v+ in HSE
Concurrent	v+,w+	v+,w+ in HSE
Repetition	*[...]	Just like in CHP

B. Insert and Delete

The INSERT block prepends a word (chip address) to a packet. We design the insert in single-rail, and its PRS and circuit implementation are shown in Fig. 7. To perform the insert, the output datapath is multiplexed between the pre-programmed chip address (zero) and the input datapath from M (datapath not shown); specifically, the output datapath is set to zero when oa is low, and to the input datapath otherwise. We must enforce the timing assumption that the output of the multiplexer is valid at the time a request is made; this was accomplished by matching delay paths in the layout.

Notice that our circuit requires a custom element (_pr) whose PRS does not match that of any standard elements; _pr is implemented using a pull-up network of PMOS transistors, a pull-down network of NMOS transistors and a **staticizer**, which is a weak feedback inverter used to hold the output logic state when neither the pull-up or pull-down networks are active.

The DELETE block strips the chip address from a packet, and is also designed in single-rail. The PRS and circuits for the delete are shown in Fig. 7.

C. Increment, Decrement, and FIFOs

The INCR and DECR blocks, which perform data manipulation, are implemented in dual-rail. We omit the synthesis for these blocks because they have been described in detail previously [23]; our specific implementations are simplified versions of the adder–subtracter circuits from this reference. Note that our CHP specification requires that we *only* increment and decrement packet head words; this is accomplished by connecting the dual-rail head request (ph from Fig. 6) to the least-significant carry–borrow-in. That is, we add–subtract 1 from the head word, and 0 for all other words.

Surrounding each INCR and DECR block are bit-independent dual-rail FIFOs (first-in-first-out pipelined latches), as described in [24], [25]; within the FIFO path, each bit ripples to the next buffer stage as soon as it is empty, independent of other bits in the datapath. Of course, the datapath must be re-aligned each time we convert from

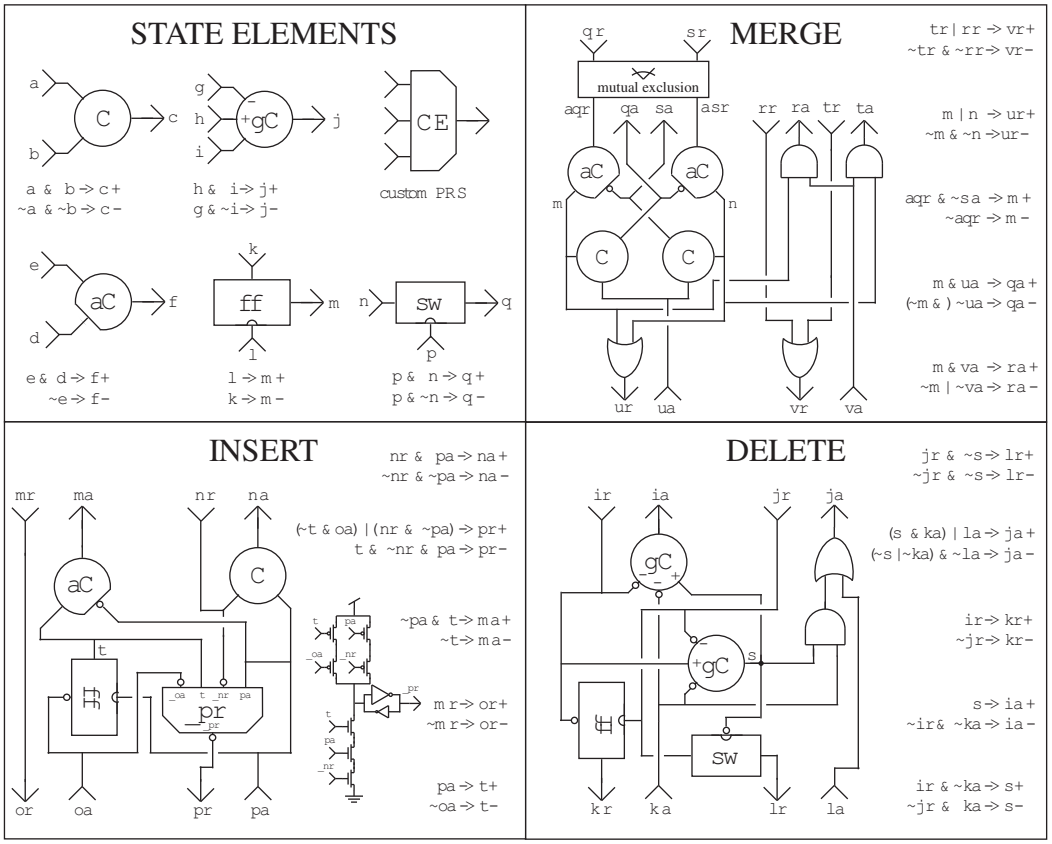


Fig. 7. PRS and circuits for $\top$ operators designed in single-rail. Note that we denote active low signals by prepending an underscore (e.g., _a) to the signal's name.

