A Digital Neurosynaptic Core Using Event-Driven QDI Circuits

Nabil Imam[1,2,3], Filipp Akopyan[2], John Arthur[2], Paul Merolla[2], Rajit Manohar[1], Dharmendra S Modha[2]
[1]Cornell University, Ithaca, NY
[2]IBM Research - Almaden, San Jose, CA
[3]ni49@cornell.edu

***Abstract*—We design and implement a key building block of a scalable neuromorphic architecture capable of running spiking neural networks in compact and low-power hardware. Our innovation is a configurable *neurosynaptic core* that combines 256 integrate-and-fire neurons, 1024 input axons, and 1024x256 synapses in 4.2mm^2 of silicon using a 45nm SOI process. We are able to achieve ultra-low energy consumption 1) at the circuit-level by using an asynchronous design where circuits only switch while performing neural updates; 2) at the core-level by implementing a 256 neural fanout in a single operation using a crossbar memory; and 3) at the architecture-level by restricting core-to-core communication to spike events, which occur relatively sparsely in time. Our implementation is purely digital, resulting in reliable and deterministic operation that achieves for the first time one-to-one correspondence with a software simulator. At 45pJ per spike, our core is readily scalable and provides a platform for implementing a wide array of real-time computations. As an example, we demonstrate a sound localization system using coincidence-detecting neurons.**

I. Introduction

Neural systems in biology [1] are capable of an incredible range of real-time computations with metabolic constraints that require them to maintain strict energy efficiency. Tasks such as pattern recognition, sensory reconstruction and motor pattern generation are carried out by these dense, low-power neural circuits much more efficiently than traditional computers. In these systems nerve cells called neurons are the basic computational units. Neurons communicate with one another through the generation and modulation of spike trains where an individual spike is an all-or-nothing pulse. The junction between the output of one neuron and the input of another is called a synapse. The human brain consists of a staggering number of neurons and synapses—over a 100 billion neurons and over 1 trillion synapses.

While the simulation of large-scale brain-like networks has become feasible with modern supercomputers [2], the power and space they require prevent them from being useful in mobile systems for real-world tasks. On the other hand, VLSI implementations of these networks—referred to as neuromorphic chips [3]—can approach the area and power efficiency of their biological counterparts and can therefore be used for a wide range of real-time applications involving machine perception and learning.

Traditionally, neuromorphic designs used continuous-time analog circuits to model biological components, and digital asynchronous circuits for spike communication [4]. Analog circuits have been popular in the past, since they are compact, and reduce power consumption by directly using the I-V relationship of transistors to mimic the dynamics of neurons. Dense analog circuits however are sensitive to fabrication process variations, ambient temperatures and noisy environments, making it difficult to configure circuits that operate reliably under a wide range of external parameters. This limited correspondence between what the software (the neural algorithm) has been configured to do and how the hardware (the analog implementation) functions is an obstacle to algorithm development and deployment and therefore limits the usefulness of these chips. In addition the lack of high-density capacitors and increasing sub-threshold currents in the latest fabrication technologies make analog implementations even more difficult and unreliable.

In contrast to the continuous-time operation of analog circuits, the discrete-time operation of digital circuits can also be used to replicate neural activity. In fact, discrete-event simulations are the primary method of study in computational neuroscience research [5]. In this paper we introduce a purely digital implementation of a neuromorphic system. Using low-power event-driven circuits and the latest process nodes we overcome the problems of analog neuromorphic circuits without sacrificing area and power budgets. The operation of the digital implementation is completely deterministic, producing one-to-one correspondence with software neural simulators, thereby ensuring that any algorithm developed in software will work in hardware despite process variability.

Deterministic operation of brain-like networks can also be achieved on digital commodity chips, namely a DSP, a GPU or a FPGA. However the parallel and event-driven nature of these networks is not a natural fit to the sequential processing model of conventional computer architectures. A large bandwidth is necessary to communicate spikes between the physically separated processor and memory in these architectures, leading to high power consumption and limited scaling. In contrast, we integrate a crossbar synapse array with our neurons resulting in a tight locality between memory (the synapses) and computation (the neurons). The asynchronous design methodology that we use fits naturally to the distributed processing of neurons and ensures that power dissipation of inactive parts of the system are kept at a minimum. Our quasi-delay-insensitive (QDI) [6] implementation leads to extremely robust circuits that remain operational under a wide range of process, voltage and temperature variations, making them ideally suited to mobile, embedded applications.

As our main contribution, we present the design and implementation of a scalable asynchronous *neurosynaptic core*. In this paper we discuss: (i) the asynchronous circuits that mimic central elements of biological neural systems; (ii) an architecture that integrates computation, communication and memory; (iii) the asynchronous communication infrastructure required to accomodate the architecture; and (iv) the synchronization mechanisms required to maintain a one-to-one correspondence with software (this is the first neuromorphic system to demonstrate such an equivalence). Our prototype chip consists of a single core with 256 digital leaky-integrate-and-fire neurons, 1024 inputs, and 1024×256 programmable binary synapses implemented with a SRAM crossbar array. The entire core fits in a 4.2mm^2 footprint in IBM's 45 nm SOI process and consumes 45pJ per spike in active power.

