RETINOMORPHIC VISION SYSTEMS II: COMMUNICATION CHANNEL DESIGN

Kwabena Boahen

Physics of Computation Laboratory
California Institute of Technology
MS 136-93, Pasadena, CA 91125, USA
buster@pcmp.caltech.edu

ABSTRACT

I discuss the tradeoffs faced when asynchronous pulse trains are transmitted among large, two-dimensional, arrays of neurons on different chips, using time-division multiplexing, and present an implementation of an arbitered, random-access, channel. The long cycle time that plagues arbitered channels is addressed in the implementation described here by pipelining. Cycle times ranging from 420ns to 730ns were achieved, for 64×64 arrays, in a 2μm CMOS process, yielding a peak throughput of 2.38M spikes/second.

1. CHANNEL-DESIGN TRADEOFFS

The two-chip neuromorphic system shown in Figure 1 uses an interchip communication channel to transmit spike-trains between neurons at corresponding locations on each chip. The channel performance may be rated according to the following criteria:

Capacity: The maximum rate at which spikes can be transmitted. It is equal to the reciprocal of the minimum communication cycle period.

Latency: The mean time a spike spends in transit between a neuron in the sending population and a neuron in the receiving population.

Temporal Dispersion: The standard deviation of the channel latency.

Integrity: The fraction of spikes that are delivered to the correct destination.

All four criteria together determine the **throughput**, which is defined as the usable fraction of the channel capacity, because the load offered to the channel must be reduced to achieve more stringent specifications for latency, temporal dispersion, and integrity. I discuss tradeoffs between adaptive quantization versus fixed quantization, and between arbitration versus free-for-all in Sections 2 and 3, respectively. Then, I describe an implementation for a pipelined, arbitered, random-access channel in Section 4, and present test results from a working multichip neuromorphic system that uses this communication channel in Section 5. My conclusions are in Section 6.

2. ADAPTIVE VERSUS FIXED

We are given a desired sampling rate f_{Nyq} and an array of N signals to be quantized. We use adaptive, 1-bit, quantizers that sample at f_{Nyq} when the signal is changing, and sample at f_{Nyq}/Z when the signal is static. Let the probability that a given quantizer samples at f_{Nyq} be a. That is, a is the fraction of the quantizers whose inputs are changing. Then, each quantizer generates bits at the rate

$$f_{\mathrm{bits}} = f_{\mathrm{Nyq}}(a + (1-a)/Z)\log_2 N,$$

because a percent of the time, it samples at f_{Nyq}; the remaining $(1-a)$ percent of the time, it samples at f_{Nyq}/Z. Furthermore, each time that it samples, $\log_2 N$ bits are sent to encode the location. On the other hand, if we simply sample each location sequentially at the desired rate f_{Nyq}, and do not locally adapt the sampling rate, we obtain a bit rate of f_{Nyq} per quantizer, because there is no distinction between active and passive quantizers, and there is no need to encode location explicitly—the location may be inferred from the position of the sample in the sequence. Therefore, adaptive quantization is more efficient if

$$a < (Z/(Z-1))(1/\log_2 N - 1/Z)$$

For example, in a 64×64 array with sampling-rate attenuation, Z, of 40, the active fraction, a, must be less than 6.1 percent. In a retinomorphic system, the adaptive neuron circuit performs sampling-rate attenuation, and the spatiotemporal bandpass filter makes the output activity sparse [1, 2].

Given a certain fixed channel throughput (F_{chan}), in samples per second, we can ask what effective sampling rate f_{Nyq} each strategy achieves. For adaptive quantization, channel throughput is allocated dynamically in the ratio $a : (1-a)/Z$ between the active and passive fractions of the user population. Hence,

$$f_{\mathrm{Nyq}} = f_{\mathrm{chan}}/(a + (1-a)/Z)$$

where $f_{\mathrm{chan}} \equiv F_{\mathrm{chan}}/N$ is the throughput per user. In contrast, fixed quantization achieves only f_{chan}. For instance, if

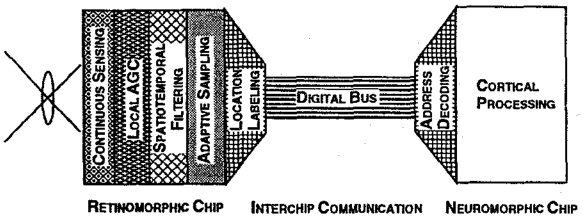

Figure 1. System concept. The retinomorphic chip quantizes the image, locally adapting its sampling rate. The resulting digital pulses (spike trains) are read out from the pixel, transmitted to another neuromorphic chip, and recreated at the corresponding location, by the interchip-communication channel.