dual-rail back to single-rail. This alignment is implemented by checking that all of the bit-independent dual-rail signals are conjunctively valid–neutral using a C-tree. The latency that a C-tree adds to the datapath is rather costly ($\log_2(n)$ stages), and provides further impetus for maintaining a dual-rail code throughout.

TABLE IV
PACKET DELIVERY

Mode	xhn-1	Chip Address	xhn	Delivery
Target	0	Zero	1	Yes
Target	0	Not Zero	0	No
Exclude	1	Zero	0	No
Exclude	1	Not Zero	1	Yes

D. Filter

The FILTER block selectively passes or filters packets based on their chip of origin. To support packet filtering, we reserve the second highest-order bit of the head word (xhn-1) for specifying the mode (*mode-select bit*), and the highest-order bit (xhn) for deciding whether the packet is earmarked for delivery (*payload bit*). The mode-select bit is a user defined flag that is set off chip (target or exclude), and the payload bit is generated on chip within the FILTER block as specified in Table IV. The payload bit, which is overwritten each time the packet is relayed, is not required to be part of the off-chip datapath, however, having access to it off chip greatly simplifies testing. Note that because we reserve the two highest-order bits of the head address for filtering, only the lower n-2 bits correspond to the chip address; accordingly, we only perform increment–decrement operations on these lower bits leaving the mode-select and payload bits unchanged.

We build the FILTER in two stages, as shown in the CHP inset in Fig. 8: First, we compute the payload bit in the PAYLOAD block, and then we appropriately filter or deliver the packet based on this bit in the DECISION block. Both the payload and decision computations are performed in dual-rail to circumvent a detailed timing analysis; accordingly, we replace the single-rail split with a dual-rail one, which eliminates superfluous protocol conversions. The output of the DECISION block is directly sequenced in single-rail since this yields a more compact solution.

The PRS and circuits for the PAYLOAD block, which are shown in Fig. 8, follow Table IV (note that there is no sequencing). For example, the payload bit is set high (ynt is true) in the targeted mode when the chip address is zero (xbt indicates an overflow), and in the excluded mode when the chip address is not zero; the payload bit is cleared when its inputs become neutral. All other inputs are simply passed through the payload block with wires (not shown); the borrow-out bit can be dropped at this point since it is no longer required in subsequent blocks.

The DECISION block checks the payload bit (zn) in the head word to determine if a packet should be delivered or