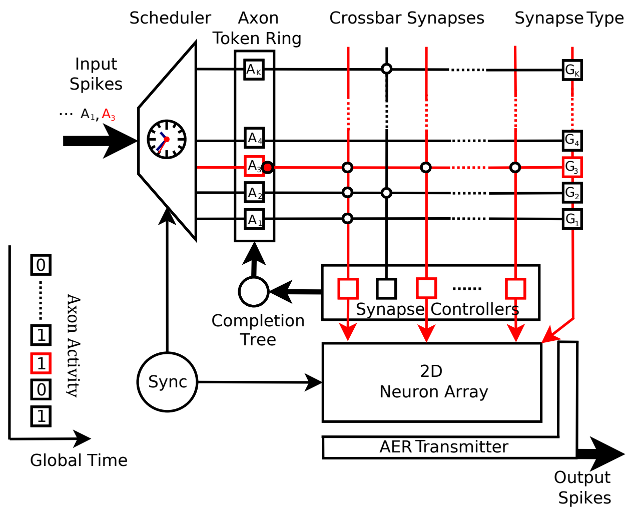
II. Architecture and Operation

A. Neurons and Synapses

The computational power of brain-like networks comes from the electrophysiological properties of individual neurons as well as their synaptic connections that form a neural network. Neurons may be modeled at various levels of biophysical detail. The leaky integrate-and-fire model is a standard approximation widely used in computational studies since it captures the behavior of real neurons in a range of situations and offers an efficient implementation. We use this neuron model as the basic computational unit of our core.

The neurons in the chip are interconnected through axons and synapses. Each axon may correspond to the output of a neuron in the same core or somewhere else in a large system of many cores. Some axons may also be driven by embedded sensors or some external driver. The connection between axon j and neuron i is represented as S_{ji}. Each axon is parameterized by a type G_j that can take one of three different values indicating the type of synapse (e.g. strong excitatory, weak excitatory or inhibitory) that the axon forms with neurons it connects to. Each neuron is parameterized by a leakage current L, a spike threshold θ and three synapse weights W^0, W^1, W^2 that correspond to the different axon types. All these parameters are configurable during start-up.

The core implements a discrete-event simulation where the neuron states are updated at each timestep according to external input and interconnectivity. The state of neuron i at some time t, is represeted by its voltage $V_i[t]$, while the state of axon j is represented by its activity bit $A_j[t]$.



NAME	DESCRIPTION	SIZE AND TYPE
V_i	Voltage of Neuron i	10 bit signed variable
$W_i^{0..2}$	3 Synaptic Weights of Neuron i	9 bit signed constant
L_i	Leak of Neuron i	9 bit signed constant
θ_i	Threshold of Neuron i	8 bit unsigned constant
S_{ji}	Connection between axon j and neuron i	Binary constant
G_j	Type of Axon j	3 distinct constants
A_j	State of Axon j	Binary Variable

Fig. 1. *Top:* Architecture of the neurosynaptic core with K axons and N neurons. Each junction in the crossbar representes a synapse between an axon (row) and dendrite (column). Each neuron has a dedicated column in the crossbar. Active synapses are represented by an open circle in the diagram. An example sequence of events in the core is illustrated. The scheduler accepts an incoming address event and communicates with the axon token-ring. The token-ring activates axon 3 (A3) by asserting the third wordline of the SRAM crossbar array. As a result, a synaptic event of type G3 is delivered to neurons N_1, N_3, and N_M. The AER transmitter sends out the addresses of these neurons if they consequently spike. *Bottom:* State variables and parameters of the system. All values are represented as integers, and all constants are configurable at start-up.

The parameters and state variables of the system are tabulated in Fig. 1(bottom). Neuron i receives the following input from axon j:

$$A_j[t] \times S_{ji} \times W_i^{G_j}.$$

The neuron's voltage is updated at each time step by subtracting a leakage from its voltage and integrating the synaptic input from all the axons:

$$V_i[t+1] = V_i[t] - L_i + \sum_{j=1}^{1024} (A_j[t] \times S_{ji} \times W_i^{G_j}).$$

When $V[t]$ exceeds a threshold θ, the neuron produces a spike (represented by a digital '1' in its output) and its voltage is reset to 0. We also enforce that negative voltages are clipped back to 0 at the end of each time step (to replicate a reversal potential of 0).

B. Communication Infrastructure

At the heart of the neurosynaptic core is a crossbar memory that forms the synapses between axons and the neurons. The crossbar array is configurable so that arbitrary networks can be set up in the system (e.g. Axons 1, 2, and 3 are connected to the first neuron in the 2D neuron array in Fig. 1). Each row of the crossbar corresponds to an axon, each column corresponds to the input of a neuron (the dendrite), and the junctions are binary synapses implemented by a two terminal memory cell (e.g., SRAM). Thus each of the N neurons may get up to K synaptic inputs depending on the activity in the axons and the configuration of the crossbar. We chose K as 1024 and N as 256 resulting in 1024×256 crossbar synapses and an enormous configuration space.

