A High-Speed Clockless Serial Link Transceiver

John Teifel and Rajit Manohar
Computer Systems Laboratory
Electrical and Computer Engineering
Cornell University
Ithaca, NY 14853, U.S.A.

Abstract

We present a high-speed, clockless, serial link transceiver for inter-chip communication in asynchronous VLSI systems. Serial link transceivers achieve high off-chip data rates by using multiplexing transmitters and demultiplexing receivers that interface parallel on-chip data paths with high-speed, serial off-chip buses. While synchronous transceivers commonly use multi-phase clocks to control the data multiplexing and demultiplexing, our clockless transceiver uses a token-ring architecture that eliminates complex clock generation and synchronization circuitry. Furthermore, our clockless receiver dynamically self-adjusts its sampling rate to match the bit rate of the transmitter. Our SPICE simulations report that in a 0.18-µm CMOS technology this transceiver design operates at up to 3-Gb/s and dissipates 77 mW of power with a 1.8-V supply voltage.

1. Introduction

We describe the design of a high-speed, clockless, serial link transceiver. As the demand for off-chip bandwidth grows with on-chip operating frequency, high bit-rate I/O pins become increasingly necessary for inter-chip signaling interfaces in VLSI systems. While it is always possible to increase off-chip bandwidth by making buses wider with more I/O pins, it is often impractical due to cost and limits in packaging technology. This suggests a chip design should efficiently utilize its existing I/O pins by driving them at high bit rates. An attractive high bit-rate I/O communication scheme, utilized in high-speed synchronous links, multiplexes and demultiplexes on-chip data onto a high-speed, off-chip serial bus. In this paper we propose an analogous scheme for asynchronous links.

Multi-gigabit/second (Gb/s) inter-chip communication links can be built in standard CMOS processes with high fan-in multiplexing drivers and high fan-out demultiplexing
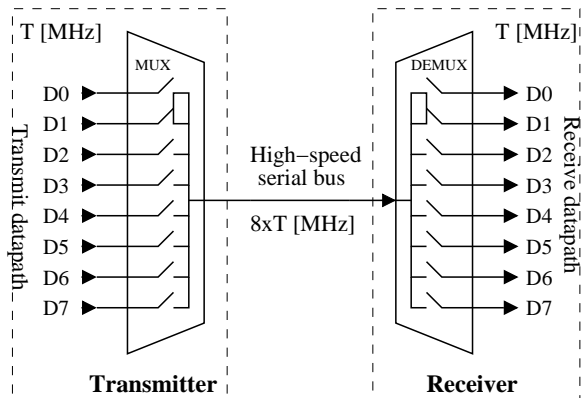

Figure 1. Serial link using multiplexed transceivers.

receivers. Figure 1 shows a typical Gb/s link that multiplexes and demultiplexes data onto a high-speed, off-chip serial bus. The on-chip data paths in the transmitter and receiver chip operate at a throughput[1] of T and employ an eight-way multiplexer and demultiplexer. During a single on-chip cycle, all of the switches in the multiplexer are sequentially enabled and eight data bits (D0..7) are transmitted on the serial bus. Similarly at the receiver, the switches in the demultiplexer sequentially sample the serial bus and decode all of the eight transmitted data bits (D0..7) in a single on-chip cycle. Since the data multiplexing occurs directly at the I/O pins, which behave as low-impedance transmission lines, we can drive the off-chip serial bus at high data-rates. In an eight-way multiplexing and demultiplexing transceiver, the off-chip serial bus is required to support a maximum bit rate of $8T$.

Existing multiplexed transceiver architectures [5, 12] use clocks to control the data multiplexing and demultiplexing. A block diagram of a conventional clocked transceiver is shown in Figure 2. The clocked transmitter uses eight phase-shifted clocks ($\phi0..7$) generated from an on-chip

---

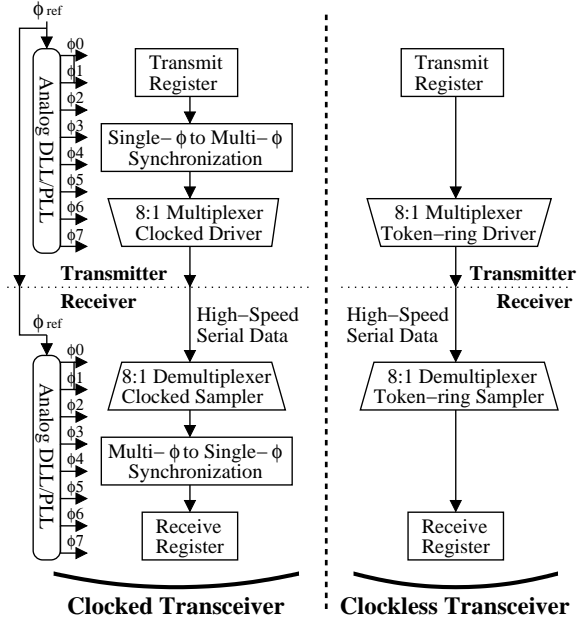[1] Throughput is the inverse of cycle time.

1

Figure 2. General transceiver architectures.

clock (ϕ_{ref}) to control the eight switches of the multiplexer, with each switch enabled by a unique clock. While the multiplexed switches operate in parallel, they transmit bits sequentially onto the serial bus because their clocks are out-of-phase. The receiver uses a reference clock (ϕ_{ref}), transmitted with the serial data, to generate its own phase-shifted clocks ($\phi 0..7$) that control the switches of the demultiplexer. The demultiplexed switches operate in parallel, but are sequentially enabled (out-of-phase) to correctly decode the serial data. Both the transmitter and receiver use PLLs (phase-locked loops) or DLLs (delay-locked loops) to generate the phase-shifted clocks that are critical for precise transmission and decoding of the serial bit stream. Additionally, clocked transceivers have a synchronization latency in converting on-chip, single-phase data to multi-phase data used at the multiplexer and demultiplexer.

Instead of generating multi-phase clocks, we propose a clockless architecture that uses asynchronous token-rings to control the data multiplexing and demultiplexing in the transceiver. Figure 2 shows a side-by-side comparison of our clockless transceiver to a traditional clocked transceiver. A clockless transceiver eliminates all of the clock generation and synchronization circuitry that complicates a clocked transceiver design. Since no clock is transmitted with the serial bit stream, a clockless serial link receiver must extract bit boundaries from the serial data stream. This requires the serial data to be encoded and transmitted on more than one data wire. For ease of encoding and decoding our clockless transceiver uses three wires to transmit serial data, whereas a clocked transceiver would use only two wires (data and clock). However, the particular data-encoding that we use in our serial link allows a clockless receiver to dynamically self-adjust its sampling rate to match the bit rate sent at the transmitter.

