A Delay-Insensitive Address-Event Link

Joseph Lin
Department of Electrical and Computer Engineering
Johns Hopkins University, Baltimore, MD 21218
Email: linjh@jhu.edu

Kwabena Boahen
Department of Bioengineering
Stanford University, Stanford, CA 94305
Email: boahen@stanford.edu

***Abstract*—We present a delay-insensitive (DI) link that provides virtual point-to-point channels between ports at corresponding locations in two-dimensional arrays on separate chips. A communication, or event, on any particular channel is represented by its input port's address, which the link encodes, conveys, and decodes. Previous work cut pad-count by transmitting row and column addresses sequentially, appending additional column addresses for concurrent communications in the same row, which are read and written in parallel, thereby boosting throughput. However, a non-DI implementation was used off-chip (bundled-data), incurring delay and area penalties when interfaced with DI circuitry used on-chip. The link described here avoids these penalties by using a DI implementation both on- and off-chip (1-of-4 codes). We describe the transmitter's and receiver's implementation in detail, including refinements made to ensure efficient and robust operation with arrays as large as 320×960, and provide test results from two chips fabricated in a 0.18μm CMOS process.**

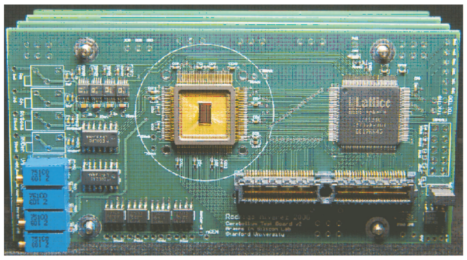
I. Address-Event Representation

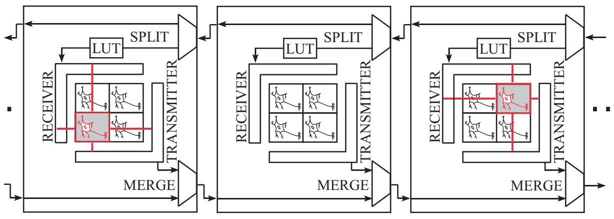
Neuromorphic engineers seek to emulate the brain's functionality by building networks of silicon neurons. Like their biological counterparts, which generate millisecond-wide, tenth-of-a-volt-high *action potentials* when sufficiently stimulated, silicon neurons generate an asynchronous digital pulse, or spike, when their input passes a threshold. Early efforts focused on sensory systems, such as the retina and the cochlea. Recent efforts have focused on cognitive systems, such as the cortex and the hippocampus. This trend has been accompanied by a shift from single- to multiple-chip systems (Fig. 1) [1], [2], [3]. The **address-event representation** (AER) has emerged as the standard for communicating spikes between chips.

Address-event links realize multiple virtual point-to-point channels with a single physical link by representing a communication on any channel with its input-port's address. They communicate spikes between two-dimensional arrays of silicon neurons by encoding the spike's row and column addresses, conveying these addresses to another chip, and decoding them to recreate the spike at its destination [4], [5]. Transmitting row and column addresses sequentially cuts pad-count, while reading and writing concurrent spikes in the same row in parallel boosts throughput [6], [7]. This **word-serial AER** link has been used in several neuromorphic systems [8], [9], [10], [11], [12].

All AER links built to date, with the exception of [1], use **bundled-data** (BD), which incurs delay and area penalties when interfaced with the **delay-insensitive** (DI) protocol used on-chip [2]. In BD, two additional lines signal when the sender outputs data (request) and when the receiver inputs it (acknowledge). A **four-phase** handshake is used: Request is asserted (data ready); acknowledge is asserted (data read); request is reset; acknowledge is reset [13]. Because the receiver latches the data when it sees the request, this line must have a longer delay than the data lines. For word-serial AER, the burst is demarcated by a second request signal, which executes two-phase handshakes at the beginning and the end with the same acknowledge signal, incurring additional delay and area penalties (Fig. 2a).

In this paper, we introduce a link that avoids these penalties by using a DI protocol both on- and off-chip—at the cost of doubling the data lines. In **1-of-2**, a four-phase handshake is performed using one line or the other to signal a bit's value,


a



b

Fig. 1. Multiple-chip stack. **a** Each board contains a cerebellum chip with 2,048 silicon neurons and 65,536 plastic synapses in $16mm^2$ of silicon, and the necessary peripherals for tuning and control. **b** The chips communicate with their neighbors through AER links—an open-ring network—where each chip is tuned to behave as a specific cell type found in the olivo-cerebellar complex.

TABLE I
1-of-4 Code

binary	00	01	10	11
1-of-4	0001	0010	0100	1000

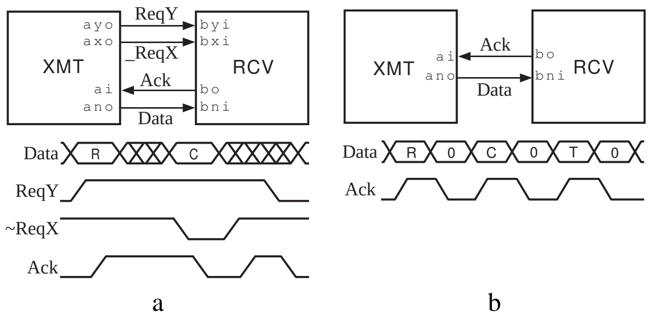


Fig. 2. Bundled Data (BD) and Delay-Insensitive (DI) Address-Event Links. **a** XMT outputs a row address (R) and raises ReqY; RCV latches it and raises Ack. Then XMT outputs a column address (C) and lowers ˜ReqX; RCV latches it and lowers Ack. Then XMT raises ˜ReqX and RCV raises Ack. This cycle repeats for additional column addresses. Finally, XMT lowers ReqY and RCV lowers Ack. **b** XMT outputs a one-hot coded row address (R); RCV latches it and raises Ack. Then XMT outputs a neutral code (0) and RCV lowers Ack. This cycle repeats for additional column addresses (C) and a tailword (T).