Spikes events are sent to and from the core using *address-event representation* (AER) packets [7]. On the output side, an AER transmitter [8] encodes spiking activity by sending the locations of active neurons through a multiplexed channel, leveraging the fact that the bandwidth of wires (easily larger than 100s of MHz) is orders of magnitude larger than the bandwidth of biological axons (in the 10's of Hz range). The spikes can be sent off chip, or routed to an axon of another core via a look-up table. On the input side, an AER receiver delivers incoming spikes to the appropriate axon at a predetermined time configured in a scheduler block. As spikes are serviced sequentially, their addresses are decoded to the crossbar where all 256 synaptic connections of an active axon are read out in parallel.

C. Discrete-time Operation

An example sequence of operation in the core is illustrated in Fig. 1. The operation has two phases during each time step.

The positive edge of a global synchronization clock (Sync) initiates the first phase of operation. In this phase, address-events along with their time stamps are sent to the core and are received by the scheduler. The scheduler evaluates the time stamps and asserts the appropriate axons that go into a token-ring. The units in the token-ring that receive active axons assert the rows of the crossbar in a mutually exclusive manner. Once a wordline in the crossbar is activated all the neurons that are connected to the axon (corresponding to the 1's in the row) receive an input spike along with information about the type of the axon. The neurons update their voltages as axon events come in. The first phase needs to complete within the first half of the global synchronization clock (that usually has a period of 1 millisecond)—giving us a precise margin in which neural updates need to complete for potentially all 1024 axon inputs.

In the second phase of operation, the negative edge of the synchronization clock is detected by all the neurons. On receiving this event, neurons whose voltages have exceeded their respective thresholds produce spikes in their output ports. The spiking addresses are encoded by the AER transmitter and sent out of the core sequentially. This phase needs to complete within the other half of the global clock—i.e. the AER transmitter has to guarantee that it can service 256 potential spikes within the global timestep.

A 1 millisecond global clock period (typical temporal precision in biological neural networks) means that the performance requirements of the circuits in the two phases of operation are easily met. Breaking neural updates into two phases ensures that the hardware is always in sync with an equivalent software simulation at the end of each time step. Specifically, the order in which address-events arrive to the core or exit from the core can be variable due to resource arbitration, especially when events are sent through a non-deterministic routing network. To preserve one-to-one correspondence, the different orderings must not influence the spiking dynamics. We achieve one-to-one equivalence by first accounting for all the synaptic events and then checking for spikes.

III. Event-Driven Implementation

We implement the architecture of the neurosynaptic core using asynchronous QDI circuits that are synchronized with the global timestep. In this section we describe the concurrent processes of our architecture using Communicating Hardware Processes (CHP - see Appendix) that can be synthesized into QDI asynchronous circuits using Martin's synthesis method [9].

A. Scheduler

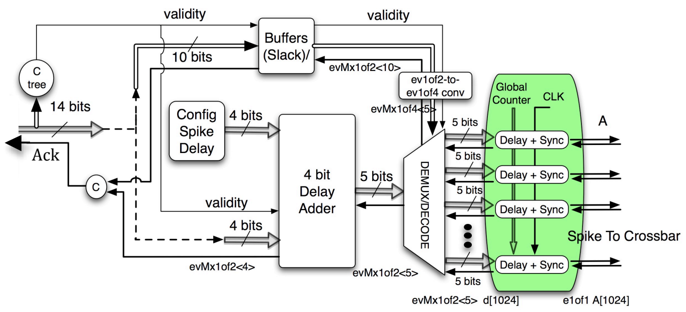


Fig. 2. Internal structure of the Scheduler. The overall function is: (1) Receive input packets; (2) Add the stored axonal delay to the time field of the packet to get the spike delivery time; (3) Demultiplex the address field of the packet; and (4) Deliver the spike to the appropriate axons at the edge of a global clock when the global timestep equals the computed delivery time of the spikes.

The scheduler receives packets from outside the core and delivers spikes on the axons at specific times. The packets may come from spiking neurons in the core or from outside the core. In addition to these packets, the scheduler also receives a clock and a global counter time. The block also stores a configurable axonal delay (inside an SRAM array) for each of the axons. Each packet (y) coming in contains an axon address ($y.dest\_addr$) and a time of spike ($y.dt$ - in units of clock ticks). The scheduler decodes the packet to determine where the spike should be delivered (the axon number). The time in the packet is added to the axonal delay for the specified axon, and this value is compared against the current global counter time on every tick of a clock that the scheduler receives. When the time matches, a spike is delivered to the crossbar. This makes the spike delivery to the crossbar synchronous with the global system clock. The CHP of the processs is:

```
$Scheduler \equiv$
    $*[[IN?y;$
        $axon\_delay := SRAM[y.dest\_addr];$
        $delay\_y := y.dt + axon\_delay;$
        $[delay\_y = timestamp];$
        $A[y.dest\_addr]!spike$
     $]]$
```

The scheduler implements the axonal delay stored in a 1024 (number of axons) by 4 (range of delays) SRAM array. It receives the packet in the channel IN, adds the axonal delay to the time in the packet, waits for the global time to reach this value and then delivers the spike to the axon corresponding to the address in the packet. Besides implementing the axonal delay, this procedure synchronizes input spikes to the core with the clock edge, implementing the first of the two phases of operation (Section II-C) that allow the system to produce 1-1 correspondence with a software simulator.