0-7803-3073-0/96/$5.00 ©1996 IEEE

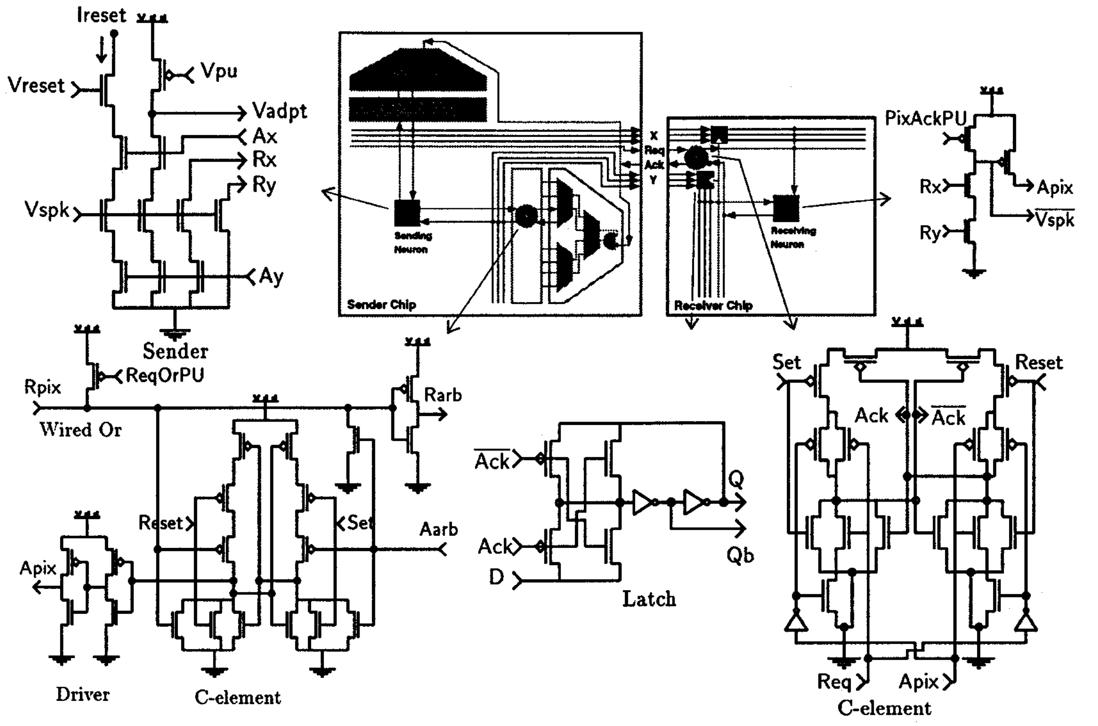

Figure 2. Pipelined, random-access, interchip-communication channel with a single sender and a single receiver. The circuits shown, proceeding counter-clockwise from top-left, read out spikes from the sending pixel and reset it; synchronize the sending pixel and the arbiter with the bus; latch the addresses at the receiver; synchronize the receiving pixel with the bus; and recreate the spike in the receiving pixel.

$f_{\rm chan}$ = 100 Hz, adaptive quantization achieves $f_{\rm Nyq}$ = 1.36 KHz, with an active fraction of 5 percent and a sampling-rate attenuation factor of 40. Thus, a two-decade increase in temporal bandwidth is achieved under these conditions; the channel latency is also reduced by the same factor.

In order to use adaptive quantization, we must perform quantization at the pixel level, provide random access to the shared communication channel, and allow pixels to transmit at will. We also need to guarantee that pixels behave intelligently and, in particular, deal with the collisions that occur when two or more pixels attempt to transmit simultaneously.

3. ARBITRATION VERSUS FREE-FOR-ALL

The original implementation of a random-access (RA) protocol included arbitration and queuing mechanisms [4] to guarantee that no spikes are lost. Queuing causes temporal dispersion when the channel is overloaded briefly, due to a burst of activity generated by external stimuli that trigger synchronous firing. Also, the arbitration procedure lengthens the communication cycle period, and reduces the channel capacity. On the other hand, if we simply allow collisions to occur and discard the corrupted addresses so generated [5], we may achieve a shorter cycle period but spike loss will increase as the load increases.