0 or 1.[1] A valid signal (one line high) cannot be confused with an invalid (both low) or incomplete signal (raising both lines is disallowed), making a separate request line—and its added delays—superfluous [14]. We dispensed with the second request signal as well by introducing a reserved address—the **tailword**—that demarcates the end of a burst (Fig. 2b).

To mitigate the increase in I/O pads, we cut power pads in half by adopting **1-of-4** coding: One of four lines is raised to signal one of four possible two-bit values (Table I). Half as many as 1-of-2 and BD (worst-case)—halving power pad requirements. For instance, our chip's 10-bit 1-of-4 datapath (9 for address plus 1 for tail) required 29 I/O pads: five sets of 1-of-4 (20 pads) with 5 $V_{\rm dd}$/Gnd pads interleaved between them, plus reset, acknowledge, and two $V_{\rm dd}$/Gnd pads for the core. If we had used bundled-data, we would have had 25 pads: 9 address with 10 $V_{\rm dd}$/Gnd pads interleaved between them, plus Y-request, X-request, acknowledge, reset, and core $V_{\rm dd}$ and Gnd, a savings of only 4 pads.

Section II reviews one-hot coding. Section III describes the architecture of the transmitter and receiver. Section IV describes the logic synthesis process and presents the resulting circuits. Section V presents test results from two chips. Section VI presents concluding remarks.

II. One-Hot Codes

One-hot codes use 2^b lines to encode b bits: The ith line is raised to transmit the ith b-bit word. As the number of bits increases, line-count increases exponentially while switching activity decreases linearly, with the optimum occuring at $b = 2$. In addition to these 2^b **valid** codes, there is a single **neutral** code: All lines low. A m-input OR checks for validity or neutrality in a single 1-of-m group; a datapath with n groups requires n m-input ORs and $n-1$ C-elements connected in a binary tree.[2] For neutrality alone, an OR-tree suffices.

[1]Popularly known as dual-rail.

[2]A C-element's output goes high if both inputs are high, low if both are low, and remains the same if they differ.

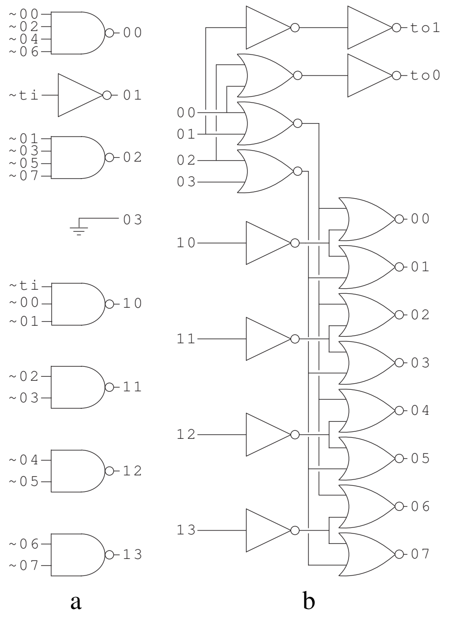


Fig. 3. Tailword Creation and Tailbit Extraction. **a** Combines tailbit (`ti`) with a 1-of-8 group to create two 1-of-4 groups. **b** Extracts tailbit (`to0,to1`) from a 1-of-4 group and combines the other bit with another 1-of-4 group to create a 1-of-8 group.

We label the ith one-hot line `xi`, where `x` denotes the group number; for a control line, it represents the signal's name. For example, the first group's 1-of-4 signals are labeled `00`, `01`, `02`, and `03`. And control signal `c`'s 1-of-2 signals are labeled `c0` ($c = 0$) and `c1` ($c = 1$).

We chose to use 1-of-4 codes for the link, with an extra address-bit (LSB) that is 1 in the tailword and 0 elsewhere (i.e., within a burst). This **tailbit** is combined with the address' three least-significant bits—for which the encoder uses a 1-of-8 code—to obtain two 1-of-4 groups (Fig. 3a). When the tailbit is extracted from the least-significant 1-of-4 group at the decoder, the remaining bit is combined with the second-least significant 1-of-4 group to recreate the 1-of-8 group (Fig. 3b).

The one-hot encoder and decoder are built from wired-NOR and NAND gates, respectively. For the encoder, we cut the wired-NOR's static power dissipation by activating its pull-up only when the encoder's output is acknowledged (Fig. 4) [6]. The pull-up's weaker sizing guarantees the output is not cleared until the input has cleared. For the decoder, we assumed that when a NAND gate clears its output, all its inputs have cleared. This timing assumption requires the address-lines' capacitances to be matched (Fig. 5). Avoiding 1-of-2 coding, which connects twice as many gates or drains to the address-line as 1-of-4 coding, which itself connects twice as many as 1-of-8, helps to achieve this—and speeds up performance while lowering power dissipation.

III. DI Architecture

We came up with architectures for the transmitter and receiver through process decomposition, the first step of Martin's three-step synthesis procedure for asynchronous circuits [15].

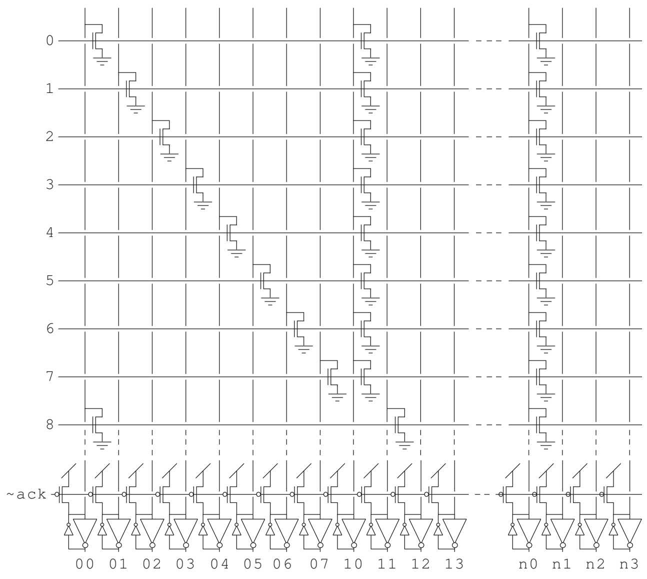


Fig. 4. One-Hot Encoder. Uses 1-of-8 for three least-significant bits and 1-of-4 for remaining bits. If a bit is leftover, it is combined with the previous two to form a 1-of-8 group. The small inverters hold state (weak-feedback).