All the circuits in our clockless transceiver, except for the off-chip serial bus, are designed using quasi-delay-insensitive (QDI) asynchronous circuits.[2] The off-chip bus is not delay-insensitive (DI) because we do not generate a low-level acknowledgment for the transmitted serial data. Purely DI buses require an acknowledgment handshake for every data value transmitted and high bit rates are not achievable as the data-rate is limited by the round-trip, inter-chip propagation delay of the DI handshake [8]. Since our clockless transceiver is targeted for systems using QDI cores, we want to minimize the timing assumptions needed to attain high bit rates on the off-chip serial bus. In our design we need only guarantee one timing constraint: *the transmitter shall not transmit at a bit rate faster than the maximum bit rate supported by the receiver*. This conservative, one-sided timing assumption allows the transmitter to transmit bits at any data-rate from zero (idle link) up to the maximum bit rate supported by the receiver.

This rest of this paper provides design details for high-speed clockless transceivers and is organized as follows. Section 2 discusses the three-wire asynchronous signaling protocol used for data-encoding on the high-speed serial bus. Section 3 provides architectural details of the token-ring multiplexers and demultiplexers and in Section 4 we show circuit details of the transmitter and receiver. In Section 5 we show how the transceiver design can be modified to handle bit errors and Section 6 discusses related work. Concluding remarks are in Section 7.

2. Three-wire Asynchronous Signaling

Instead of using a clock to determine bit boundaries, asynchronous signaling protocols encode bit boundaries as state transitions on interconnect wires. One way to do this is to use three-wire signaling protocols first proposed by Røine [21] for use in high-speed asynchronous links. In three-wire protocols, next state information regarding the protocol is transmitted across the three-wire interconnect. Figure 3 shows an example of a three-wire state diagram that is used for both data encoding at the transmitter and data decoding at the receiver. Since there are exactly three states in this protocol, next state information is efficiently transferred across the interconnect by sending a single pulse along one of the three interconnect wires. In Figure 3 we denote this pulse as P_i, for $i \in \{0, 1, 2\}$, where i is one wire of the three-wire interconnect upon which the pulse is transmitted.

---

[2]Our clockless transceiver architecture is not restricted to QDI circuits and is generally applicable to all asynchronous handshake circuits.
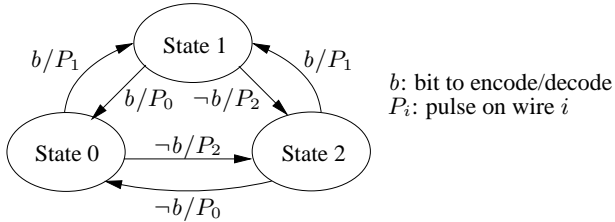
Figure 3. Three-wire state transition diagram.

b: bit to encode/decode
P_i: pulse on wire i

Inputs		Outputs	
Current State	Data Value	Next State	Wire Pulse
0	0	2	2
	1	1	1
1	0	2	2
	1	0	0
2	0	0	0
	1	1	1

Table 1. Three-wire state table for transmitter.

Inputs		Outputs	
Current State	Wire Pulse	Next State	Data Value
0	2	2	0
	1	1	1
1	2	2	0
	0	0	1
2	0	0	0
	1	1	1

Table 2. Three-wire state table for receiver.

The three-wire state machine can be initialized to any of the three states, as long as both the transmitter and receiver are initialized to the same state.[3] At the transmitter the edges of the state diagram correspond to the data bit, b, to encode and the interconnect wire, i, upon which to send a pulse, P_i. For the receiver the edges of the state diagram correspond to the pulse P_i that is received and the data bit, b, that is subsequently decoded. Since the receiver receives these pulses asynchronously, it automatically adjusts its decoding bit rate to the encoding bit rate at the transmitter. Tables 1 and 2 summarize the state transition tables for the transmitter and receiver respectively. It should be noted that in the three-wire protocol, both the transmitter and receiver must keep track of their current state to encode and decode the transmitted data. The three-wire protocol we implement has the property that two successive pulses (i.e., state transitions) will not be sent on the same wire, simplifying the decoding circuitry at the receiver. Figure 4 shows waveforms of the three-wire protocol transmitting an example bit stream along with the corresponding states of the three-wire protocol.
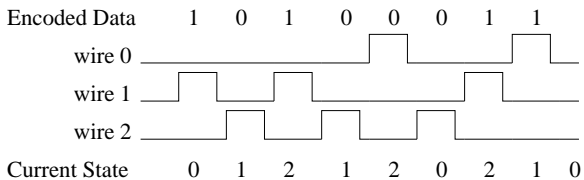

Figure 4. High-speed three-wire asynchronous signaling.

3. Token-Ring Transceiver Architecture

We use token-rings to design high-speed asynchronous multiplexed transceivers. A token-ring consists of n concurrent processing elements connected in a ring, with the output of process i connected to the input of process $(i + 1) \bmod n$. While in general token-rings can have one or more tokens traveling around the ring, we will use rings

[3]We initialize our circuits to "State 0".

with only one token in this paper. Token-rings are common asynchronous structures that are useful for implementing distributed mutual exclusion in concurrent VLSI architectures [15, 19]. Mutual exclusion is necessary in token-rings when elements in the ring can communicate with a common shared resource. To implement distributed mutual exclusion in the token-ring, access to shared resources is restricted to when elements exclusively have the token.

We construct a N:1 multiplexed transceiver with two independent N-element token-rings, one token-ring at the transmitter and one at the receiver. Each element in the token-ring is a single transceiver and communicates with a non-shared bit slice of the on-chip parallel data path and a shared off-chip serial bus. Since each element (transceiver) in the token-ring must sequentially access the shared serial bus, the token traveling around the ring naturally implements this out-of-phase access and ensures distributed mutual exclusion on the serial bus. As discussed in Section 2, each transceiver element requires the current state of the three-wire protocol to encode or decode data values. By having each transceiver compute the current state and send it on the token channel to the next transceiver in the ring, the token provides a convenient method for distributed state update in the three-wire protocol and we subsequently refer to this token as a *state-token*. In our token-ring implementation of multiplexed transceiver architectures, it is easiest to encode the token channel as a 1-of-3 channel to communicate one of three possible states in the three-wire protocol.
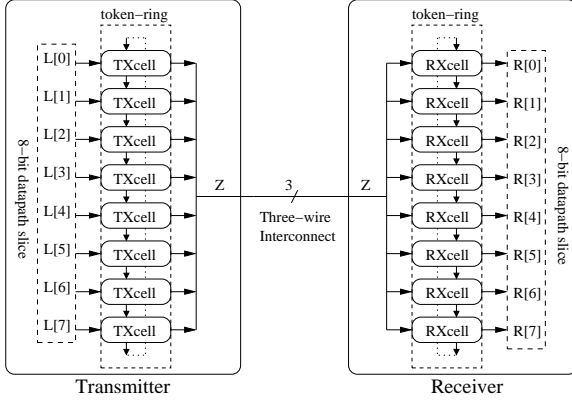
Figure 5. Basic 8:1 multiplexed transceiver architecture.