We can quantify this tradeoff using the following well-known result for the collision probability [6]:

$$p_{\rm coll} = 1 - e^{-2G},$$

where G is normalized channel load (i.e., the probability that a spike occurs during a time interval equal to the communication cycle period).[1] If we arbitrate, we will achieve a certain cycle time, and a corresponding channel capacity, in a given VLSI technology. An arbitered channel can operate at close to 100-percent capacity because the 0.86 collision probability for $G = 1$ is not a problem—users just wait their turn. Now, if we do not arbitrate, we shall achieve a shorter cycle period, with a proportionate increase in capacity. Let us assume that the cycle period is reduced by a factor of 10, which is optimistic. For the same offered load, we have $G = 0.1$, and find that $p_{\rm coll}$ = 18 percent. Thus, the simple, nonarbitered channel can transmit more spikes per second if collision rates higher than 18 percent are acceptable. For lower collision rates, the complex, arbitered channel offers more throughput, even though its cycle period is 1 order of magnitude longer. More reasonable failure probabilities of 5 percent require the nonarbitered channel to operate at only 2.5 percent of its capacity.

Indeed, the arbiterless channel must operate at high error rates to maximize utilization of the channel capacity. The channel throughput is Ge^{-2G} [6], since the probability of a successful transmission (i.e., no collision), is e^{-2G}. Throughput reaches a maximum of $0.5e^{-1} = 0.18$ when $G = 0.5$. These figures yield a loss of 64 percent of the input data. We conclude that there is no point in providing unfettered access to the channel unless this simplification reduces the cycle time by at least a factor of five.

But what about the temporal dispersion that occurs when

[1] We obtain this result by assuming independent firing probabilities and approximating the resulting binomial distribution with the Poisson distribution.

users are queued? Queuing theory tells us that [6]

$$T_{\text{avg}} = \Delta(1 - G/2)/(1 - G),$$

where Δ is the communication-cycle period. For example, at 95-percent capacity, the wait is 10.5Δ, on average. This result makes sense because every twentieth slot is empty, therefore one must wait anywhere from 1 to 20 cycles to gain access, which averages out to 10.5. For a submicrosecond cycle time, this queuing time is 2 orders of magnitude less than the 1ms resolution of a typical neuron. Therefore, throughputs close to to 100 percent of the channel capacity may be achieved without significant increases in latency or temporal dispersion. Indeed, the channel's capacity can be exceeded briefly with only a linear degradation in temporal dispersion, compared to the exponential increase in collisions that occurs when there is no arbitration.

4. CIRCUIT IMPLEMENTATIONS

The design of the pipelined, random access, communication channel is shown in Figure 2. Like the original design [4], which was called an **address-event** (AE) channel, this implementation is completely self-timed; thus, every communication must be acknowledged by a feedback signal. We also use these acknowledge signals to implement a queuing mechanism: it makes a pixel wait simply by refusing to acknowledge that pixel. At the beginning of a communication cycle, the request and acknowledge signals are both low. The communication cycle proceeds as follows.

On the sender side, spiking neurons first make requests to the Y arbiter, which selects only one row at a time. It does so in a hierarchical fashion, using a decision tree that is built from two-input arbiter cells [4, 3]. All spiking neurons in the selected row then make requests to the X arbiter. The address encoders drive the addresses of the selected row and column onto the bus, and Req goes high. When Ack goes high, the select signals are disabled by the AND gate at the top of the arbiter tree, and Req is taken low. If necessary, the C-elements [7] between the arbiters and the rows and columns will delay inactivating the selects (and the Req signal) until the pixel has withdrawn its row and column requests, confirming that it has reset.

On the receiver side, as soon as Req goes high, the address bits are latched and Ack goes high immediately. So, while the sender chip is deactivating its internal request and select signals, the receiver decodes the addresses and selects the corresponding pixel. When the sender takes Req low, the receiver responds by taking Ack low immediately. If necessary, the receiver's C-element will delay inactivation of the decoders (and the Ack signal) until the pixel activates the wired-OR circuit, confirming that it got the spike. The following paragraphs describe the logic-circuit designs used.

The logic in the sending neuron is shown in the upper-left corner of Figure 2; it is similar to that described in [4, 3]. The neuron takes Vspk high when it spikes, and pulls the row request line Ry low. The column request line Rx is pulled low when the row select line Ay goes high. Finally, Ireset is turned on when the column select line Ax goes high, and the neuron is reset. Vadpt is also pulled low to dump charge on an integrator in order to adapt the firing rate [2]. I added a third transistor, driven by Vspk, to the reset chain to make the width of the reset pulse independent of the communication-cycle period

The sender's C-element circuit is shown in the lower-left corner of Figure 2. There are two differences between this circuit and the handshaking circuit of Lazzaro et. al. [3]. First, Lazzaro et. al. reset all the arbiter's inputs, to prevent the arbiter from granting another request while Ack is high. By using the AND gate at the top of the arbiter tree to disable the arbiter's outputs (Aarb) when Ack is high, my design leaves the arbiter's inputs undisturbed, preserving its state, and thereby reducing the time and power required to arbitrate. This approach also allows all spiking neurons in a selected row to be serviced without redoing the Y arbitration. Second, Lazzaro et. al. assume that the pixel will withdraw its request before the receiver acknowledges. This assumption may not hold if the receiver is pipelined; when the assumption fails, the row or column select lines may be cleared before the pixel has been reset.