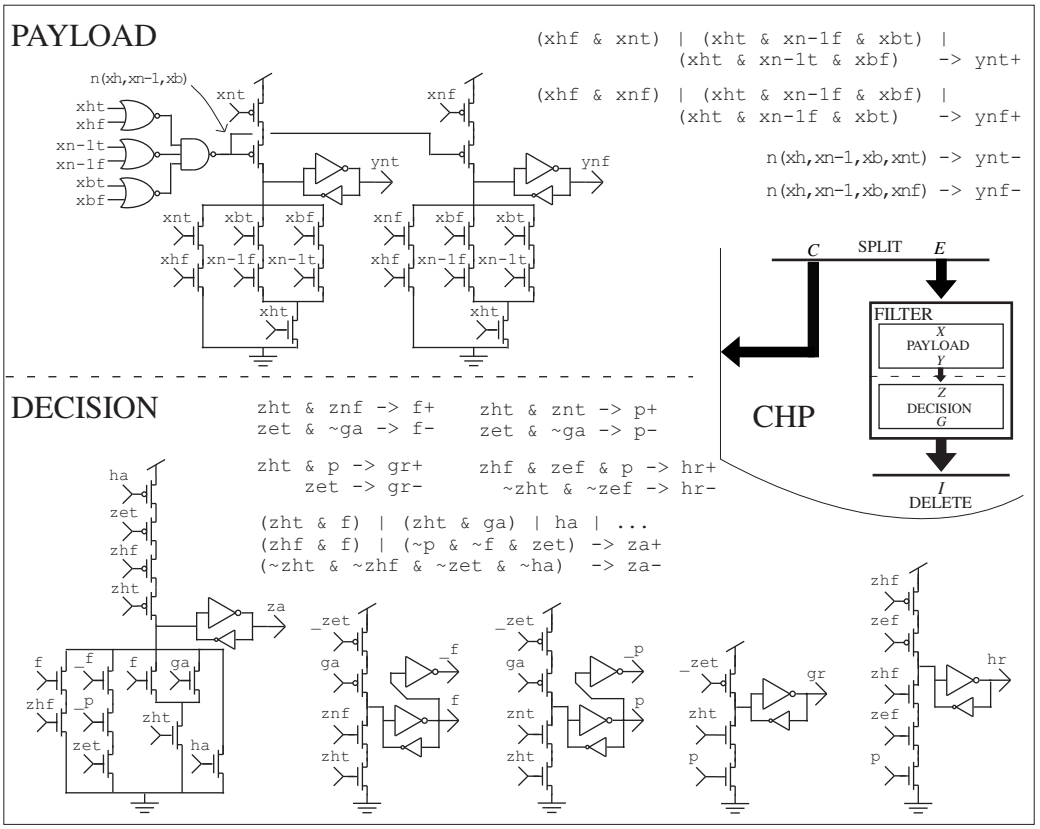


Fig. 8. PRS and circuits for the FILTER block. Note that the PAYLOAD block uses `n(·)` to test for neutrality (i.e., all of its arguments are low conjunctively) before setting `ynt` or `ynf` low.

filtered. We introduce two state variables that indicate which case applies: `p` is set when a packet will be passed through and `f` is set when a packet will be filtered. The PRS and circuits for the decision computation is shown in Fig. 8. Since none of the PRS match the standard elements, we show the circuits solely with transistors.

VI. Results

The circuits presented in this paper have been implemented in two chips using full-custom design, both fabricated in 0.25-μm CMOS technology. The first chip integrates a $\top$ (without the filter block) with a transceiver array that has $32 \times 64 \times 4$ (8,192) orientation selective neurons. This system was designed for an image-processing application proposed by Shi and Boahen [26]. The second chip (with the filter block) is an inverse imager whose $320 \times 240 \times 4$ pixels convert pulse-frequency into analog currents that are video-encoded using a scanner [27]. This second chip supports targeted and excluded packet filtering. Our goal in this section is to verify that the $\top$s operate correctly, in addition to gauging their performance — it is not our intention to present an application (see [2], [28] for details of applications).

A. $\top$ testing

We built the system shown in Fig. 9 to test the functionality of a $\top$. In particular, we recorded packet communication at M-in and M-out to test the merge, and at S-in and S-out to test the split. At each of these four test points, we captured signals on the three-wire link that comprises: an 8-bit address represented as a decimal (`Addr`), an active-high head–tail request (`pr`), an active-low data request (`_qr`), and an acknowledge (`pqa`); refer to Fig. 5 for the sequence of the three-wire hand shake protocol.

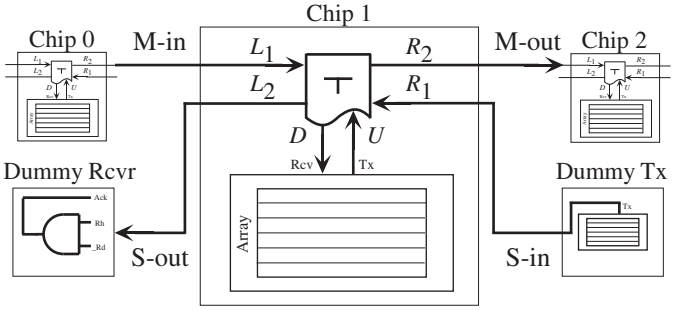


Fig. 9. Setup used to test merge and split sides of the $\top$.

To test the merge path, we cascaded Chips 0, 1 and 2 as shown in Fig. 9 (R_2 from Chip 0 is connected to L_1 of Chip 1, and R_2 of Chip 1 is connected to L_1 of Chip 2). Each chip has a neuron array that is generating events; in this example, we are interested in the case of colliding packets. Fig. 10 shows traces for M-in and M-out captured with a logic analyzer over a 260 ns duration (20 ns/div). The first recorded packet in this example is $\{0, 32, 127\}$ at M-out; from these addresses, we can infer that a neuron from row 32 and column 127 was active in