Figure 5 shows a basic three-wire protocol implementation using an asynchronous, multiplexed, token-ring architecture. The serial transmitter uses an eight element token-ring that multiplexes parallel data ($L[0..7]$) onto a shared bus (Z) that drives the three interconnect wires. The state-token passed around the ring is the current state value of the three-wire protocol. Each element of the token-ring contains an identical transmitter circuit and the state-token controls which transmitter is actively driving the shared bus. The period of a token is equal to N times the inverse of the transmitted bit rate and each transmitter needs a local cycle time of at least $1/N$ of the bit rate. In our target technology ($0.18\mu m$ CMOS) the token-ring throughput is determined by how fast each individual stage can drive the shared bus without suffering signal integrity problems, which in our circuit design is independent of how optimally the handshakes complete between token-ring elements. As a side result, the number of elements in the token-ring does not need to be throughput-optimized and so the number can be any reasonable value (limited by the wiring capacitance of the shared bus). In the example shown in Figure 5 a token-ring of size eight is chosen for convenience of interfacing with a 32-bit on-chip data path. The serial receiver uses a similar eight element token-ring and shared bus (Z) to reconstruct parallel data ($R[0..7]$) from the serial interconnect. Like the transmitter, each element of the token-ring contains an identical receiver circuit and the state-token controls which receiver is sampling the shared bus. In both the transmitter and receiver, the token-ring circuit elements that operate on the shared buses utilize conservative timing assumptions (non-QDI) to achieve high serial bit rates and are discussed in detail in subsequent sections.

4. Circuit Implementation

We assume the three-wire interconnect is built from a standard printed circuit board using either microstrip or stripline traces. Various transmission-line signaling schemes are possible with the three-wire protocol, including standard-CMOS, series-terminated, parallel-terminated, or Gunning Transceiver Logic (GTL) [2, 10]. We decided against differential signaling because it doubles the amount of required pins and interconnect signals. In this paper, a ground-referenced (i.e., single-ended) signaling scheme with parallel-terminated transmission lines and open-drain drivers is used for the interconnect design. Figure 6 shows an example of this interconnect style. A disadvantage of this scheme is the high static power dissipation when the transmission line is being driven low. GTL type interconnects have better noise immunity but suffer from the same static power dissipation problem and have more complicated interface circuitry than the parallel-terminated interconnect we chose to implement.
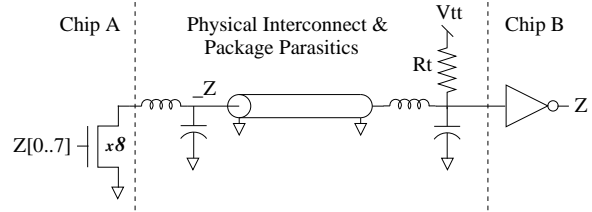

Figure 6. Electrical interconnect configuration.

Figure 6 illustrates how an open-drain, single-ended signaling scheme can be used to implement one wire of the asynchronous three-wire protocol. The transmitter on *Chip A* drives the interconnect wire with eight open-drain NMOS transistors (one for each TXcell) and the receiver on *Chip B* uses an inverter to restore the transmitted signal to on-chip voltage levels. The transmitted waveforms have their logical senses inverted. The *Rt* resistor terminates the transmission line through *Vtt*, which can be different than the on-chip voltage. This circuit connection both minimizes signal integrity distortion on the transmission line and also restores the transmission line to a *Vtt* voltage level when it is not being driven low by the NMOS drivers. As a result, there is no static power dissipation when the multiplexed transmitter is not transmitting data (i.e., when the NMOS transistors are not being driven).

We describe clockless transceiver circuits using Communicating Hardware Processes (CHP), whose syntax is summarized in the appendix, and construct them using Martin's synthesis method [16].

4.1. Multiplexed Transmitter

The CHP for the multiplexed transmitter process is:

$$TXcell \equiv *[\,U?s, L?x; \;\; y := f_s(x,s); \;\; data\_tx(Z,y), D!y\;]$$

The transmitter first receives the current state-token (s) from the previous token-ring cell on U and the data bit (x) to encode on L. It then computes (f_s) the next state value, copies this value on D, and the $data\_tx$ function transmits the value on the interconnect-driver wires (Z) that drive the open-drain drivers in Figure 6. While the peak token-ring frequency determines the maximum achievable bit rate, the actual transmitted bit rate is data-dependent on how fast the on-chip transmitter core supplies data to the L channel.

We use *process decomposition* [18] to break the TXcell process into a non-QDI process, Tdriver, that contains the $data\_tx$ implementation and a QDI process, Tproc, that contains the f_s computation and other channel accesses. The Tproc and Tdriver processes can be constructed as follows:

$$Tproc \;\equiv\; *[\; U?s, \;\; L?x; \;\; T!f_s(x,s) \;]$$
$$\|\;\; Tdriver \;\equiv\; *[\; T?y; \;\; data\_tx(Z, y), \;\; D!y \;]$$

Although the D communication can logically be at the end of either the Tproc or the Tdriver processes, we place it in the Tdriver process so we can more easily control the bit rate of the transmitter (by placing delay-lines on either the T or D channels). The final Tcell process is shown in Figure 7. The $BitWidth$ and $BitRate$ signals in the figure are analog control signals (to be discussed shortly) that are used to control timing margins for the circuits that implement the $data\_tx$ function.
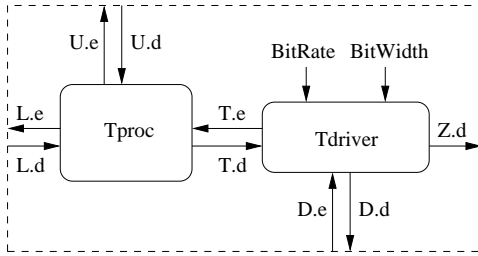

Figure 7. Multiplexed transmit cell (TXcell).