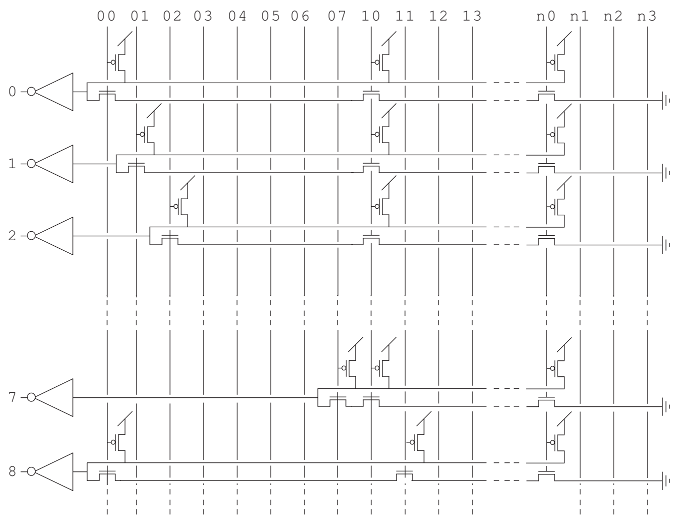


Fig. 5. One-Hot Decoder. Uses the same grouping-scheme as the encoder.

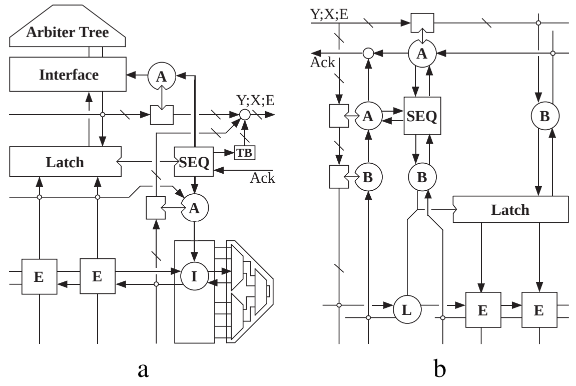


Fig. 6. Transmitter and Receiver Architecture. **a** *Transmitter:* The selected row's events are readout in parallel and transmitted in a single packet—row address (Y), column address(es) (X), and tailword (E, generated by TB). **b** *Receiver:* The packet's events are written to the addressed row in parallel. Discs symbolize combinational logic. Note that the latches send back valid/neutral signals to their controllers (dimples);

The resulting DI architectures are similar to the BD designs [6], [7]; we will note the key differences.

In the DI transmitter (Fig. 6a), event generators (E) make requests to the row arbiter through an interface (I). The row arbiter selects one row and directs an encoder to output that row's address, which is buffered (A). Selection by the arbiter also enables the row's event generators to raise their column lines, which are latched before being relayed to the column arbiter. As was done for the row, this arbiter directs an encoder to output these columns' addresses to a second buffer (A) one by one. After prompting the first buffer to output the row address, the sequencer (S) repeatedly prompts the second one to output the column addresses—until the latch signals that it is empty. At which point the sequencer causes the tailword (TB) to be output and prompts the latch to load the next row, which was read out while the burst was being sent. The row, column, and tail words are output through OR-gates, in contrast to the BD design, which used a two-way multiplexor.

In the DI receiver (Fig. 6b), the sequencer directs buffers (A) to load incoming row and column addresses. They are decoded, stored (L and Latch), and sent to the event-recipients when the tailword shows up. Additional buffering (B) allows a burst to be decoded while the previous one is being written. Note that the row-address is decoded immediately, unlike the BD design, which waited till the burst ended.

As our DI architectures are similar to the BD designs, we skip describing process decomposition and proceed to handshaking expansion (HSE), the second synthesis step.

IV. Handshaking Expansion

In HSE, each communication is expanded into a pair of four-phase request-acknowledge sequences:

```
# Four-phase #
*[ao+;[ai];ao-;[~ai]]
*[[pi];po+;[~pi];po-]
```

The active port (A) asserts the request (`ao+`) and waits for the acknowledge (`[ai]`), then deasserts the request (`ao-`) and waits for the acknowledge to clear (`[~ai]`)—a **lazy-active** port checks immediately before raising the request to start the next handshake. The passive port (P) waits for the request (`[pi]`) before asserting the acknowledge (`po+`), then waits for the request to clear (`[~pi]`) before deasserting the acknowledge (`po-`). Note that the first two-phase communication synchronizes A and P; the second is superfluous—it just returns the signals to their initial state. For more on HSE notation, see Table II (at the end of Section VI). We present the transmitter's HSE, followed by the receiver.