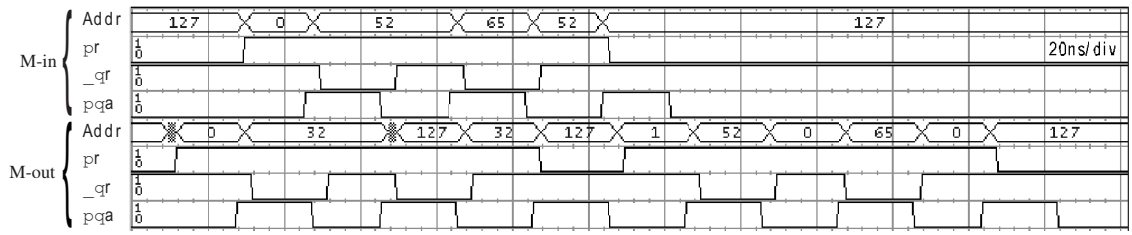


Fig. 10. Captured merge path signals showing two colliding packets, one from Chip 0 (M-in), the other from Chip 1 (M-out). pr is the active-high head–tail request, _qr is the active-low data request, and pqa is the acknowledge. The later arriving packet (from Chip 0) is queued and sent out as soon as the first packet is transmitted. Data was taken at 2ns timing resolution. Note that each chip is located on a different board, connected through 8-inch cables.

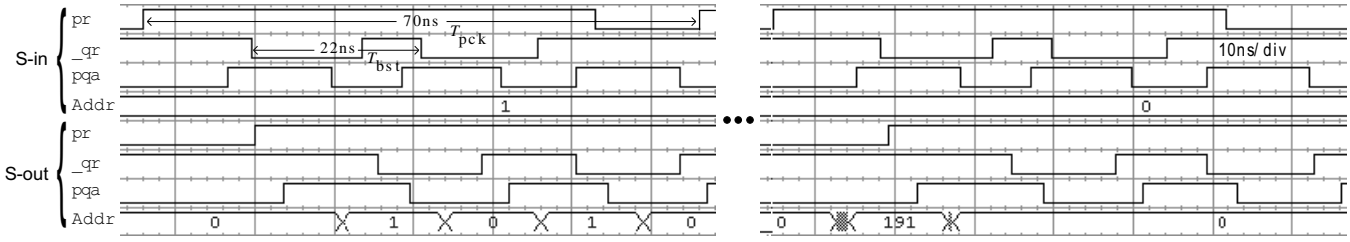


Fig. 11. Captured split path signals showing (left) a packet being correctly filtered (bit 8 of the head word is 0), and (right) a packet being correctly delivered (bit 8 of the head word is 1). Data was taken at 500ps timing resolution. The chip, CPLD, and AND gate are all located on the same board.

Chip 1 (since the chip-address is 0). During this Chip 1 to Chip 2 communication, we observe the packet $\{0, 52, 65\}$ at M-in. Chip 1 is able to queue this incoming packet using FIFOs (this chip has a 64 deep FIFO bank) while it is transmitting the first packet to Chip 2. Immediately following the first packet, the queued packet is transmitted to Chip 2, with an appropriately incremented chip-address.

To test the split path (with targeting), we fed in packets to R_1 of Chip 1 using a dummy transmitter (implemented with a CPLD), which allowed us to control both the mode and the chip-address value. A dummy receiver sends an acknowledge signal to L_2, generated by ANDing together pr and _qr requests from L_2. Fig. 11 shows the resulting traces from S-in and S-out in the cases that the dummy transmitter generates the packets $\{1, 1, 1\}$ (left) and $\{0, 0, 0\}$ (right). For the first input packet at S-in, the chip-address is 1 and the mode is set to targeted (the first 6 bits of the head word correspond to the chip-address, and the 7th bit corresponds to the mode select-bit). The packet at S-out is $\{0, 1, 1\}$, indicating that the chip-address was decremented and that the packet was filtered (i.e., the incoming chip-address was greater than 0, so bit 8, or the delivery status-bit, is 0). For the second input packet, the chip-address is 0 so this packet should be delivered. The packet at S-out is $\{191, 0, 0\}$ (191 is 10111111 in binary), indicating that the chip-address was decremented and that the packet was delivered (delivery status bit is 1).

Table V summarizes the results for all mode–delivery combinations. To check whether the packets that claimed to be delivered were actually delivered, we verified that the delivery status-bit coincided with pixel stimulation using a scanner [29] that allowed us to monitor the analog output of a targeted pixel (data not shown). These results demonstrate that the circuits described in this paper function correctly.

TABLE V
TESTED PACKET ROUTING ACTIONS

Input Head	Chip-Address	Mode Select	Output Head	Delivered
00000001	1	Targeted	00000000	No
00000000	0	Targeted	10111111	Yes
01000000	0	Excluded	01111111	No
01000001	1	Excluded	11000000	Yes

B. Signaling rate

We performed timing measurements to gauge the signaling rates of our links. For these measurements, we only consider the split side because its setup connects chips with short PCB traces. We expect that the merge side would have nearly identical rates (since they share similar circuitry); however, we were not able to test this because the merge setup connects chips with 8-inch ribbon cables, significantly increasing signaling times.