The internal blocks of the scheduler are illustrated in Fig. 2. For our prototype chip, we used a common delay block instead of a full 1024×4 SRAM array to implement the axonal delay. This fixed delay is a 4 bit number that can be configured at the chip's startup, and is the delay value for all axons. The Adder adds this delay to the time value in the packet and passes it to the DEMUX/DECODE unit. The control for the DEMUX/DECODE unit is the destination address in the input packet, slack-matched to the SRAM and the adder. The DEMUX then outputs 5 bits indicating the final timestamp for spike release to the axon. This value goes to 1 of 1024 [Delay + Sync] blocks. These blocks must contain full buffers at the input because all the hardware used prior to this stage is time-shared and the DEMUX output channel has to be "released" as soon as possible to allow assertion of the next spike. Each [Delay + Sync] unit compares the timestamp of the global timer with the obtained final timestamp and initiates a spikes to an axon when the value matches.

The clock synchronization is contained in the [Delay + Sync] unit. The synchronization part is not trivial and requires some design effort. The data obtained from the DEMUX/DECODE must be aligned to the clock without any knowledge of the data arrival time in relationship to the clock edge. Precautions have to be taken in order to avoid potential metastability during synchronization with the global clock. Such metastbility may occur since there is no timing correlation between the edge of the global clock and the packets arriving from the router. A modified two flip-flop synchronization scheme is used in our design [10].

B. Axon Token-Ring

At an edge of the synchronizing clock, the scheduler pulls up the axon lines that have spikes. Each axon has a corresponding row (a wordline) in the crossbar memory array. The dendrites (inputs) of a neuron correspond to a column (bit line) of the crossbar array. Since a dendrite can potentially connect to multiple axons, the rows of the crossbar have to be asserted in a mutually exclusive manner. This function is carried out by an array of axon servers that implement a token-ring mutual exclusion algorithm [11]. Each server has an axon as its input and the wordline of the crossbar as its output. A token circulates among the servers to give them mutually exclusive access to the crossbar.

When an axon is asserted, its server requests its neighbor to pass the token. The request propagates through the token-ring, and the token is passed along to the requesting server. Upon the arrival of the token, the server asserts the corresponding row of the crossbar. The CHP of an individual server is given below. Channel A communicates with the axon from the scheduler and channel WL with the crossbar, while channels U and D communicate with the neighbor above or below. The local variable b represents the token. The D port of the last server is connected to the U port of the first server. The channel C communicates with a completion tree (see Fig. 1) that indicates when all the neurons have completed processing events for a particular axon.

```
$Axon\_Server \equiv$
    $*[[\overline{A} \longrightarrow [b \longrightarrow skip \ [] \neg b \longrightarrow D!; b\uparrow]; WL!;$
                  $[\overline{C} \longrightarrow C; A?]$
      $| \overline{U} \longrightarrow [b \longrightarrow skip \ [] \neg b \longrightarrow D!]; b\downarrow; U?$
     $]]$
```

C. Crossbar Memory

The states of the memory cells of the crossbar array represent the synaptic configuration of a network (i.e. which axons connect to which neurons) and determine the unique properties of that particular network. Organizing the synapses in this crossbar structure allows an active axon to fan out to potentially 256 neurons in parallel through a single control operation. For large connectivity, this reduces the dynamic power consumption and accelerates the speed at which the network is updated.

The configuration of the crossbar has to be set up prior to the operation time of the chip. We included shift register scanners to configure the bit cells of the array. Standard 6T SRAM cells were used as the bit cells. The axon token-ring controls the wordlines of the bit cells while a set of synapse controllers interfaces the bitlines with the neurons. The CHP of a synapse controller unit is given below.

```
$Synapse\_Controller\_Unit \equiv$
    *[[$BL.t \longrightarrow N!;\ \ C!$
      ▯ $BL.f \longrightarrow\ \ C!$
     ]]
```

When the axon token-ring drives one of the wordlines of the crossbar, one of the two bitlines of each cell in the corresponding row will discharge asserting either the $BL.t$ or the $BL.f$ wires of the synapse controller. If $BL.t$ is asserted the controller communicates with the neuron corresponding to the column to update the neuron's state. Once this communication is over, or if $BL.f$ was the wire originally asserted, the controller communicates with a completion tree. The right-most controller receives the "type" information for each axon (they are stored in the last 2 bit cells of each crossbar row) and communicates this information to the neurons. Once all synapse controllers have completed their operation, the completion tree communicates wih the C channel of the axon token-ring, after which the token-ring unit with the token releases the wordline of the crossbar and token becomes free to travel. Another token-ring unit with an active axon will then get the token and assert its corresponding wordline.

D. Neuron

The neurons are the basic computational units in the system. They were implemented using dedicated circuits (non-multiplexed), allowing all the neural updates to happen in parallel. This parallelism comes at the cost of relative density inefficiency that is in part mitigated by our use of a dense technology.

The design of the neuron circuits needs to accomodate two conflicting requirements. On the one hand, a neuron must service events quickly to avoid holding up the crossbar. On the other hand, a neuron receives very few events in typical scenarios, so it must not burn dynamic power when it is idle. A purely event-driven design is therefore ideal: it has a fast service time (100MHz-GHz range), but only burns power during the rare occurrence of an event.