A. Transmitter

In the transmitter, EVT initiates the cycle (Fig. 7). With P passive, R active, and C active, EVT's HSE is:

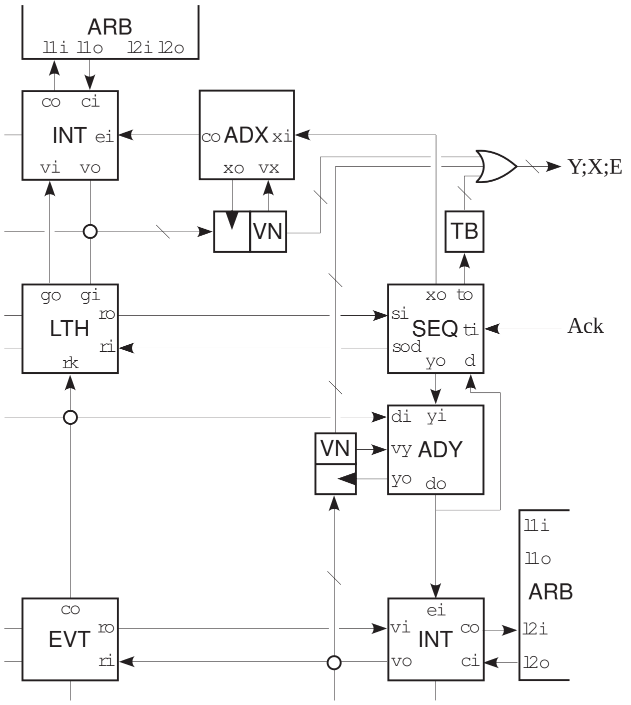


Fig. 7. Transmitter Schematic. Elaborated to show signal names and valid/neutral detectors (VN).

```
# EVT #
  *[[pi];ro+;[ri];co+,po+;
    [~pi];ro-;[~ri];co-,po-]
```

The first and second two-phase communications with INT signal that EVT's row has been selected (ro+;[ri]) and its column signal (co+) has been read (ro-;[~ri]), respectively. EVT communicates with the event-generator (po+;[~pi]) after the first and clears its column signal after the second.

INT receives EVT's row request through a row-wide wired-OR, identical to the encoder's (see Fig. 4).[3] With V passive, C active, and E active, INT's HSE is:

```
# INT #
  *[[vi];co+;[ci&~ei];eo+,vo+;
    [~vi&ei];co-;[~ci];eo-,vo-]
```

The first and second two-phase communications with ARB signal when INT's row is selected (co+;[ci]) and unselected (co-;[~ci]). Thus, INT communicates with EVT (vo+;[~vi]) and the row-address encoder (eo+;[ei]) after the first but postpones clearing these signals till after the second. This pipelining is safe because the next INT selected by the arbiter waits until ei clears (lazy-active). INT is also used to interface LTH with the column arbiter and encoder.

$N-1$ copies of ARB, connected in a binary tree, arbitrate INT's request and those from the other $N-1$ rows (or columns). With L_1 and L_2 passive and R active, each ARB handles its daughters' requests with two identical concurrent HSE sequences:

```
# ARB #
  *[[l1i&~ri];a1o+;r1o+;[ri&a1i];l1o+;
    [~l1i];a1o-;r1o-;[~a1i];l1o-]
||
  *[[l2i&~ri];a2o+;r2o+;[ri&a2i];l2o+;
    [~l2i];a2o-;r2o-;[~a2i];l2o-]
```

Requests from one daughter ([l1i];r1o+) or the other ([l2i];r2o) propagate up the tree, ORed to produce a single outgoing request (ro), which feeds back at the top. Acknowledges propagate down the tree, steered to one side ([ri&a1i];l1o+) or the other ([ri&a2i];l2o+) by a local two-way arbiter (see Fig. 8). Clearing follows the same sequence, except that R is lazy-active, thereby preventing new requests from being accepted until ri clears. For more detail on this non-greedy arbiter, see [6].

LTH receives EVT's column signal through a column-tall wired-OR, identical to the encoder's (see Fig. 4). With R passive and G active, LTH's HSE is:

```
# LTH #
  *[[ri&rk];go+;ro+;[~ri&gi];go-;[~gi];ro-]
```

The first and second two-phase communications with SEQ signal when LTH is full ([ri];ro+) and when it is empty ([~ri];ro-), respectively.[4] The latter completes after LTH finishes communicating with INT. LTH does not check that the column signal cleared ([~rk])—SEQ ascertains that before initiating the next cycle (see below)—this way, the empty signal and the column signal clear concurrently.

ADX captures the row address when directed by SEQ. With D and Y passive, ADY's HSE is:

```
# ADY #
  *[[yi];yo+;[vy];do+;[~yi];yo-;[~vy&~di];do-]
```

It prompts its buffer to latch and output the row address ([yi];yo+), and acknowledges receipt as soon as the output becomes valid ([vy];do+). It does not check the column address' ([dni]) and requests' ([di]) validity because that is implied (by [vy] and [yi], respectively). The signals clear in the same sequence.[5]

Similarly, ADX captures the column address when directed by SEQ. With X and C passive, ADX's HSE is identical to ADY's:

```
# ADX #
  *[[xi];xo+;[vx];co+;[~xi];xo-;[~vx];co-]
```

Except that [~di]'s counterpart is missing because there are no extra lines to check.

That brings us to SEQ, which orchestrates the whole show. With S, Y, X and T active, its two concurrent HSE sequences are:

[3]To ensure that ro remains low until ri clears, other EVTs in that row must be prevented from placing a request when ri is true.

[4]Because the acknowledges feed into an OR-tree, the first of multiple acknowledges to arrive triggers the full signal, thus their delays must be matched.

[5]Note that the column address lines' neutrality is not checked—[~vy] does not imply [~dni]—therefore their delay must be shorter than the column request lines'.