Given the manner in which the Tproc and Tdriver processes were decomposed, their circuit implementations can be compiled independently [18]. We compile Tproc as a standard QDI precharge-half-buffer (PCHB) process [14] and it is shown in Figure 8. The production rules for the pull-down computation of f_s are:

$$en \wedge (L.0 \wedge U.2 \vee L.1 \wedge U.1) \;\rightarrow\; \_T.0\downarrow$$
$$en \wedge L.1 \wedge (U.0 \vee U.2) \;\;\;\;\;\;\;\;\; \rightarrow\; \_T.1\downarrow$$
$$en \wedge L.0 \wedge (U.0 \vee U.1) \;\;\;\;\;\;\;\;\; \rightarrow\; \_T.2\downarrow$$

Tdriver is more interesting to compile because it is a non-QDI process and needs to generate a self-resetting waveform on the interconnect-driver wires (Z) as well as control the transmission bit rate. We also need to guarantee that Z is

not driven high, consequently activating the open-drain interconnect drivers, when no data is being transmitted. The internal circuits in this process are QDI, but the Z channel is non-QDI because its output is not acknowledged.
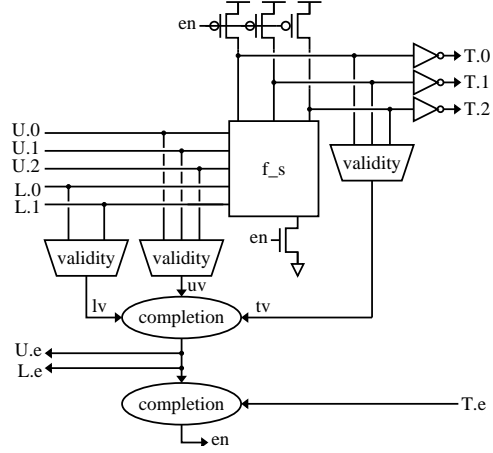

Figure 8. Tproc circuit implementation.

The initial handshaking expansion for the Tdriver process is given by:

$$*[[T_{DL}]; \;\; [v(T)]; \;\; Z := y; \;\; [v(Z)];$$
$$([Z_{DL}]; \;\; Z \Downarrow),$$
$$(D := y; \;\; [\neg D.e]; \;\; T.e\downarrow; \;\; [n(T)]; \;\; D \Downarrow);$$
$$[D.e]; \;\; T.e\uparrow$$
$$]$$

Z_{DL} is a delay-line, whose adjustable latency is controlled by the voltage on the *BitWidth* signal, and sets the bit width of the output bit stream by controlling the pulse width on the Z data wires. Likewise T_{DL} is a delay-line, whose latency is controlled by the *BitRate* signal, sets the bit rate of the output bit stream by controlling the speed of the handshake on T. We use *process factorization* [17] to break this handshaking expansion into two concurrent parts:

$$*[[\neg D.e]; \;\; T.e\downarrow; \;\; [D.e]; \;\; T.e\uparrow]$$
$$\|$$
$$*[[T_{DL}]; \;\; [v(T)]; \;\; Z := y; \;\; [v(Z)];$$
$$([Z_{DL}]; \;\; Z \Downarrow), \;\; (D := y; [n(T)]; \;\; D \Downarrow)$$
$$]$$

The first part contains all of the enable signals and is simply compiled as a wire. The second part contains the remaining logic necessary to implement Tdriver and we focus on optimizing this in the rest of this section. The initial handshaking expansion of the second part of Tdriver first raises Z and then in parallel lowers Z and performs a handshake on D. If the "$D := y$" rule is moved earlier in the handshake then a state variable can be eliminated and a more efficient handshake expansion results:

$$*[[T_{DL}]; \quad [v(T)];$$
$$Z := y; \quad [v(Z)]; \quad (D := y, [Z_{DL}]);$$
$$Z \Downarrow; \quad [n(T)]; \quad D \Downarrow$$
$$]$$

It should be noted that this reshuffling weakly couples Z_{DL} with T_{DL} since "$Z \Downarrow$" must now wait for both the "$D := y$" and "$[Z_{DL}]$" rules. While this prevents Z_{DL} from independently setting the pulse width on the Z data wires, the pulse width can still be modified indirectly by also adjusting T_{DL}, which will indirectly control the speed of the "$D := y$" rule. This slightly limits the adjustability of the bit width and bit rate, but in our circuits the ranges are acceptable to justify this more efficient reshuffling.
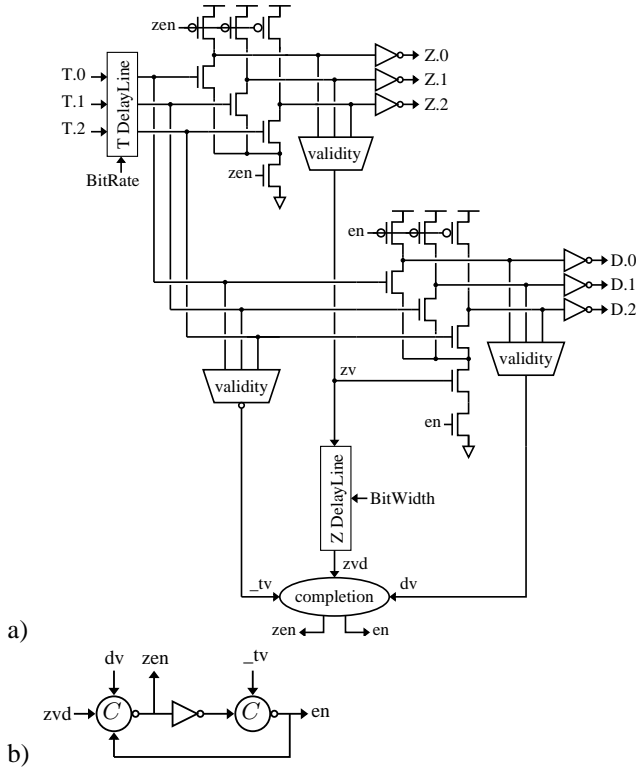

Figure 9. (a) Tdriver circuit implementation, (b) non-standard completion circuit.

The compiled production rules for this handshake expansion are shown in Figure 9. The two delay-lines in Figure 9a control all of the timing assumptions in the multiplexed transmitter design. The delays of the Z_{DL} and T_{DL} delay-lines should be adjusted such that the bit rate and bit width are sufficient for the demultiplexed receiver to decode the transmitted bit pulses. The pull-down for the precharge computation block of D is gated by zv to enforce that "$Z := y$" is ahead of "$D := y$" in the handshaking expansion. Figure 9b gives the non-standard completion circuit that is used to allow the Z data rails to precharge

without waiting for the handshake to complete on D.