For the split side, the measured burst rate (T_{bst}) was 22ns using a CPLD as a dummy transmitter (Fig. 11); this corresponds to data-word rates of 45.5 M events/s. We do not expect that replacing the dummy circuitry with our chips would affect these measurements because of the internal pipelines. These measurements reveal that the rate of data reception is limited by the delay of the receiver acknowledgment (pqa). In particular, Fig. 11 shows that the receiver takes at least 10ns to acknowledge data reception (pr going high or _qr going low). Simulations confirm that a protocol converter circuit causes this delay. We discuss some potential ways to speed up communication in the following section.

Based on T_{bst} and T_{pck} measurements garnered from Fig. 11, we can calculate the maximum load that our system can handle using the approximation derived in [13]. Briefly,

the number of slots spent waiting in the high-load case is:

$$n_{\top} \approx \frac{N_{\mathsf{row}}(T_{\mathsf{pck}} - T)\frac{T_{\mathsf{bst}}}{T}}{T_{\mathsf{bst}}(1 - \frac{T_{\mathsf{bst}}}{T})} \quad (3)$$

where N_{row} is the number of rows and T is the mean inter-event interval. Plugging in $N_{\mathsf{row}} = 64$, $T_{\mathsf{pck}} = 70$ns, $T_{\mathsf{bst}} = 22$ns, and $n_{\top}$ =32 (we have 64 for FIFOs, each with a slack of 1/2), we estimate that our queue size can handle $1/T = 18.5$ million events/s, or 40% of our capacity. At this throughput, our latency (per link) is $n_{\top}T = 1.73\mu$s. To reach 80% of the capacity, the number of FIFOs would need to be increased from 64 to 990, and the latency would increase from 1.73μs to 13.6μs.

VII. Discussion

In this paper, we have examined a number of the design challenges in creating multichip neuromorphic systems using word-serial links. To support dense neural connections without generating superfluous network traffic, we chose to distribute events using a broadcast. We showed that a 1-D grid, which links nearest neighbors, is superior to a bus for broadcasting because the grid is expandable (i.e., adding chips does not decrease capacity). A communication controller, called a $\top$, was designed to relay events in a grid while keeping track of their sources.

In practice, the $\top$ operated as specified — though signaling rate measurements revealed a number of bottlenecks that limit performance. One bottleneck, which is described in Section VI-B, is caused by on-chip protocol conversions. The obvious solution is to eliminate these conversions altogether and maintain a delay-insensitive code throughout. An additional bottleneck is introduced by our four-phase off-chip signaling protocol. Currently, data communication requires that the control signals (_qr,pqa) are reset after each data transfer (i.e., two vacuous transitions). The solution is to migrate to a two-phase off-chip protocol that transfers data on both upward and downward transitions, which would result in a doubling of the data rates.

The $\top$ that we described provides simple delivery functions based on a packet's point of origin. The current implementation supports all-to-all pointwise connectivity (i.e., every neuron in one chip connects to corresponding locations in the other chips), and limited connectivity (targeted and excluded deliveries). This same $\top$ design could be used to create more sophisticated connection patterns. In particular, we can include a programmable look-up table in the filter that would specify which source chips are connected to which target chips. Currently, we are working on such a system [30].

The broadcast strategy that we have presented in this paper could be used to implement general connectivity between neurons in an efficient manner. For example, an event from a neuron that projects to a 1,000 neurons distributed across a ten-chip grid could first be broadcast to all the chips, and then replicated a 100 times within each chip (where communication is discounted). Fanning out in this way is efficient because the expensive operation of replicating events occurs in parallel. We plan to add the required look-up table and the circuitry to replicate events to the $\top$ in the near future.

Acknowledgment

The authors would like to thank Thomas Choi for designing and implementing the Gabor filter array that was used in the first multichip system, and Yann Baud and Florian Bochud for designing a pulse-frequency receiver array that was used in the second multichip system. We also thank Brian Taba and Kai Hynna for their invaluable help on using the automated AER tools to generate the core.

Appendix

We transform our CHP descriptions into circuits following Martin's methodology for synthesizing asynchronous systems [31].[8] First, each communication is fleshed out into a request–acknowledge sequence using a notation known as hand-shaking expansion (HSE); in essence, HSE specifies the order of signal transitions (or hand shakes) as required for self-timed data transfer between processes. Then, the HSE is decomposed into a set of boolean expressions where guards enforce the specified order of signal transitions; the guards and their corresponding transitions, which are called a production rule set (PRS), can be directly implemented with transistors.