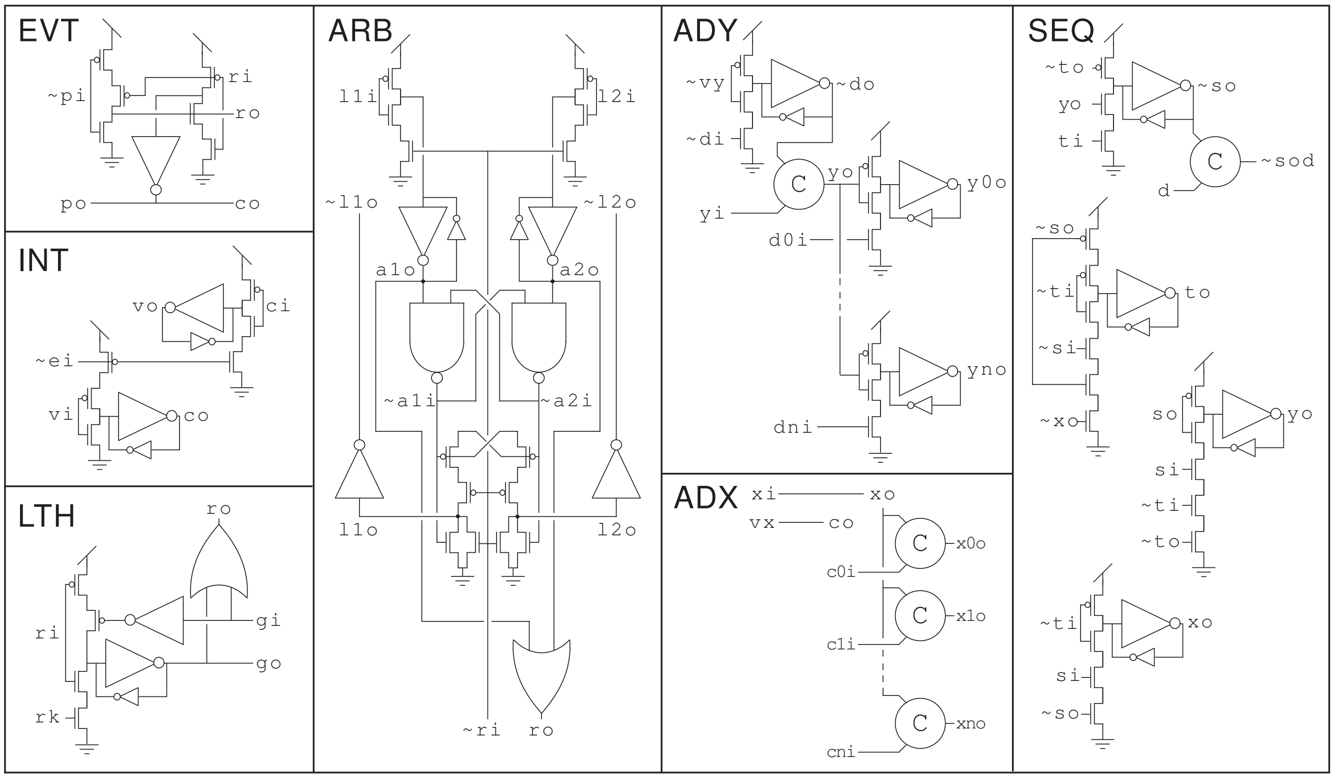


Fig. 8. Synthesized Transmitter Circuits. The tilde denotes a signal's complement. ADX's `xi` is wired to `xo`, and its `vj` is wired to `co`.

```
# SEQ #
  *[[~ti&si&so];yo+;[ti];so-;yo-;
    [~ti&~si&~so];to+;so+;[ti];to-]
||
  *[[~ti&si&~so];xo+;[ti];xo-]
```

They output row/tail (Y or T) and column (X) words, respectively, through triangular communications with the receiver that loop through ADY, ADX, or TB (see Fig. 7). Control switches to the second sequence halfway through the first sequence, which starts when LTH becomes full (`[si]`) and resumes when it becomes empty (`[~si]`). To ensure the column signals clear before LTH becomes transparent, we guarded SEQ's `so+` with ADY's `do-` (see `~sod` in Fig. 8).

Once we have HSE sequences, we proceed to the final step, compiling them into production-rule sets (PRS), which are straightforward to implement with CMOS transistors [15]. Due to space constraints, only the synthesized CMOS circuits are shown (Fig. 8). In the next section, we present the receiver's handshaking sequences.

B. Receiver

In the receiver, SEQ initiates the cycle (Fig. 9). With Y, X, and S active, its HSE is:

```
# SEQ #
  *[yo+;[yi];xo+;[xi];so+;
    yo-;xo-;[~xi&si];so-;[~si&~yi]]
```

It directs ADY to load the row address (`yo+;[yi]`), ADX to load the column addresses (`xo+;[xi]`), and BUF to write the burst into the array (`so+;[si]`). Apart from ensuring that `si` clears before the next burst is read in (see below), the order in which the signals clear is unimportant.

ADY inputs the row address when directed by SEQ. With Y passive, G passive, and H active, ADY's HSE is:

```
# ADY #
  *[[yi];ho+;[vh];go+;[ng];yo+;
    go-;[hi];ho-;[~vh&~hi&~yi];yo-]
```

It prompts its buffer to latch and output the row address (`[yi];ho+`), and acknowledges receipt as soon as the output becomes valid (`[vh];go+`)—without waiting for HBUF to read the output (`[hi]`). Only after the input clears (`[ng]`) does it acknowledge SEQ (`yo+`), which can now safely ask ADX to load the column address. Clearing `go` early allows this to happen as quickly as possible.

By providing additional buffering, HBUF allows another row address to be input while the decoder is still selecting the previous row. With H passive and P active, HBUF's HSE is:

```
# HBUF #
  *[[~pi];po+;[vp];ho+;[pi];po-;[~vp];ho-]
```

It acknowledges receipt as soon as its buffer captures the row address (`[vp];ho+`), without waiting for the decoder's acknowledge (`[pi]`). Notice that HBUF would be identical to the transmitter's ADX if P was passive.