During the first phase of operation (Section. II-C) each neuron receives event-driven synaptic input from the crossbar memory synapse controller. The neurons update their state (represented by an internal voltage V) by integrating the incoming synapses (Section. II-A). During the second phase of operation, the neurons synchronize with the global clock edge, at which point they output a spike (represented by a 1 bit output) if their voltage is above threshold or leak out an amount of the voltage if it is below threshold. The synchronization of the neuron during the second phase of operation, along with the synchronization of the scheduler during the first phase (Section. III-A) ensures that the operation in the core is in 1-1 correspondence with a software simulator.

The internal parameters of the neurons (the threshold, the leak, and the three synaptic strengths) are configured at start-up. They are all represented with 8-bit integers in 2's compliment form. The internal voltage of a neuron is represented by a 10-bit integer also in 2's compliment form.

The operation of the neuron is decomposed into control and datapath blocks. A block diagram of the processes involved is illustrated in Fig. 3.

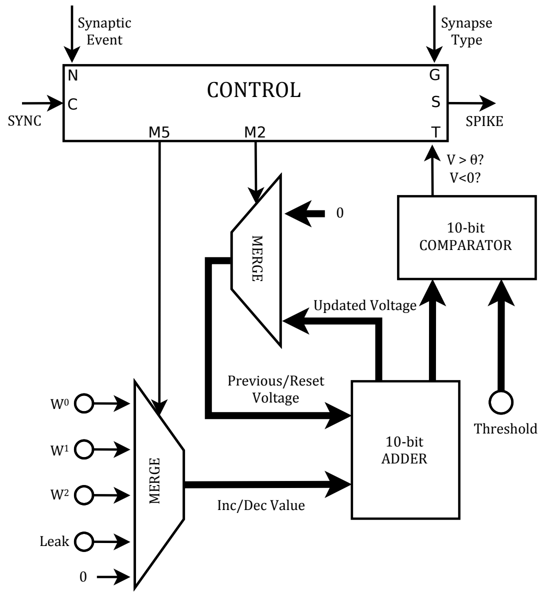


Fig. 3. Block diagram of the neuron. The control block interfaces with the crossbar, directs information flow in the datapath, synchronizes with the global time step and outputs a spike when the neuron voltage exceeds its threshold. The datapath elements update the voltage after each synaptic event and check for a spike when directed by the control.

The CHP of the control block is:

```
$Neuron\_Control \equiv$
    *[[$\overline{N} \longrightarrow G?x;$   [$x = Type0 \longrightarrow M5!0$
                          ▯$x = Type1 \longrightarrow M5!1$
                          ▯$x = Type2 \longrightarrow M5!2$
                          ], $M2!0; N$
      ▯$\overline{C} \longrightarrow\ \ T?y;$  [$y = spike \longrightarrow S,\ \ M5!3,$
                                  $M2!1$
                          ▯$y! = spike \wedge (y <= 0) \longrightarrow$
                                  $M5!4, M2!1$
                          ▯$y! = spike \wedge (y > 0) \longrightarrow$
                                  $M5!3, M2!0$
                          ]; $C$
     ]]
```

The control block interfaces with the synaptic input coming from the neuron's dedicated crossbar column through the channel N. Upon a communication request on this channel, the control reads in the "type" information of the axon through the channel G that connects to the synapse controller representing the last two columns of the crossbar. Before completing the handshake in N the control communicates with the datapath of the neuron through channels $M5$ (that relays synaptic type information) and $M2$ (for voltage control in the datapath). Once these communication actions have completed N and G are acknowledged and the next cycle of synaptic inputs may come in.

In the second half of the global timestep, the neurons receive a synchronization event in the channel C (that is driven by a process that uses one of the edges of the global clock to initiate a handshake). When this event comes in, the control initiates a communication with the datapath through the T channel. The datapath sends back one of three distinct values indicating whether the voltage for the current timestep is above threshold, below threshold but above zero; or below zero. If the voltage is above threshold the control communicates with the AER transmitter through the channel S to send out a spike. Through $M2$ the control also instructs the datapath to reset the voltage to 0 if there is a spike or if the voltage is below 0. If the voltage has not been reset to 0, the control instructs the datapath to apply the leak to the current voltage.

The CHP of the datapath units are:

```
$MERGE\_5 \equiv$
    *[[$\overline{M5} \longrightarrow M5?d$;
              [$d = 0 \longrightarrow AI1!W^0$
              ▯$d = 1 \longrightarrow AI1!W^1$
              ▯$d = 2 \longrightarrow AI1!W^2$
              ▯$d = 3 \longrightarrow AI1!L$
              ▯$d = 4 \longrightarrow AI1!0$
              ];
    ]]
$MERGE\_2 \equiv$
    *[[$\overline{M2} \longrightarrow M2?d$;
                  [$d = 0 \longrightarrow AO0?x;\quad AI0!x$
                  ▯$d = 1 \longrightarrow AI0!0$
                  ];
    ]]
$ADDER \equiv$
    *[$\overline{AI0} \wedge \overline{AI1} \longrightarrow AI0?x,\ \ AI1?y;\ \ z := x + y$
     ▯$\overline{AO0} \longrightarrow AO0!z$
     ▯$\overline{AO1} \longrightarrow AO1!z$
     ]]
$COMPARATOR \equiv$
    *[$\overline{T} \longrightarrow AO1?V$;
          [$V > Threshold \longrightarrow T!0$
          ▯$(V < Threshold) \wedge (V <= 0) \longrightarrow T!1$
          ▯$(V < Threshold) \wedge (V > 0) \longrightarrow T!2$
          ]; $T$
    ]
```