Here, we present HSE for relevant blocks; please refer to Table III for a summary of HSE notation.

A. Merge

The merge has two input ports `(qr,qa,rr,ra)` and `(sr,sa,tr,ta)`, and one output port `(ur,ua,vr,va)` (all in single-rail), and its HSE is:

```
# merge #
   *[[aqr & ~sa]; m+; ur+; [ua]; qa+;
     [~aqr]; m-; ur-; [~ua]; qa-]
|| *[[rr]; vr+; [va]; ra+; [~rr]; vr-;
     [~va]; ra-]
|| *[[asr & ~qa]; n+; ur+; [ua]; sa+;
     [~asr]; n-; ur-; [~ua]; sa-]
|| *[[tr]; vr+ [va]; ta+; [~tr]; vr-;
     [~va]; ta-]
```

The signals `aqr` and `asr` are generated using a mutual-exclusion element [22], [6] that arbitrates between `qr` and `sr` as shown in Fig. 7; this element is necessary because `qr` and `sr` may arrive near simultaneously and cause a race condition. Because the mutual exclusion element does not follow our single-rail sequence (a new side can be selected before receiving an acknowledgment), we must introduce two internal selection signals, `m` and `n` to enforce the correct sequence (these signals are only set once the complementary side has completed its previous communication).

B. Insert

The INSERT has an input port `(mr,ma,nr,na)` and an output port `(or,oa,pr,pa)` (both in single-rail), and its HSE is:

[8] A number of different methodologies have been successfully used to design asynchronous systems, all of which are roughly equivalent (see [24], [32] for other possibilities). Martin's methodology, however, has been the de facto standard for implementing address-event systems [6].

```
# insert #
   *[[mr]; or+; [oa]; pr+; [pa]; t+; pr-;
     [~pa]; ma+; [~mr]; or-; [~oa]; t-; ma-]
|| *[[nr]; pr+; [pa]; na+; [~nr]; pr-;
     [~pa]; na-]
```

When there is a pending packet communication on the input side of the INSERT (`mr` is true), the output initiates a head communication on `(or,oa)` while the datapath output is multiplexed to a pre-programmed word (0). Immediately following this inserted head word, the output datapath is multiplexed back to the input datapath (switched by `oa`), and then performs a body communication on `(pr,pa)`, sending the input packet's head word as the output packet's first data word. Subsequent body communications resume as normal.

C. Delete

The DELETE block has an input port `(ir,ia,jr,ja)` and an output port `(kr,ka,lr,la)` (both in single-rail), and its HSE is:

```
# delete #
   *[[ir]; s+; ia+; [jr]; kr+; [ka]; ja+;
     [~jr]; s-; ja-; [~ir]; kr-; [~ka]; ia-]
|| *[[jr & ~s]; lr+; [la]; ja+; [~jr]; lr-;
     [~la]; ja-]
```

When there is a packet communication pending on the input side `[ir]`, we immediately acknowledge by taking `ia` high without ever starting an output communication. Instead, we use the second word from the input packet as the output packet's head; it is communicated by taking `kr` high after `[jr]`. Subsequent words are fed through to the output as normal. The `s` variable is used to disambiguate the first received body communication on `jr` from subsequent ones. That is, we use `s` to block `lr`.

D. Split

The SPLIT has an input port `(ar,aa,br,ba)` and two output ports `(cr,ca,dr,da)` and `(er,ea,fr,fa)` (all in single-rail), and its HSE:

```
# split #
   *[[ar]; cr+, er+; [ca & ea]; aa+;
     [~ar]; cr-, er-; [~ca & ~ea]; aa-]
|| *[[br]; dr+, fr+; [da & fa]; ba+;
     [~br]; dr-, fr-; [~da & ~fa]; ba-]
```

Notice that both head–tail and data sequences are identical and independent (they do not share any signals).

E. Filter

The payload stage has a dual-rail input port (`xb,xn,xn-1,xh`) and a dual-rail output port (`yb,yn,yn-1,yh`), where `xb` is the borrow-out bit from the decremented chip address. There is no sequencing for computing the payload bit since its validity and neutrality are determined by its inputs.