LTH captures the row decoder's output, allowing it to acknowledge HBUF before the row is actually selected. With B and V passive, LTH's HSE is:

```
# LTH #
  *[[bi];bo+;[vi];vo+;[~bi];bo-;[~vi];vo-]
```

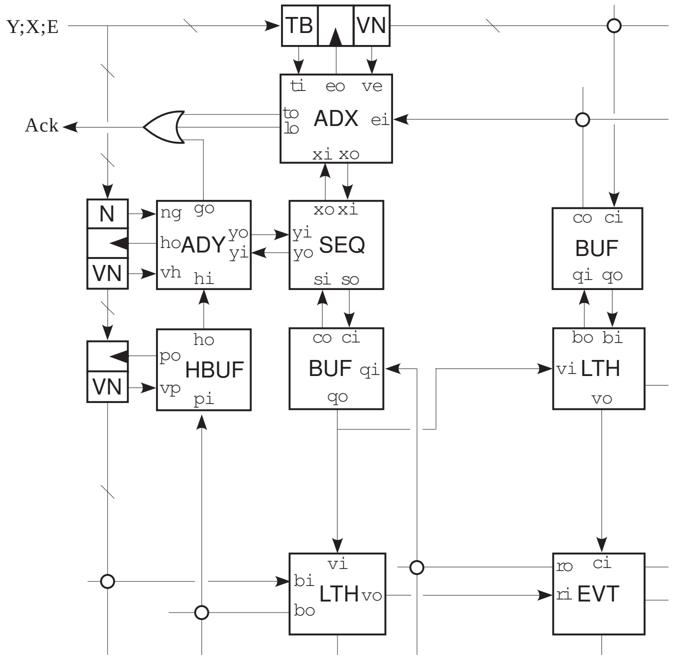


Fig. 9. Receiver Schematic. Elaborated to show signal names, valid/neutral detectors (VN), and tailbit extractor (TB).

Having captured the decoder's output (`[bi];bo+`), LTH selects the row when BUF prompts it (`[vi];vo+`). The signals clear in the same sequence. LTH is also used to select a column.

ADX inputs the column address when directed by SEQ. With X passive, E active, L passive , and T passive, its two concurrent sequences are:

```
# ADX #
   *[~ei&xi&ti0];eo+;[ve];lo+;
     [ei&~ti0];eo-;[~ve];lo-]
||
   *[[ xi&ti1];xo+;to+;[~xi&~ti1];xo-;to-]
```

The first sequence relays column addresses to the decoder while the second sequence communicates with SEQ; control switches to the latter when the tailbit becomes true (`[ti1]`). Notice that, apart from `[ti0]`, the first sequence is identical to HBUF's.

By providing a buffer between the column decoder and LTH, BUF allows another burst to be decoded while the one in LTH is being written. With C passive and Q active, BUF's HSE is identical to LTH's, except that Q is lazy-active rather than passive:

```
# BUF #
   *[[ci];co+;[~qi];qo+;[~ci];co-;[qi];qo-]
```

To prevent the stored event from proceeding to LTH while the previous row is being written, we stall it by guarding LTH's `bo+` with `[~vi]`, which works because `vi` is true when the new burst comes in.[6] BUF is also used to buffer SEQ from LTH.

[6]SEQ waits for `~si`, which corresponds to the other BUF's `co-`, which happens after BUF's `qo+`, which corresponds to LTH's `vi`.

That brings us to EVT, which receives the event from the two LTHs through triangular communications initiated by SEQ's BUF (see Fig. 9). With R passive, C passive, and P active, EVT's HSE is:

```
# ROW #
*[[ri&ci];po+;[pi];ro+;[~ri&~ci];po-;[~pi];ro-]
```

When the LTHs select its row and column (`ri` and `ci`), EVT arouses the event-recipient and relays its acknowledge to BUF (`po+;[pi];ro+`). The row-wide wired-OR that feeds `ro` to an array-tall OR-tree on the periphery is identical to the encoder's except that the select signal inactivates the pull-up [7]. Clearing follows the same sequence.[7]

As with the transmitter, we skip compiling HSE into PRS and show the synthesized CMOS circuits (Fig. 10).

V. Test Results

We present test-data for the link's transmitter and receiver, obtained from two chips, both fabricated in 0.18μm CMOS. The first chip, with a 256×8 array of silicon neurons in $4.9 \times 2.9\text{mm}^2$, provided the transmitter data (Fig. 11a). The second chip, with a 320×960 array of pixels in $6.2 \times 4.8\text{mm}^2$ [1], provided the receiver data (Fig. 11b). Both chips were designed to be daisy-chained, and were tested talking to another copy of the same chip. The transmitter chip merged the incoming address-events with those from its own transmitter to form the outgoing stream.The receiver chip copied the incoming address-events to its own receiver as well as to the outgoing stream. FIFOs buffered the outgoing stream, enabling us to use full FIFOs—filled by holding up the link—to pump address-events at full-speed from chip to chip.

We measured an interword-interval of 23ns for the transmitter (Fig. 12) and 16ns for the receiver (Fig. 13). The difference in burst-rate is attributed to the longer wires connecting the transmitter chips, which were on separate boards (see Fig. 1a), whereas the receiver chips were side-by-side on the same board. Indeed, it appears that the transmitter's cycle-time was limited by interchip-delays because the interval was the same (23ns) when we dumped out events from a full FIFO. This result is not surprising, since the array size was rather small. On the contrary, it appears that, in the worst case, the receiver writes events into the array at a slower clip than the 64ns it takes to deliver each event's four words—chip address, row address, column address, and tailword. Further testing is required to ascertain this.

Our receiver testing revealed that not all of a burst's events made it into the target row (Fig. 14). We determined that the leftward and downward bias was due to delays in the wires, modeled with RC-segments. The width of the pulse BUF launches is determined by the propagation delay down to the selected row, plus the time it takes for the first pixel in that row to acknowledge, plus the propagation delay back up through the OR-tree. While the first gets longer with distance, the second and third remain constant. Thus, the lower the

[7]Because clearing is triggered by the first EVT to acknowledge, when multiple column lines are active their delays must be matched.