When the control drives $MERGE\_5$, the process forwards either a synaptic strength (during the integration phase), a leak (during the resetting phase if $V > 0$) or the value 0 (during the resetting phase if $V <= 0$) to one of the inputs of the $ADDER$ process. When the control drives $MERGE\_2$, the process forwards either the previous voltage (during integration) or the value 0 (if $V > threshold$ or $V < 0$ during the firing phase) to the other input of $ADDER$. The $ADDER$ process is a 10 bit adder that sends out the sum of its inputs to the $COMPARATOR$ and the $MERGE\_2$ processes when they request it. The control drives $COMPARATOR$ when it needs to evaluate the state of the neuron voltage.

E. AER transmitter

Spikes from the 2-dimensional array of neurons are sent out of the core through token-ring AER transmitter circuits [8] that allow all the neurons to share one output channel. In this scheme, a set of row servers and column servers circulate tokens in each dimension of the neuron array and give spiking neurons mutually exclusive access to the shared communication channel. A counter keeps track of the location of the tokens, and sends out neuron addresses upon request. This methodology leads to compact transmitter circuits capable of efficiently servicing clusters of spiking activity.

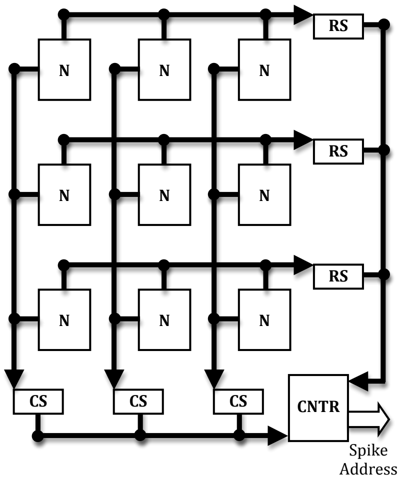


Fig. 4. Illustration of the AER transmitter architecture using a 3×3 sample neuron array. When a neuron (N) has a spike, it communicates with its corresponding row server (RS) and column server (CS). A counter (CNTR) keeps track of circulating row and column tokens and sends out the spike address through the shared output bus.

The design of the transmitter is illustrated in Fig 4. The sequence of operation is: (1) A spiking neuron asserts a row request line; (2) The corresponding row server requests for the row token from its neighbors; (3) As the row token moves, the counter keeps track of its position; (4) Upon receipt of the token, the row server acknowledges the row request; (5) The neuron then asserts a column request line; (6) The corresponding column server requests the column token from its neighbors; (7) As the column token moves, the counter keeps track of its position; (8) Upon receipt of the token the column server communicates with the counter to send out the row and column token addresses and then acknowledges the column request; (9) The neuron does a second communication with the row server to indicate the completion of service.

The neurons interface with their respective servers via open-drain shared row request lines and shared column request lines. The servers also communicate with the counter via shared wires. These wires need to be carefully implemented since transitions are shared across processes and span an entire dimension of the neuron array.

IV. Results

The neurosynaptic core was fabricated using IBM's 45nm SOI process and occupied $4.2mm^2$ of silicon (Fig. 5, left). Each of the 256 neurons in the core occupies 35μm $\times$ 95μm. Each SRAM bitcell in the 1024×256 synaptic crossbar array occupied $1.3\mu m^2$ (plus another 1.9 μm^2 associated with conservatively-designed periphery). A custom printed circuit board allows the chip to interface with a PC through a USB link (Fig. 5, right).

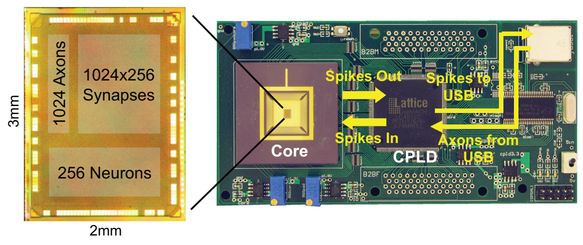


Fig. 5. *Left:* Neurosynaptic die measures 2mm $\times$ 3 mm including the I-O pads. *Right:* Test board that interfaces with the chip via a USB 2.0 link. Spike events are sent to a PC for data collection and can also be routed back to the chip via the CPLD.

A. Active Power

Our primary focus during the design was the reduction of active power since passive power can be addressed through fabrication options and active leakage reduction circuitry [13]. The purely event-driven nature of the core allowed us to achieve an ultra-low-power design where active power is dependent only on the activity rate of the neurons. Our QDI methodology allows us to readily reduce the operating voltage in the core without affecting the correctness of the neural algorithms that it is running. We discuss the power consumption of the chip in Ref. [14]. At V_{dd} = 0.85, the core consumes just 45pJ/spike.