The DECISION has a dual-rail input port (`zh,ze,zn,za`) and a single-rail output (`gr,ga,hr,ha`). The HSE is:

```
# decision #
   *[[zht & znt]; p+; gr+; [ga]; za+;
     [~zht]; za-]
|| *[[zht & znf]; f+; za+; [~zht]; za-]
|| *[[zhf & zef & p]; hr+; [ha]; za+;
     [~zht & ~zef]; hr-; [~ha]; za-]
|| *[[zhf & zef & f]; za+; [~zhf & ~zef]; za-]
|| *[[zet]; gr-; [~ga]; p-, f-; za+;
     [~zet]; za-]
```

REFERENCES

[1] J. V. Arthur and K. Boahen, "Recurrently connected silicon neurons with active dendrites for one-shot learning," in *IJCNN'04 International Joint Conference on Neural Networks*. IEEE Press, 2004, pp. 1699–1704.

[2] T. Y. W. Choi, P. A. Merolla, J. V. Arthur, K. Boahen, and B. E. Shi, "Neuromorphic implementation of orientation hypercolumns," *IEEE Transactions on Circuits and Systems-I: Regular Papers*, vol. 52, no. 3, pp. 1049–1060, June 2005.

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

[4] C. A. Mead, *Analog VLSI and Neural Systems*. Reading MA: Addison Wesley, 1989.

[5] A. Andreou and K. Boahen, "Translinear circuits in subthreshold MOS," *Journal of Analog Integrated Circuits and Signal Processing*, vol. 9, pp. 141–166, 1996.

[6] K. A. Boahen, "Point-to-point connectivity between neuromorphic chips using address-events," *IEEE Transactions on Circuits and Systems II*, vol. 47, no. 5, pp. 416–434, 2000.

[7] M. Mahowald, *An Analog VLSI System for Stereoscopic Vision*. Norwell, MA, USA: Kluwer Academic Publishers, 1994.

[8] M. Sivilotti, "Wiring considerations in analog VLSI systems, with application to field-programmable networks," PhD Thesis, California Institute of Technology, 1991.

[9] E. Ozalevli and C. M. Higgins, "Reconfigurable biologically inspired visual motion systems using modular neuromorphic VLSI chips," *Transactions on Circuits and Systems I: Regular Papers*, vol. 52, no. 1, pp. 79–92, 2005.

[10] G. Indiveri, W. A. M, and K. J, "A reconfigurable neuromorphic VLSI multi-chip system applied to visual motion computation," *in Proc. 7th Int. Conf. Microelectronics for Neural, Fuzzy and Bio-inspired Systems*, pp. 37–44, 1999.

[11] G. N. Patel, M. S. Reid, D. E. Schimmel, and S. P. DeWeerth, "An asynchronous architecture for modeling intersegmental neural communication," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 14, no. 2, pp. 97–110, 2006.

[12] J. Zasio, *Right The First Time: A Practical Handbook On High Speed PCB And System Design*. Speeding Edge, 2003.

[13] K. A. Boahen, "A burst-mode word-serial address-event link III: Analysis and test results," *IEEE Transactions on Circuits and Systems II*, vol. 51, no. 7, pp. 1292–1300, 2004.

[14] W. J. Dally and B. Towles, *Principles and Practices of Interconnection Networks*. San Francisco CA: Morgan Kaufmann, 2004.

[15] K. A. Boahen, "A burst-mode word-serial address-event link I: Transmitter design," *IEEE Transactions on Circuits and Systems II*, vol. 51, no. 7, pp. 1269–1280, 2004.

[16] ——, "A burst-mode word-serial address-event link II: Receiver design," *IEEE Transactions on Circuits and Systems II*, vol. 51, no. 7, pp. 1281–1292, 2004.

[17] A. Martin, "Compiling communicating processes into delay-insensitive VLSI circuits," *Distributed Computing*, vol. 1, no. 4, pp. 226–234, 1986.

[18] C. Mead and L. Conway, *Introduction to VLSI Systems*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1979.

[19] J. Lin, "Word-serial address-event transceiver layout compiler," *The Neuromorphic Engineer*, vol. 1, no. 2, p. 2, 2004.

[20] K. Zaghloul and K. A. Boahen, "Optic nerve signals in a neuromorphic chip II: Testing and results," *IEEE Transactions on Biomedical Engineering*, vol. 51, no. 4, pp. 667–675, 2004.

[21] J. Sparso and S. Furber, *Principles of Asynchronous Circuit Design - A Systems Perspective*. Kluwer Academic Press, 2001.

[22] C. Seitz, *System Timing*, L. C. C Mead, Ed. Boston: Addison-Wesley, 1980.