4.2. Demultiplexed Receiver

The CHP for the demultiplexed receiver process is:

$$RXcell \equiv *[U?s; \quad z := s;$$
$$*[z = s \longrightarrow z := data\_rx(Z)]; D!z, R!f_x(s, z)$$
$$]$$

The receiver first receives the current state-token (s) from the previous token-ring cell on Channel U and then samples the shared state wires (Z) until the next state[4] (z) arrives from the off-chip interconnect. It then copies the next state value to the next cell on Channel D, and computes (f_x) the encoded data value based on the current and next states of the three-wire protocol and sends this value on Channel R. The $data\_rx$ function samples the Z channel until a valid data token appears on it and then immediately returns the data value. While the peak token-ring frequency determines the maximum decodable bit rate, the actual decode rate is data-dependent on how fast the transmitter sends bits and on how fast the on-chip receiver core processes data on the R channel.

We would like to decompose the RXcell process into a non-QDI process, Rsamp, that contains the loop that samples the state wires and a QDI process, Rproc, that contains all of the other computation. To facilitate this, RXcell is transformed to:

$$*[[\overline{U} \longrightarrow s := U; \quad z := s;$$
$$*[z = s \longrightarrow z := data\_rx(Z)]$$
$$];$$
$$y := f_x(s, z); \quad U?s; \quad R!y, D!f_s(s, y)$$
$$]$$

where f_s computes the the next state based on the current state and the encoded data value (y). The Rsamp and Rproc processes can then be decomposed as follows:

$$Rsamp \equiv *[[\overline{U} \longrightarrow s := U; \quad z := s;$$
$$*[z = s \longrightarrow z := data\_rx(Z)]$$
$$];$$
$$T!f_x(s, z)$$
$$]$$
$$Rproc \equiv *[T?y, U?s; \quad R!y, D!f_s(s, y) \ ]$$

The final RXcell process is shown in Figure 10.

Since the Rsamp and Rproc processes share the data rails of the U Channel, their circuits must be compiled together to ensure that the data rails of U are valid for the entire time the probe selection statement is executing in Rsamp.

---

[4]The next state is guaranteed by the three-wire protocol to be different than the current state and so the receiver will never run ahead of the transmitter.
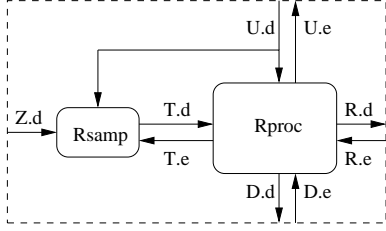
Figure 10. Demultiplexed receive cell (RXcell).

Rsamp is simpler than the CHP may first appear and is implemented as a latched sense amplifier, as shown in Figure 11. The sense amplifier activates and drives the T channel data rails when the current state-token, the next state value from the interconnect, and the T channel enable signal (T.e) are all valid. When the T.e signal is low the sense amplifier resets itself.
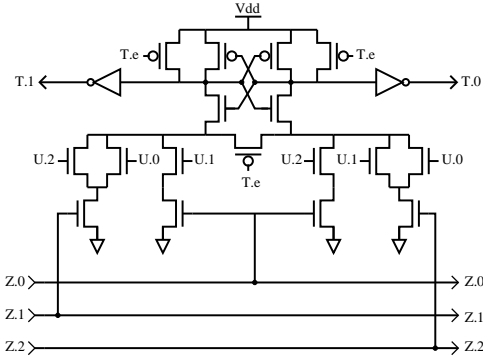

Figure 11. Rsamp circuit implementation.

Rproc is compiled as a normal QDI PCHB process except to implement the probe on U correctly, the T.e signal must be driven based on the validity and neutrality of U (rather than a normal PCHB handshake reshuffling that computes T.e from the state of the T channel). Also we do not check the neutrality of the T data rails (as Rsamp has 7 full cycles to reset itself). The Rproc circuit is shown in Figure 12 and the production rules for the pull-down computation of f_s are:

$$en \wedge (T.0 \wedge U.2 \vee T.1 \wedge U.1) \rightarrow \_D.0\downarrow$$
$$en \wedge T.1 \wedge (U.0 \vee U.2) \rightarrow \_D.1\downarrow$$
$$en \wedge T.0 \wedge (U.0 \vee U.1) \rightarrow \_D.2\downarrow$$

The major timing assumption in this receiver design is that a receive cell can process a bit from the interconnect and pass control to the next cell in time for it to process the next bit. In general, the receiver's token-ring must have a peak bit-processing throughput equal to or greater than the maximum bit rate of the three-wire interconnect. This is a one-sided timing assumption only, as the transmitter can transmit at an arbitrarily slow bit rate. If the interconnect
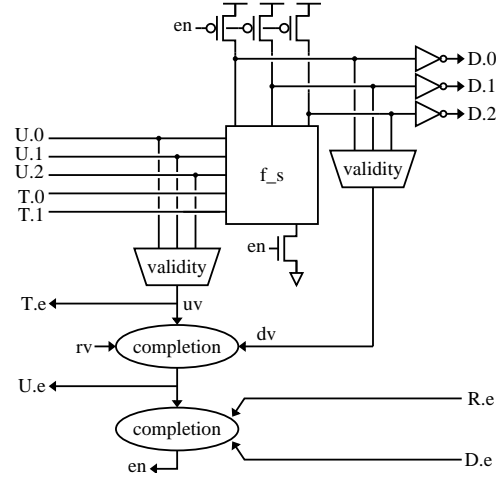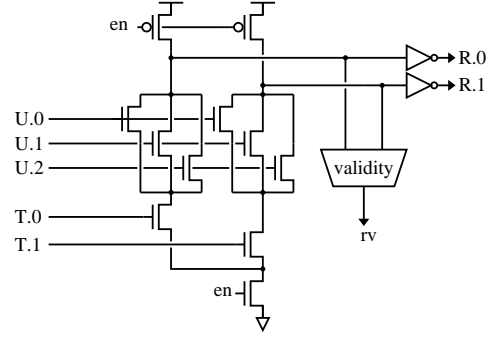

Figure 12. Rproc circuit implementation.

bit rate is too high for the receiver, transmitted state bits will be mis-sampled and the three-wire state protocol at the receiver will get out of synchronization with the state machine at the transmitter. This will not cause circuit failure, but will prevent any further data values from being decoded correctly.

A conservative link design would adjust the delay-lines in the transmitter to ensure that bit pulses on the three-wire interconnect are mutually exclusive. However, the receiver design we use allows this timing constraint to be slightly relaxed since its circuitry uses sense amplifiers to decode the incoming bit stream and only samples a small portion of the bit's waveform. If the receiving circuitry operates fast enough to decode each transmitted bit upon its leading edge transition,[5] then strict mutual exclusion need only be enforced for every *other* bit pulse. Only every other bit requires mutual exclusion because each RXcell has a priori information about the previously decoded bit from the prior cell and hence knows to wait for a new bit to arrive, which by the three-wire protocol is guaranteed to be different than the previously decoded bit. This relaxed timing constraint

---

[5] Our simulations show that this is true for our design, as the receiver circuitry (due to its lesser complexity) can operate at a higher frequency than the transmitter circuitry.

scheme still requires strict ordering on the *leading edges* of transmitted bits along the three wires, as well as enough delay between the leading edges of consecutive bits so that the sense amplifiers have time to latch the current bit and stabilize before the leading edge of the next bit arrives at the receiver.