The receiver's C-element (it is slightly different from the sender's) is shown in the lower-right corner of Figure 2. This C-element's output signal drives the Ack signal, strobes the latches, and activates the address decoders. The address-bit latch and the logic inside the receiving pixel are also shown in the figure (middle of lower row and upper-right corner, respectively). The latch is opaque when Ack is high, and is transparent when Ack is low. The pixel logic produces an active low spike whose duration depends on the delay of the wired-OR and the decoder, and on the duration of the sender's reset phase. Circuits for the blocks that are not described here—namely, the arbiter cell, the address encoder, and the address decoder—are given in [4, 3].

5. SYSTEM PERFORMANCE ANALYSIS

Timing measurements for the AE channel are shown in Figure 3. By analyzing the timing measurements to determine critical paths, I have found architectural modifications that can reduce the cycle period significantly: (i). Moving the AND gate that disables the arbiter's selection signal to the bottom of the tree would shave off a total of 144ns, but requires an AND gate for each row and each column. (ii). Removing the input to the column–wired-OR from the arbiter would allow Rpix to go back up immediately after Apix goes high, but this requires redesigning the sender's C-element. (iii). Doubling the drive of the two series devices in the pixel that pull down the column line would reduce the delay to 60ns, and allow us to hide reseting these lines in the 59ns it takes for Apix $\uparrow \Rightarrow$ Req $\uparrow \Rightarrow$ Ack $\uparrow \Rightarrow$ Xsel $\downarrow$, shaving off a total of 120ns, when arbitration occurs in only the X dimension, and 180ns when arbitration occurs in both dimensions. These changes, together, will reduce the cycle time to 156ns with arbitration in one dimension, and to 406ns with arbitration in both dimensions.

During testing, I found that the sender occasionally generates illegitimate addresses, i. e. outside the 1 to 64 range of pixel locations. In particular, row (Y) addresses higher than 64 were observed. This occurs when row 64 and one, or more, other row are selected simultaneously, and the encoder ORs their addresses together. I traced this problem to the sender's C-element (Refer to Figure 2). After Ack goes high and Aarb goes low, the arbiter sees Rpix go low, and it selects another row, but Rpix does not get below the threshold for resetting the flip-flop, so the flip-flop remains set and keeps the previous row selected. This scenario is plausible because the threshold of the inverter that drives Rarb is lower than that of the flip-flop's reset input; I calculated 2.83V and 3.27V, respectively. If any neuron in that row spikes while Rpix is between the two values, the wired-OR line (Rpix) will be pulled back low, and the flip-flop will not be reset, but the arbiter's input will go low. Rpix spends about $0.44/2.5 \times 120\text{ns} = 21\text{ns}$ in this critical window. At a total spike rate of 100KHz, we expect a collision rate of

