A 1-change-in-4 Delay-Insensitive Interchip Link

Anand Chandrasekaran and Kwabena Boahen
Bioengineering Department
Stanford University
{anandc,boahen}@stanford.edu

Abstract—**We present a 1-change-in-4 (1c4) link for interchip communication that extends level-encoded dual-rail (LEDR). LEDR transmits a bit on every transition by using all four 2-bit codewords, with the bit encoded by the current codeword (level-encoding) rather than the difference between it and the previous codeword (transition signaling). 1c4 transmits two bits on every transition by using all sixteen 4-bit codewords, preserving LEDR's level-encoding property. Delay-insensitive I/O pad implementations for 1c4 encoding and decoding are described. Measurements of this chip-to-chip link, fabricated in a 0.18μm CMOS process, yielded a peak data-rate of 315 Mb/s at 1.8V and an energy efficiency of 89.8pJ/bit.**

I. Delay-Insensitivity

Delay-insentivity (DI) requires every transition to be acknowledged. Therefore, another transition must make it back before the next transition can be sent out. Thus, round-trip delays, determined by the speed of light, the dielectric's permittivity, and the interconnect's length, set the maximum signaling rate—less than a gigahertz for traces longer than a few centimeters.[1] With this round-trip limit, maximizing the bit-rate comes down to minimizing the number of transitions used to transmit each bit.

Dual-rail (DR), the most widely used DI coding, uses four transitions to transmit each bit. One line is raised to send a 1; the other is raised to send a 0. Receipt is acknowledged by raising a third line. Two additional transitions are made to lower these lines before the next cycle starts. Thus, DR (aka 1-in-2) sends one bit per 4 transitions using a total of 5 pins—including two for power—delivering only 0.05 bits/transition/pin. 1-change-in-4 (1c4), first implemented in an CPLD in [1], improves this performance to 0.14 bits/transition/pin by extending Level Encoded Dual Rail (LEDR), first proposed in [2].

LEDR transmits a bit with two transitions—one for data and one for acknowledge—instead of four. This efficiency is achieved by using all four 2-bit codewords, and assigning them such that each is a bit-flip away from words for both bit-values. In odd phases, 01 encodes a 0 and 10 encodes a 1 (just like DR). In even phases, 00 encodes a 0 and 11 encodes a 1. The first line (MSB) always corresponds to the bit's value while the second line (LSB) ensures the correct parity. This line is toggled if the bit's value does not change; the other line is toggled if it changes.

By extension, 1c4 transmits two bits with two transitions by using all sixteen 4-bit codewords, assigned such that each is a

[1]Synchronous interchip links acheive gigabit/sec/pin rates by having several bits in flight on an interconnect, properly terminated to avoid reflections.

	Odd			Even	
00	0001	1110	00	0000	1111
01	0010	1101	01	0011	1100
10	0100	1011	10	0101	1010
11	0111	1000	11	0110	1001

Fig. 1. Binary to 1-change-in-4 Conversion. Each of 4 data-values is assigned two 4-bit codewords and their complements. For decoding, the two middle bits yield the binary value (uninverted codewords). (From [1])

bit-flip away from words for all four data-values (Fig. 1). The second and third lines correspond to the bits' value; the first line (MSB) indicates whether or not this value is inverted; and the fourth line (LSB) ensures the parity is correct. This line is toggled if neither bit changes while the first line is toggled if both bits change; one of the other two lines is toggled if one bit or the other changes. Nowick et al. proposed the same coding scheme, dubbed it Level Encoded Transition Signaling (LETS), and generalized it to 1-change-in-N [3].

In addition to reducing the number of transitions, 1c4 reduces the number of pins by sharing power and acknowledge between two bits. There is no need to increase the number of power pads because only one line transitions at a time—just like in DR. Consequently, the number of pins is reduced from 10 to 7 (no 3-pin overhead for second bit), in addition to decreasing the number of transitions from 4 to 2 (same as LEDR), hence the threefold improvement in bits/transition/pin.

Section II describes our 1c4 coding algorithm. Section III describes the encoder and decoder circuits. Section IV presents test results and conclusions.