4.3. Simulation Results

The multiplexed transceiver circuits presented in the previous sections have been designed and laid out in TSMC's $0.18\mu m$ CMOS process (FO4 delay≈65ps) and simulated in SPICE. Both the on-chip multiplexed driver and demultiplexed receiver circuits have simulated functionally at bit rates up to 3 Gbps (bit-width ≈ 5.1 FO4 delays). The transmitter is the speed-limiting factor in this design, due to the circuit overhead necessary to generate self-timed bit pulses on the interconnect wires. While all on-chip parasitics were included in the simulations, only a simple spice model (no transmission line effects) was used for the off-chip path. Using more sophisticated models would allow the transceiver circuits to be tuned for application-specific interconnect scenarios. Figure 13 shows a simulated waveform at the open-drain output of the multiplexed driver circuits with $Vdd = 1.8$V, $Vtt = 1.8$V, $R_t = 50\Omega$, and 500fF of pad and pin capacitance [1].
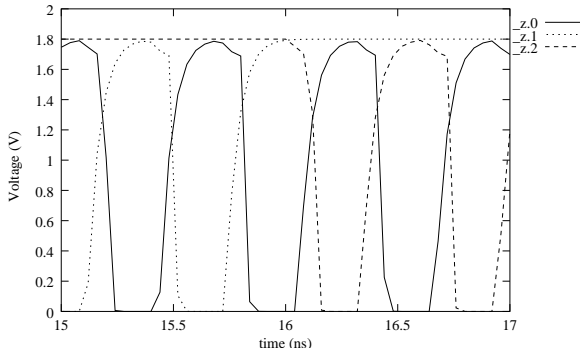

Figure 13. Three-wire multiplexed transmitter output.

The link consumes 23.5pJ/bit, or 76.5mW when running at full throughput. 77% of the power is consumed through the pull-up resistor. When Vtt is reduced to 1.2V, the link consumes 16.4pJ/bit, or 53.3mW when running at full throughput. 66% of the power is consumed through the pull-up resistor. The large amount of power consumed through the pull-up resistor in the open-drain driver signaling configuration indicates that additional power savings in this design can be obtained by using more energy-efficient (and sophisticated) electrical signaling schemes in the front-end, interconnect stages of the multiplexed transceivers.

The token-ring receiver core area is 600 x 1400 λ^2 (54 x 126 μm^2) and the token-ring transmitter core area is 600

x 2000 λ^2 (54 x 180 μm^2), which is competitive with the smallest synchronous links [13].

5. Token Resynchronization

Bit errors may occur as a result of unsustainable bit rates, signal degradation on the interconnect, or anything that violates the timing assumptions at the demultiplexed receiver. In asynchronous multiplexed architectures, bit errors will cause one or more transmitted bits to be lost. These lost bits will desynchronize the three-wire state encoding and decoding token-rings at the transmitter and receiver, preventing subsequent data bits from being decoded correctly. This is in contrast to a synchronous system, where a bit error will simply result in an incorrect data value and not cause subsequent data values to be decoded incorrectly.

For illustrative purposes in this section we will use the simple asynchronous link configuration[6] that is shown in Figure 14. C is a channel, of any standard asynchronous handshake, that gives control-flow feedback information to the transmitter concerning the status of the receiver (e.g., how many bits to transmit, the amount of buffer space in the receiver, etc.). We assume that tokens on channel C are queued, such that the transmission of bits on the serial link and the sending of control tokens will occur in parallel. Without loss of generality, in this example we assume that the control tokens sent on C simply tell the transmitter to send 8 bits of data and we call this a "*normal*" control token. As long as control information is still being sent on C and bits are transmitted on the link, this link will not deadlock when a bit error occurs. However, after a bit error occurs the decoded data bits will be incorrect and will be misaligned by a bit.
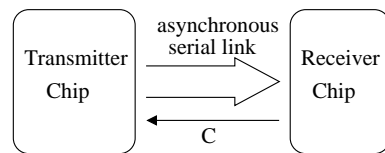

Figure 14. Example link configuration.

For an asynchronous link to successfully recover from a bit error, the state-token in both the transmitter and receiver needs to be re-initialized and the token reset to its initial starting position in the token-ring. This process, which we refer to as *token resynchronization*, requires that the transmitted data be tagged with bits generated from error correcting codes (ECC) so that the demultiplexed receiver can detect bit errors. We assume these ECC bits are part of the transmitted data stream and transparent to the token-ring transceiver circuits. To facilitate re-initializing the state-

---

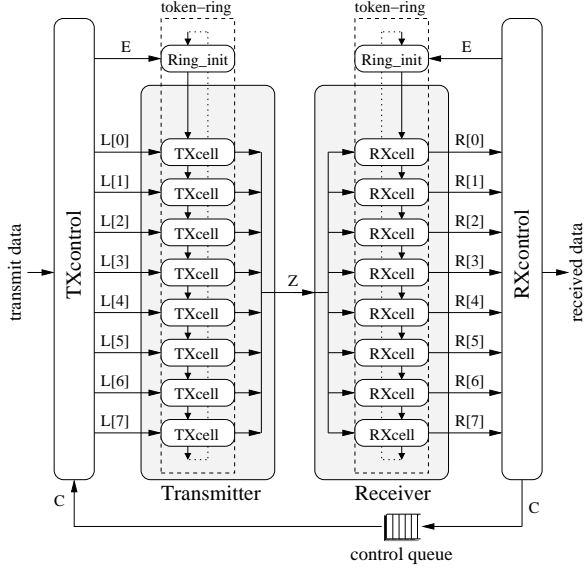[6]A study of various asynchronous link configurations is in [25].

Figure 15. Multiplexed transceiver and control circuits modified to support token resynchronization.

token, we insert the following initialization process at the head of the transmitter and receive token-rings:

$$Ring\_init \equiv$$
$$*[\,U?s, E?e;\ [\neg e \longrightarrow D!s [\!] e \longrightarrow D!''reset\_state'']\,]$$

where $''reset\_state''$ is the reset value of the state-token, E is an error channel that determines when the token should be re-initialized, and the U and D token channels are connected to adjacent token-ring elements in an analogous manner to the transceiver token-ring elements. The E channel also provides a convenient control mechanism to stall the progression of the state-token that will be exploited in the resynchronization control circuitry. Since the placement of the $Ring\_init$ process is symmetric for both the transmit and receive token-rings, the small additional latency for the token to pass through this process will not limit the peak bit rate, although the average bit rate may be slightly lower.