B. Example Application

The neurosynaptic core produces one-to-one equivalence with models in software neural simulators making it convenient to set up neural algorithms on the chip. A wide range of traditional artificial neural network algorithms [15] for various applications can readily be configured. In addition, the biologically-grounded neuron models in our chip allow novel bio-inspired approaches to be implemented for solving engineering problems. For example, individual neurons in the core can efficiently implement a common processing technique in biological neural pathways—coincidence detection—that does not have an analog [16] in the computational units of traditional artificial neural networks. In Fig. 6 we illustrate a bio-inspired sound-localization method in our chip using this technique.

V. Conclusion

We have demonstrated a compact modular architecture that can be a building block for large-scale neuromorphic systems. Our asynchronous neurosynaptic core combines digital neurons, large synaptic fanout, and address-event communication circuits in a low-power event-drive fabric. The embedded crossbar array allows us to implement synaptic fanout without resorting to off-chip memory that can create I-O bottlenecks. The asynchronous quasi-delay-insensitive circuits that we used led to robust circuits that are operational across a wide range of voltages and temperatures, making the neurosynaptic core ideally suited for mobile applications. Our implementation methodology guaranteed that there is strict correspondence between the neural algorithm running on the core and an equivalent simulation of it in a software simulator, greatly increasing the usability of the chip and its deployment in real-world scenarios.

Appendix

The CHP notation we use is based on Hoare's CSP [17]. A full description of CHP and its semantics can be found in [9]. What follows is a short and informal description.

- Assignment: $a := b$. This statement means "assign the value of b to a." We also write $a\!\uparrow$ for $a := true$, and $a\!\downarrow$ for $a := false$.
- Selection: $[G1 \rightarrow S1 \,\square\, \ldots \,\square\, Gn \rightarrow Sn]$, where G_i's are boolean expressions (guards) and S_i's are program parts. The execution of this command corresponds to waiting until one of the guards is $true$, and then executing one of the statements with a $true$ guard. The notation $[G]$ is short-hand for $[G \rightarrow skip]$, and denotes waiting for the predicate G to become true. If the guards are not mutually exclusive, we use the vertical bar "|" instead of "$\square$."
- Repetition: $*[G1 \rightarrow S1 \,\square\, \ldots \,\square\, Gn \rightarrow Sn]$. The execution of this command corresponds to choosing one of the $true$ guards and executing the corresponding statement, repeating this until all guards evaluate to $false$. The notation $*[S]$ is short-hand for $*[true \rightarrow S]$.
- Send: $X!e$ means send the value of e over channel X.
- Receive: $Y?v$ means receive a value over channel Y and store it in variable v.
- Probe: The boolean expression $\overline{X}$ is $true$ iff a communication over channel X can complete without suspending.
- Sequential Composition: $S; T$
- Parallel Composition: $S \parallel T$ or S, T.