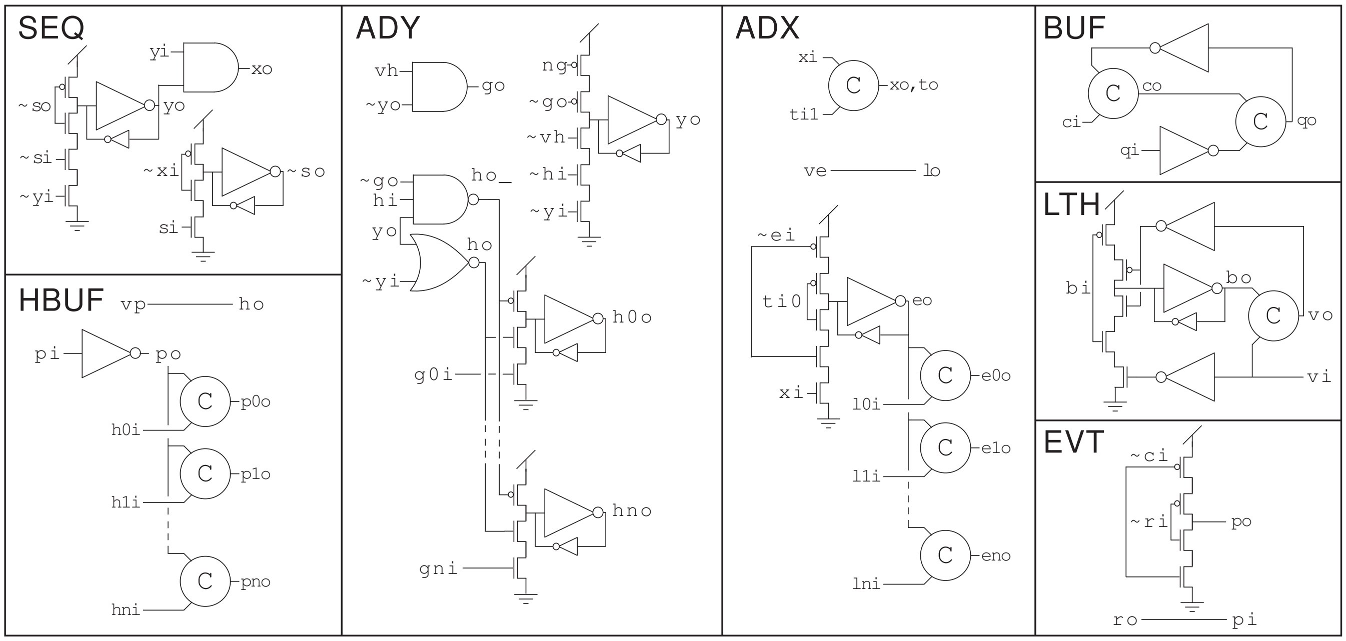


Fig. 10. Synthesized Receiver Circuits. HBUF's `vp` is wired to `ho`; ADX's `ve` is wired to `lo`.

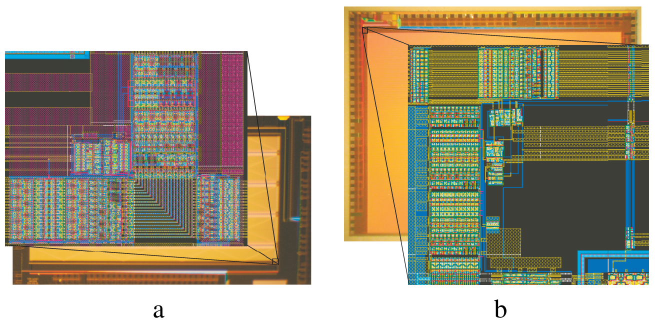


Fig. 11. Transmitter and receiver die micrograph and layout (insert). **a** The transmitter circuitry that services this 14.1mm^2 chip's 256 $\times$ 8 array occupies 0.57mm^2 (0.18-μm CMOS). **b** The receiver circuitry that services this 29.6mm^2 chip's 320 $\times$ 960 array occupies 1.26mm^2 (0.18-μm CMOS).

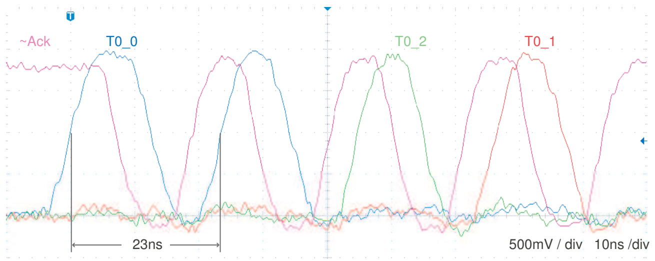


Fig. 12. Transmitter Performance. Four words in an address-event—chip, row, column, and tail word—measured at 23ns per word. Three of the first 1-of-4 group's signals are shown, together with the active-low acknowledge.

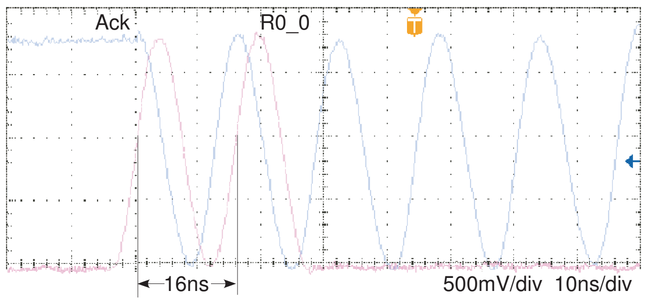


Fig. 13. Receiver Performance. Active-low acknowledge shown with first signal of the first 1-of-4 group–measured at 16ns per word.

selected row, the longer the pulse. These longer pulses travel further along the selected row. Thus, the lower the row, the more pixels are selected.[8] This behavior can be rectified by moving the OR-tree to the far side of the array to guarantee that the select pulse makes it all the way across.

We investigated the role of wiring delays further by simulating a 256×256 transceiver array on a $13.9 \times 11.8\text{mm}^2$ chip, with global lines replaced with RC-segments extracted from the layout. We found that the wired-ORs used to aggregate row and column requests—and in the encoder—tended to become active again when the acknowledge was cleared. Because, the line's far end remains below V_{inv} and switches the request back high when charge redistributes along the line (Fig. 15). We rectified this behavior by placing the pull-up at the far end of the line, which guarantees that the entire line is above V_{inv} when the output goes low.