Figure 15 shows the modified token-rings with our multiplexed transceiver architecture. The control queue on channel C allows the transceiver to have multiple outstanding control tokens and prevents deadlock when bit errors occur. At reset, this queue is filled with a finite number of "$normal$" control tokens. We assume that the $RXcontrol$ process has enough data buffering on the R channels to buffer all outstanding data requests.

The circuits that control the multiplexed transceivers in an asynchronous link need to be slightly modified to support token resynchronization. We give the following code fragment for the process that controls the receiver:

$$RXcontrol \equiv *[\ ...\ R?data;\ err := ecc\_error(data);$$
$$[\neg err \longrightarrow E!\textbf{false}, C!''normal''$$
$$[\!] err \longrightarrow (i : depth : C!''sync'';)$$
$$E!\textbf{true};$$
$$(i : depth : C!''normal'';)$$
$$]; \ ...\ ]$$

where ecc_error is an ECC function that returns true if a bit error is detected in the received data and $depth$ is the depth of the control queue (i.e., the maximum number of outstanding control tokens). This control process operates similarly to the normal multiplexed architecture when no error has been detected, except it sends **false** on the new E channel in the receiver's token-ring. When the receiver detects a bit error we send "$sync$" control tokens on C to flush the control queue of preceding "$normal$" tokens that will transmit data to the receiver. During this flush the receiver's token-ring is stalled (by not communicating on E) because we know that subsequent transmitted data bits will be decoded incorrectly and hence they will not be decoded at all by the receiver. After all of the "$normal$" control tokens have been flushed, we send **true** on E to reset the state-token at the receiver and the receiver is ready for normal operation again. Normal operation resumes by filling the queue on channel C with "$normal$" control tokens.

The process that controls the multiplexed transmitter is modified by means of the following code fragment to support token resynchronization:

$$TXcontrol \equiv *[\ ...\ C?c;$$
$$[c = ''normal'' \longrightarrow$$
$$[\neg init \longrightarrow E!\textbf{false}, L!data$$
$$[\!]\ init \longrightarrow E!\textbf{true}, L!data;\ init\!\downarrow$$
$$]$$
$$[\!] c = ''sync'' \longrightarrow init\!\uparrow$$
$$]; \ ...\ ]$$

where $init$ is false upon reset. During normal operation the transmit control process sends **false** on E and passes data along to the L channels of the transmitter to be transmitted on the link. When the transmit control process receives a "$sync$" control token on C, the receiver's token-ring has been stalled due to a bit error and so we stall the transmitter as well. After the control queue has been flushed (i.e., when a "$normal$" control token is received) the transmit control process resets the state-token at the transmitter by sending **true** on E and the transmitter resumes normal operation.

6. Related Work

Asynchronous three-wire inter-chip interfaces first appeared in [23], where a three-wire wired-or signaling scheme was used for control handshaking on a multiple receiver bus. Røine proposed [21] and implemented [22] a three-wire protocol suitable for high-speed interfaces that

is similar to the three-wire protocol we used in this paper. However, his three-wire link uses on-chip serial input and output streams that are in the two-phase DS-format,[7] whereas our three-wire link uses on-chip parallel four-phase input and output streams. In contrast to the novel distributed-state token-ring we used in this paper for interfacing with on-chip parallel data, his design uses simpler combinational logic to implement the states of the three-wire protocol because his link interfaces with on-chip serial data. He does not address the overhead and additional latency required to convert on-chip parallel data paths to the serial two-phase DS-format, which we believe to be both a performance penalty and less practical than our three-wire link that interfaces directly with high-performance, on-chip, parallel data-path circuits. While our design can save energy by shutting down and not transmitting data across the interconnect, Røine's design must always be transmitting data because the level shifter on his receiver fails when data is transmitted below a certain minimum rate.

Asynchronous two-wire protocols are also possible for use in off-chip interfaces but do not have the property that two successive state transitions occur on different wires (as in three-wire protocols). This makes the two-wire protocol receiver circuitry more complex and prone to timing errors, especially in a demultiplexed receiver architecture that has multiple decoders operating in parallel on the same serial data stream. Traditional two-phase data encodings can be used in two-wire links, where toggling one wire indicates a logical zero and toggling the other wire indicates a logical one. An alternative two-phase, two-wire data-encoding scheme called Level-Encoded Dual-Rail [6] uses two distinct states per logic value. It uses a four state Gray-code to encode data such that when one of the wires toggles, at least one of the two wires always carries the logical value of the transmitted data bit (the Gray-code determines which wire). This scheme has slightly simpler decoding circuitry but more complicated encoding circuitry than traditional two-phase data encodings. An asynchronous differential two-wire protocol using 3-level voltage signaling was proposed in [24], however the 3-level voltage signaling scheme is impractical as it is much more prone to receiver error than the three-wire protocol because of its sensitivity to noise and slew rates on the interconnect signals. While two-wire protocols use one less wire than three-wire protocols, they are inherently more complicated to implement and this complexity limits their achievable bit rates in the asynchronous transceiver architecture described in this paper.

High-speed synchronous link examples, as described in Section 1, are numerous in the literature (e.g., [5, 12, 26,

27]). The main difference between clocked and clockless transceivers is that synchronous designs require complicated clock generation and synchronization circuitry. While an asynchronous three-wire transceiver need only ensure the signal integrity of the state pulses and their relative skew, a synchronous transceiver must ensure the relative timing skew and signal integrity between the clock and data signals. Generating clean multi-phase clocks with low phase offsets and low jitter requires significant design effort. In a synchronous transceiver the clock should be continuously driven from the transmitter to the receiver so that the two chips remain synchronized, as resynchronizing the receiver's DLL or PLL has a high latency cost. This limits the transceiver to transmitting at a single bit rate and is a power waste when no data is transmitted across a synchronous link. Our clockless transceiver, however, consumes no power when data is not being transmitted and can start and stop transmitting bits without suffering from a synchronization latency. Additionally, our clockless transceiver does not need the multi-phase to single-phase data conversion circuitry, which is a large fraction of the latency in synchronous links [9]. Recent synchronous high-speed links have used more complicated signaling schemes and rely on complex analog signal processing to either reduce power [3, 13] or greatly increase the symbol rate [4, 7]. While the asynchronous link presented in this paper cannot compete with the bit rates in these very high-speed synchronous links, we expect that it is possible to apply some of the complex analog signaling schemes used in clocked links to the front end of our asynchronous transceiver architecture and achieve comparable performance improvements.