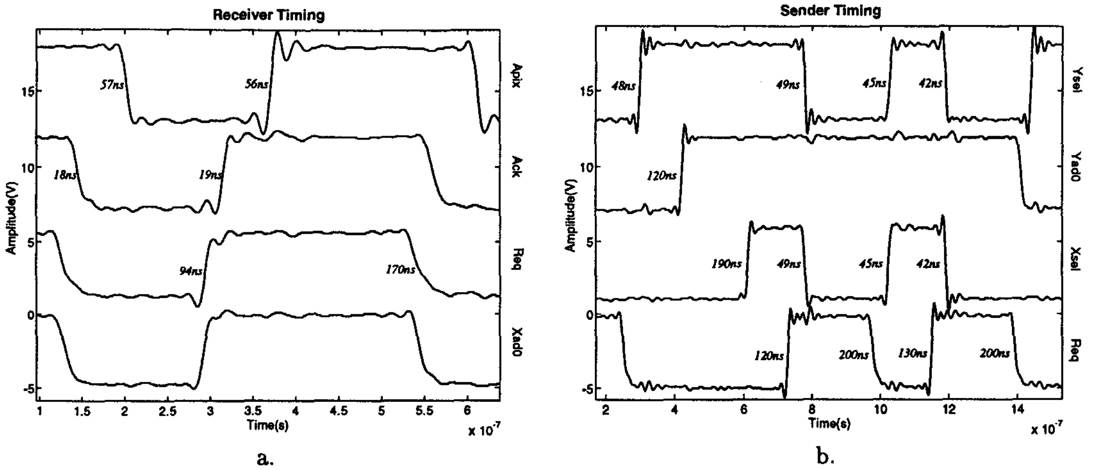

Figure 3. Communication channel timing, with measurements of time intervals between succeeding transitions. (a). Timing of the acknowledge signals from the receiving pixel (Apix) and the receiver chip (Ack), and the X-address (Xad0), relative to the request signal (Req). Pipelining the receiver reduces the cycle period by 113ns, the total time difference between Ack and Apix. (b). Timing of the select signals fed into the top of the arbiter trees (Ysel and Xsel), and the Y-address bit (Yad0), relative to the request signal (Req). The cycle period is 730ns for the first cycle, where arbitration occurs in both dimensions, and is 420ns for the second cycle, where arbitration occurs in only the X dimension.

0.05Hz, just for row sixty-four alone. I observed a rate of 0.06Hz; the higher rate observed may be due to correlations in firing times. To eliminate these collisions, we should disable neurons from firing while their row is selected, i. e., Ay is high. That way, Rpix will remain low until Apix goes low, ensuring that the flip-flop is reset.

6. CONCLUSIONS

A theoretical analysis of the tradeoffs involved in designing a time-division multiplexed communication channel for neuromorphic systems was presented. The theory showed that: (1) A random-access channel offers higher temporal resolution and shorter latencies when activity is sparse, especially if the channel capacity is allocated dynamically by adapting the sampling rates locally. (2) An arbiterless channel achieves a maximum throughput of 18 percent of the channel capacity, with a collision rate of 64 percent, whereas an arbitered channel can operate at 95 percent of the channel capacity, with a latency of 10 times the cycle period. Thus, unless the cycle time of the arbiterless channel is 5 times shorter, the arbitered channel will offer higher performance in terms of spikes transmitted per second.

Test results from a pipelined, arbitered, random-access channel were presented and analyzed. The 420ns to 730ns cycle periods this design achieved represent a threefold to fourfold improvement in the 2μs rate reported in the original work [4], for an identical array size of 64 x 64, fabricated in the same 2μm technology. Lazzaro et. al. have made similar improvements to the original design [3] and report cycle times in the 100-140ns range, but the array size and the chip size are a lot smaller.

7. ACKNOWLEDGMENTS

I thank my advisor, Carver Mead, for sharing his insights into the operation of the nervous system. I thank Misha Mahowald for making available layouts for the arbiter, the address encoders, and the address decoder; John Lazzaro, Alain Martin, and Jose Tierno for helpful discussions on address-events and asynchronous VLSI; Tobi Delbruck for help with the Mac address-event interface; and Jeff Dickson for help with PCB design.

This work is supported in part by the Office of Naval Research, by ARPA, by the Beckman Foundation, by the Center for Neuromorphic Systems Engineering as a part of the National Science Foundation Engineering Research Center Program, and by the California Trade and Commerce Agency, Office of Strategic Technology.

REFERENCES

[1] K. A. Boahen. Retinomorphic vision systems. In *MicroNeuro'96: Fifth Int. Conf. on Neural Networks and Fuzzy Systems*, Los Alamitos, CA, February 1996. IEEE Computer Soc. Press.

[2] K. A. Boahen. Retinomorphic vision systems i: Pixel design. In *IEEE Int. Symp. on Circuits and Systems*, Piscataway, NJ, July 1996. IEEE Circuits and Systems Soc., IEEE Press.

[3] J. Lazzaro, J. Wawrzynek, M. Mahowald, M. Sivilotti, and D. Gillespie. Silicon auditory processors as computer peripherals. *IEEE Trans. on Neural Networks*, 4(3):523-528, 1993.

[4] M. Mahowald. *An Analog VLSI Stereoscopic Vision System*. Kluwer Academic Pub., Boston, MA, 1994.

[5] A. Mortara, E. Vittoz, and P. Venier. A communication scheme for analog vlsi perceptive systems. *IEEE Trans. Solid-State Circ.*, 30(6):660–669, 1995.

[6] M. Schwartz. *Telecommunication Networks: Protocols, Modeling, and Analysis*. Addison-Wesley, Reading, MA, 1987.

[7] I. E. Sutherland. Micropipelines. *Communications of the ACM*, 32(6):720–738, 1989.