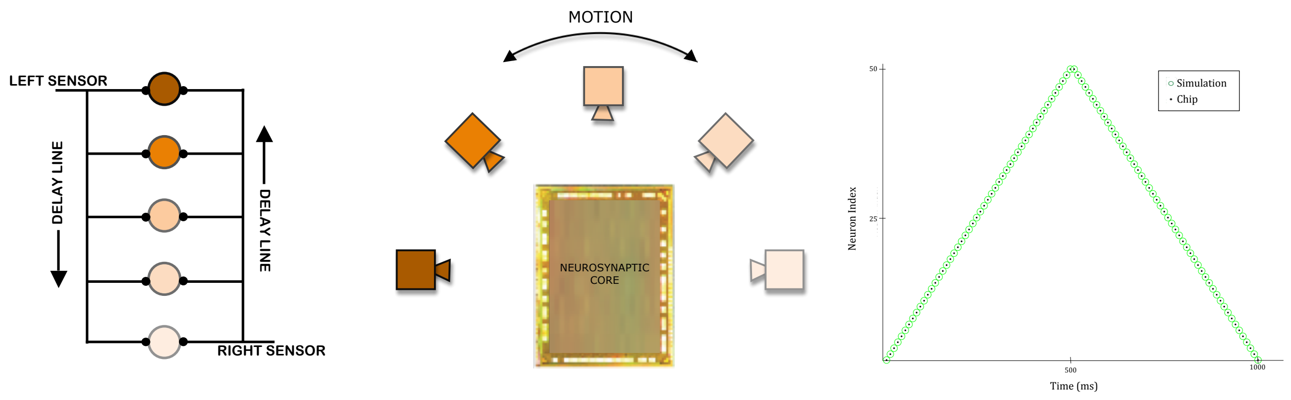


Fig. 6. Sound localization using axonal delays and coincidence detection. Coincidence detecting neurons spike when their inputs come within a time window defined by the neurons' internal parameters. *Left:* There is a difference in the arrival times of a sound in two sensors located at opposite sides of a system (e.g. two ears). Hence the sensors contain information about the location of the sound source. This information is decoded by a set of coincidence-detecting neurons (circles) arranged systemically along axonal delay lines. For example, the top-most neuron has a short delay line from the left sensor but a long delay line from the right sensor. When the sound source is exactly adjacent to the right sensor, the axonal delay in the right sensor path compensates for the time lag between the arrival of the sound in the two sensors. The neuron would thus receive coincident inputs and respond maximally, decoding the location of the sound source. Avian auditory pathways use this mechanism for sound localization [12]. *Center:* The neurosynaptic core can use a similar method for localizing a sound source based on the inputs of two sensors (not shown). The sensors would have to receive the sound and convert them into address-event packets. The first input (corresponding to the sensor closer to the sound source) will hit several axon lines that implement different axonal delays using the scheduler (in our prototype system, the operation of the scheduler is replicated outside the chip). Each neuron in the core will connect to one of these axon lines. The second input will hit a separate axon line that connects to all neurons. Those neurons that have temporally coincident inputs will spike maximally, representing the location of the sound source in neural space. The difference between the input arrival times in the two sensors will typically be in the 10s of microseconds range. The sensors may amplify this difference to keep the core at its usual (millisecond) precision range, or the time step in the core may be made more precise. *Right:* The dynamics of the chip when the axon lines are driven in the manner suggested. The sound source starts at the left of the core, moving all the way to the right in a semi-circular trajectory, and then back to its original position. As the source moves, a unique neuron spikes to indicate the new location. A total of 50 neurons were included to identify 50 distinct positions. A software simulation running the same algorithm confirms that our chip is in 1-1 correspondence.

ACKNOWLEDGMENTS

This research was sponsored by DARPA under contract No. HR0011-09-C-0002. The views and conclusions contained herein are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of DARPA or the U.S. Government. Authors thank William Risk and Sue Gilson for project management, Ben Hill, Ben Johnson, Rob Karmazin, Carlos Otero, Jon Tse, Scott Fairbanks, Scott Hall, Kavita Prasad for physical design, Steven Esser and Greg Corrado for their work on digital neurons, and Andrew Cassidy and Rodrigo Alvarez for developing software for data collection.

REFERENCES

[1] E. R. Kandel, J. H. Schwartz, and T. M. Jessell, *Principles of Neural Science*. McGraw-Hill Medical, 4th ed., July 2000.

[2] R. Ananthanarayanan, S. K. Esser, H. D. Simon, and D. S. Modha, "The cat is out of the bag: cortical simulations with 10^9 neurons, 10^{13} synapses," in *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*, SC '09, (New York, NY, USA), pp. 63:1–63:12, ACM, 2009.

[3] C. Mead, "Neuromorphic electronic systems," *Proceedings of the IEEE*, vol. 78, pp. 1629 –1636, Oct. 1990.

[4] K. Boahen, "Neurogrid: emulating a million neurons in the cortex," *IEEE international conference of the engineering in medicine and biology society*, 2006.

[5] P. Dayan and L. F. Abbott, *Theoretical Neuroscience: Computational and Mathematical Modeling of Neural Systems*. The MIT Press, 2005.

[6] A. J. Martin, "Programming in VLSI: From communicating processes to delay-insensitive circuits," in *Developments in Concurrency and Communication, UT Year of Programming Series* (C. A. R. Hoare, ed.), pp. 1–64, Addison-Wesley, 1990.

[7] M. A. Mahowald, *VLSI analogs of neuronal visual processing: a synthesis of form and function*. PhD thesis, Pasadena, CA, USA, 1992. UMI Order No. GAX92-32201.

[8] N. Imam and R. Manohar, "Address-event communication using token-ring mutual exclusion," in *ASYNC*, pp. 99–108, IEEE Computer Society, 2011.

[9] A. J. Martin, "Programming in VLSI: From communicating processes to delay-insensitive circuits," in *Developments in Concurrency and Communication, UT Year of Programming Series* (C. A. R. Hoare, ed.), pp. 1–64, Addison-Wesley, 1990.

[10] F. Akopyan, *Hybrid Synchronous/Asynchronous Design*. Ph.D. thesis, Cornell University, April 2011.

[11] A. J. Martin, "Distributed mutual exclusion on a ring of processes," *Sci. Comput. Program.*, vol. 5, pp. 265–276, October 1985.

[12] B. Grothe, "New roles for synaptic inhibition in sound localization," *Nature Reviews Neuroscience*, vol. 4, pp. 1–11, 2003.

[13] C. T. O. Otero, J. Tse, and R. Manohar, "Static power reduction techniques for asynchronous circuits," in *IEEE International Symposium on Asynchronous Circuits and Systems*, May 2010.

[14] P. Merolla, J. Arthur, F. Akopyan, N. Imam, R. Manohar, and D. Modha, "A digital neurosynaptic core using embedded crossbar memory with 45pJ per spike in 45nm," *Proceedings of the IEEE Custom Integrated Circuits Conference*, September 2011.

[15] S. Haykin, *Neural Networks and Learning Machines (3rd Edition)*. Prentice Hall, 3 ed., Nov. 2008.

[16] W. Maass and C. M. Bishop, eds., *Pulsed neural networks*. Cambridge, MA, USA: MIT Press, 1999.

[17] C. A. R. Hoare, "Communicating sequential processes," *Communications of the ACM*, vol. 21, no. 8, pp. 666–677, 1978.