7. Conclusion

Instead of attempting to design a clockless multiplexed transceiver with a superior bit rate than a clocked transceiver, our goal was to design a clockless architecture with a relatively high bit rate that avoided the complexities of clocked architectures (e.g., multi-phase clock generation and synchronization circuitry). By using asynchronous token-rings to control data multiplexing and demultiplexing, this transceiver exploits parallelism at the transmitter and receiver to drive off-chip I/O pins at high bit rates. Our clockless transceiver supports a data-dependent bit rate and the receiver dynamically self-adjusts its sample rate to match the transmitted bit stream. Since timing assumptions were kept to a minimum in this design, our clockless transceiver enables the construction of robust, high-speed inter-chip communication links for asynchronous VLSI systems.

---

[7]Two-phase DS-format is a two wire serial signaling protocol, where the D-signal carries serial data and the S-signal is a parity signal that indicates transitions between identical data bits on the D-signal [21].

Acknowledgments

A   Summary of CHP Notation

The CHP notation we use is based on Hoare's CSP [11]. A full description CHP and its semantics can be found in [16]. 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: $[G1 \rightarrow S1 \; [] \; ... \; [] \; Gn \rightarrow Sn]$, 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 "[]."

- Repetition: $*[G1 \rightarrow S1 \; [] \; ... \; [] \; Gn \rightarrow Sn]$. 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.

- Simultaneous Composition: $S \bullet T$ both S and T are communication actions and they complete simultaneously.

Additionally, we use the notation $X := x$ to indicate that the data rails of channel X are set to a valid value x, and the notation $X \Downarrow$ to indicate that the data rails of channel X are set to the neutral value. $v(X)$ denotes the validity of channel X and $n(X)$ denotes the neutrality [20].

References

[1] http://www.asat.com/mosis

[2] H. B. Bakoglu. *Circuits, Interconnections, and Packaging for VLSI.* Addison Wesley, 1990.

[3] K. K.-Y. Chang, W. Ellersick, T.-S. Chuang, S. Sidiropoulos, M. Horowitz. A 2 Gb/s/pin CMOS asymmetric serial link. *VLSI Circuits Symposium*, June 1998, pages 216-217.

[4] W. Dally and J. Poulton, "Transmitter Equalization for 4 Gb/s signaling," *IEEE Micro* Jan/Feb 1997, pages 48-56.

[5] W.J. Dally, M.-J. E. Lee, F.-T. R. An, J. Poulton, and S. Tell. High Performance Electrical Signaling. *MPPOI98*, 1998.

[6] M. Dean, T. Williams, and D. Dill. Efficient self-timing with level-encoded 2-phase dual-rail (LEDR). *Advanced Research in VLSI: Proceedings of the 1991 UC Santa Cruz Conference*, 1991.

[7] R. Farjad-Rad, C.-K.K. Yang, M. Horowitz. A 0.3-um CMOS 8-Gb/s 4-PAM serial link transceiver. *IEEE Journal of Solid State Circuits*, May 2000, pages 757-764.

[8] S.B. Furber, A. Efthymiou, and Montek Singh. A Power-Efficient Duplex Communication System. *Workshop on Asynchronous Interfaces: Tools, Techniques and Implementations (AINT-2000)*, July 2000.

[9] M. Galles. Scalable Pipelined Interconnect for Distributed Endpoint Routing: the SGI SPIDER Chip. *Proc. Symp. High Performance Interconnects (Hot Interconnects 4)*, August 1996.

[10] B. Gunning, L. Yuan, T. Nguyen, T. Wong. A CMOS low-voltage-swing transmission-line transceiver. *IEEE International Solid-State Circuits Conference*, 1992.

[11] C.A.R. Hoare. Communicating Sequential Processes. *Communications of the ACM*, **21**(8):666–677, 1978.

[12] M. Horowitz, C.-K. K. Yang, S. Sidiropoulos. High-speed electrical signaling: overview and limitations. *IEEE Micro*, January 1998, pages 12-24

[13] M.-J. E. Lee, W. Dally, P. Chiang. Low-Power Area-Efficient High-Speed I/O Circuit Techniques. *IEEE Journal of Solid-State Circuits*, November 2000, Vol. 35, No. 11, pages 1591-1599.

[14] A.M. Lines. *Pipelined Asynchronous Circuits*. M.S. Thesis, California Institute of Technology, 1996.

[15] A.J. Martin. Distributed Mutual Exclusion on a Ring of Processes. *Science of Computer Programming*, 5, 265-276, 1985.

[16] A. J. Martin. Compiling Communicating Processes into Delay-insensitive VLSI circuits. *Distributed Computing*, **1**(4), 1986.

[17] A.J. Martin. Formal Program Transformations for VLSI Circuit Synthesis. In E.W. Dijkstra, editor, *Formal Development of Programs and Proofs*, UT Year of Programming Series, pp. 59–80, Addison Wesley, 1989.

[18] A.J. Martin. Synthesis of Asynchronous VLSI Circuits. *Formal Methods for VLSI Design*, J. Staunstrup, Ed. North-Holland, 1990.

[19] A.J. Martin. *Asynchronous Circuits for Token-Ring Mutual Exclusion*. Caltech Computer Science Technical report CS-TR-90-09.

[20] A.J. Martin. Asynchronous Datapaths and the Design of an Asynchronous Adder. *Formal Methods in System Design*, **1**:117–137, 1992.

[21] P.T. Røine. A System for Asynchronous High-speed Chip to Chip Communication. *Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems*, 1996.

[22] P.T. Røine. *Performance of Synchronous and Asynchronous High-Speed Links: A Practical Experiment*. PhD thesis, Dept. of Informatics, University of Oslo, May 2000.

[23] I.E. Sutherland, C.E. Molnar, R.F. Sproull, and J.C. Mudge. The trimosbus. *Proceedings of the First Caltech Conference on Very Large Scale Integration*, 1979.

[24] C. Svensson and J. Yuan. A 3-level asynchronous protocol for a differential two-wire communication link. *IEEE Journal of Solid-State Circuits*, 1994.

[25] J. Teifel. *Interchip Communication in Asynchronous VLSI Systems*. M.S. thesis, Cornell University, May 2002. (Available as Cornell Computer Systems Lab Technical Report CSL-TR-2002-1027).

[26] C.-K. Yang and M. Horowitz. A 0.8um CMOS 2.5Gb/s oversampling receiver and transmitter for serial links. *IEEE Journal of Solid-State Circuits*. December 1996, pages 2015-2023.

[27] E. Yeung, M. Horowitz. A 2.4 Gb/s/pin simultaneous bidirectional parallel link with per-pin skew compensation. *Journal of Solid State Circuits*, November 2000, pages 1619-1628.