[8]The pulse BUF launches rightward will arrive at EVT simultaneously—and will be subject to the same amount of filtering—if vertical LTH-to-LTH RC-segments are identical to vertical EVT-to-EVT RC-segments and horizontal LTH-to-LTH RC-segments are identical to horizontal EVT-to-EVT RC-segments.

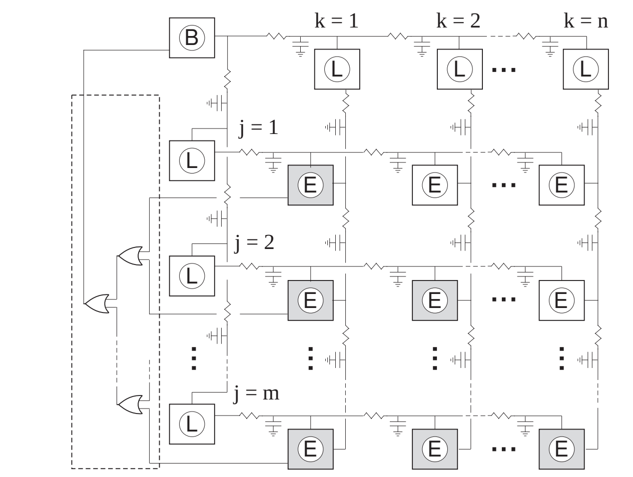


Fig. 14. RC delay model. Shaded pixels receive an event when a row-wide burst is input. Differential wiring delays account for the leftward and downward bias.

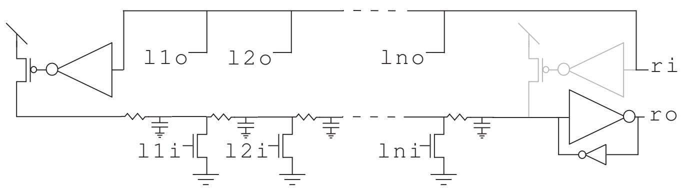


Fig. 15. Wired-OR for Requests. Placing the active pull-up at the far end of the line guarantees correct behavior.

VI. Discussion

We presented DI transmitter and receiver designs based on the BD burst-mode word-serial architectures developed in [6], [7]. Making interchip communication DI streamlined interfacing with on-chip DI circuitry, eliminating delay and area penalties incurred with BD. However, DI requires twice as many I/O pads as BD. We mitigated this by using 1-of-4 coding, which cuts the number of power pads in half.

For area-efficiency, our design assumes matched delays when signals are broadcast to an entire row or column, such as on the receiver's row-select line. In large chips, differential wiring delays violated this assumption, causing events sent to the far side of a row to be dropped when events were sent to the near side in the same burst. These distributed-RC effects also negatively impacted the wired-OR design with an active pull-up that the transmitter uses to aggregate row- and column requests. We proposed fixes in both cases that simply involve relocating the receiver's OR-tree and the transmitter's active pull-ups to the far end of the lines in question.

Acknowledgment

The authors would like to thank Rodrigo Alvarez for providing data on the transmitter performance and John Arthur for simulating the 256×256 transceiver. This work was supported in part by NSF's CAREER (Grant ECS00-93851) and EMT (Grant CCF-0630505) programs.

TABLE II
HSE Notation

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 till `v` is high
Sequential	`[u];v+`	≡ `u -> v+` in PRS
Concurrent	`v+,w+`	≡ `v+,w+` in PRS
Repetition	`*[v+;v-]`	Repeats forever

References

[1] J. Lin, P. Merolla, J. Arthur, and K. Boahen, "Programmable connections in neuromorphic grids," *Circuits and Systems, 2006. MWSCAS '06. 49th IEEE International Midwest Symposium on*, vol. 1, pp. 80–84, Aug. 2006.

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

[3] L. Plana, S. Furber, S. Temple, M. Khan, Y. Shi, J. Wu, and S. Yang, "A gals infrastructure for a massively parallel multiprocessor," *Design and Test of Computers, IEEE*, vol. 24, no. 5, pp. 454–463, Sept.-Oct. 2007.

[4] M. Sivilotti, "Wiring considerations in analog VLSI systems, with application to field-programmable networks," Ph.D. dissertation, California Institute of Technology, Pasadena CA, 1991.

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

[6] K. Boahen, "A burst-mode word-serial address-event link-i: transmitter design," *IEEE Trans. Circuits Syst. I*, vol. 51, no. 7, pp. 1269–1280, 2004.

[7] ——, "A burst-mode word-serial address-event link-ii: receiver design," *IEEE Trans. Circuits Syst. I*, vol. 51, no. 7, pp. 1281–1291, 2004.

[8] K. Zaghloul and K. A. Boahen, "Optic nerve signals in a neuromorphic chip: Outer and inner retina models," *IEEE Trans. on Biomed. Eng.*, vol. 51, no. 4, pp. 657–666, Submitted.

[9] T. Y. W. Choi, B. E. Shi, and K. Boahen, "An on-off orientation selective address event representation image transceiver chip," *IEEE Trans. Circuits Syst. I*, vol. 51, no. 2, pp. 342–353, 2004.

[10] P. Merolla and K. Boahen, "A recurrent model of orientation maps with simple and complex cells," in *Advances in Neural Information Processing*, S. Thrun and L. Saul, Eds., vol. 15. San Mateo CA: Morgan Kaufman, 2004, pp. 995–1002.

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

[12] J. Arthur and K. Boahen, "Synchrony in silicon: The gamma rhythm," *IEEE Trans. Neural Networks*, vol. 18, no. 6, pp. 1815–1825, 2007.

[13] C. A. Mead, *Introduction to VLSI Systems*. Reading MA: Addison Wesley, 1980.

[14] A. J. Martin, "Programming in vlsi: From communicating processes to delay-insensitive circuits," California Institute of Technology, Technical Report CaltechCSTR:1989.cs-tr-89-01, 1989.

[15] ——, "Synthesis of asynchronous vlsi circuits," in *Formal Methods For VLSI Design*, J. Staunstrup, Ed. North-Holland, 1990, pp. 237–283.