[23] A. Martin, "Asynchronous datapaths and the design of an asynchronous adder." California Institute of Technology, Tech. Rep. Caltech-CS-TR-91-08, 1991.
[24] I. E. Sutherland, "Micropipelines," *Communications of the ACM*, vol. 32, no. 6, pp. 720–738, 1989.
[25] A. M. Lines, "Pipelined asynchronous circuits," Masters Thesis, California Institute of Technology, 1998.
[26] B. E. Shi and K. Boahen, "Competitively coupled orientation selective cellular neural networks," *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications*, vol. 49, no. 3, pp. 388–94, March 2002.
[27] Y. Baud and F. Bochud, "AER-VGA receiver chip," Masters Thesis, Institute of Neuroinformatics, ETH Zurich, 2004.
[28] T. Y. W. Choi, B. E. Shi, and K. Boahen, "An ON-OFF orientation selective address event representation image transceiver chip," in *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 51, no. 2, 2004, pp. 342–353.
[29] C. A. Mead and T. Delbruck, "Scanners for visualizing activity of analog VLSI circuitry," *Analog Integrated Circuits and Signal Processing*, vol. 1, no. 2, pp. 93–106, 1991.
[30] J. Lin, P. Merolla, J. Arthur, and K. Boahen, "Programmable connections in neuromorphic grids," in *49th IEEE Midwest Symposium on Circuits and Systems*. IEEE Press, 2006.
[31] A. Martin, *Programming in VLSI: from Communicating Processes to Delay-Insensitive Circuits*. Addison-Wesley, 1990, pp. 1–64.
[32] T. Verhoeff, "Delay-insensitive codes - an overview," *Distributed Computing*, vol. 3, no. 1, pp. 1–8, 1988.

Paul A. Merolla received the B.S. degree with *high distinction* in electrical engineering from the University of Virginia, Charlottesville, in 2000. He is currently working towards his Ph.D. in bioengineering at the University of Pennsylvania, Philadelphia.

His research interests include VLSI models of cortical networks, vision systems, and asynchronous digital interfaces for routing and interchip connectivity.

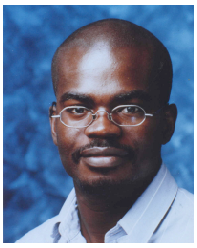
John V. Arthur received the B.S.E. degree (*summa cum laude*) in electrical engineering from Arizona State University, Tempe, in 2000. He is currently working toward the Ph.D. degree in bioengineering at the University of Pennsylvania.

His research interests include mixed-mode very large-scale integration, neuromorphic learning systems, silicon olfactory recognition, generation of neural rhythms, and asynchronous interchip communication.

Bertram E. Shi received the B.S. and M.S. degrees in electrical engineering from Stanford University, Stanford, CA, and the Ph.D. degree in electrical engineering from the University of California at Berkeley, in 1987, 1988, and 1994, respectively.

He then joined the faculty of the Department of Electrical Engineering at the Hong Kong University of Science and Technology, where he is currently an Associate Professor. His research interests are in analog very large-scale integration and cellular neural networks, bio-inspired and neuromorphic engineering, machine vision, and image processing.

Dr. Shi's IEEE activities have included Student Activities Chair of the IEEE Hong Kong Section, Associate Editor for the IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS–I: FUNDAMENTAL THEORY AND APPLICATIONS, Secretary and Chair-elect for the IEEE Circuits and Systems Society Technical Committee on Cellular Neural Networks, and Array Computing and Distinguished Lecturer for the IEEE Circuits and Systems Society.



Kwabena A. Boahen is an Associate Professor in the Bioengineering Department at Stanford University. He is a bioengineer who is using silicon integrated circuits to emulate the way neurons compute, linking the seemingly disparate fields of electronics and computer science with neurobiology and medicine. His contributions to the field of neuromorphic engineering include a silicon retina that could be used to give the blind sight and a self-organizing chip that emulates the way the developing brain wires itself up. His scholarship is widely recognized, with over sixty publications to his name, including a cover story in the May 2005 issue of Scientific American. He has received several distinguished honors, including a Fellowship from the Packard Foundation in 1999, a CAREER award from the National Science Foundation in 2001, a Young Investigator Award from the Office of Naval Research in 2002, and the National Institutes of Health Directors Pioneer Award in 2006. Professor Boahens BS and MSE degrees are in Electrical and Computer Engineering, from the Johns Hopkins University, Baltimore MD (both earned in 1989), where he made Tau Beta Kappa. His PhD degree is in Computation and Neural Systems, from the California Institute of Technology, Pasadena CA (1997), where he held a Sloan Fellowship for Theoretical Neurobiology. From 1997 to 2005, he was on the faculty of the University of Pennsylvania, Philadelphia PA, where he held the Skirkanich Term Junior Chair.