II. Butterfly Network for 1-change-in-4

Given the two databits (D1,D0) and the previous codeword (C3,..,C0), the following four mutually exclusive conditions determine which of the codeword's bits to toggle:

$$\begin{aligned} \mathrm{C0}&: \mathrm{D1}' = \mathrm{C2}\ \&\ \mathrm{D0}' = \mathrm{C1} \\ \mathrm{C1}&: \mathrm{D1}' = \mathrm{C2}\ \&\ \mathrm{D0}' \neq \mathrm{C1} \\ \mathrm{C2}&: \mathrm{D1}' \neq \mathrm{C2}\ \&\ \mathrm{D0}' = \mathrm{C1} \\ \mathrm{C3}&: \mathrm{D1}' \neq \mathrm{C2}\ \&\ \mathrm{D0}' \neq \mathrm{C1} \end{aligned}$$

where $\mathrm{D1}' := \mathrm{D1} \oplus \mathrm{C3}$ and $\mathrm{D0}' := \mathrm{D0} \oplus \mathrm{C3}$ (xors) inverts the databits before comparing if the previous code was inverted (C3=1). Conversely, decoding is straightforward: $\mathrm{C2} \oplus \mathrm{C3}$ and $\mathrm{C1} \oplus \mathrm{C3}$ extract D1 and D0, respectively.

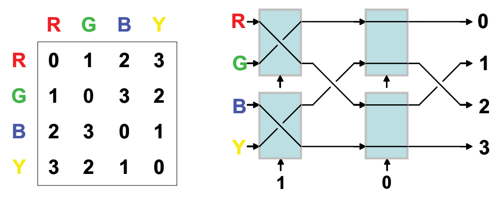

Fig. 2. Toggling bits. *Left* – Previous and current data (R=00, G=01, B=10, or Y=11) specify row and column—or vise versa. *Right* – Butterfly network toggles correct bit, given the current data (left, 1-of-4 code) and the previous data (bottom, binary code)—G in this example. Each block has two 2-to-1 muxes; the same signal controls all muxes in that tier.

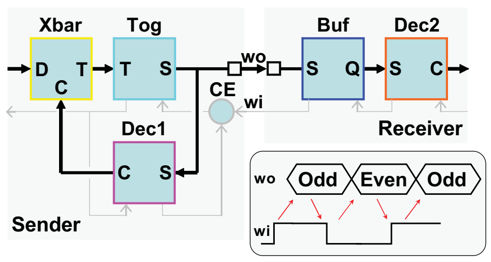

Fig. 3. Link's Function Blocks. Namely, butterfly (Xbar), toggle (Tog), feedback decoder (Dec1), C-element (CE), and feedforward decoder (Dec2); a buffer (Buf) was added to improve throughput. Acknowledges/enables are shown in gray. *Insert*–Two-phase handshake: The (active) Receiver takes wi high (low) to request an odd (even) word from the (passive) Sender.

A butterfly network can determine which bit to toggle. From the algorithm above, it is clear that whether the previous codeword was odd/even or inverted/uninverted is irrelevant. For instance, the fourth line always toggles when the databits change from 00 to 11 (see Fig. 1). All sixteen possibililities—called a bit-flip matrix in [3]—are shown in Fig. 2. The permutations from row to row (2nd row swaps neighboring bits, 3rd row swaps left and right halves, 4th row performs both swaps) are realized by a butterfly network (1st tier, 2nd tier, or both tiers, respectively).

The butterfly's two tiers are controlled by the previous data-value, obtained from the previous codeword by a decoder circuit identical to that at the link's receiving end (Fig. 3). The butterfly's output is captured by the toggle's internal state-variable, which is used to update the toggle's output once transmission of the previous codeword is complete. The alternative is a feedforward architecture, where, instead of decoding the previous codeword, storage is added to capture the previous data-value.

While storing the previous data-value appears simpler than decoding the previous codeword, the latter yields a more efficient and robust implementation. Efficiency arises from exploiting the toggles' internal state instead of introducing additional storage. Switching the input between the butterfly and a latch is also avoided, streamlining operation. Robustness arises from comparing the new data-value with the value that was actually sent instead of a stored value. Any discrepancy between the last two would persist until the system is reset—because the entire codeword is never updated in normal operation—individual lines are toggled instead.

TABLE I
HSE NOTATION

Operation	Notation	Explanation
Complement	`~v`	Inversion of `v`
And	`v & w`	High if both are high
Or	`v | w`	Low if both are low
Equal	`v = w`	High if equal
Set	`v+`	Drive `v` high
Clear	`v-`	Drive `v` low
Wait	`[v]`	Wait till `v` is high
Concurrent	`v+,w+`	≡ `v+,w+` in PRS
Sequential	`[u];v+`	≡ `u -> v+` in PRS
Assignment	`v:=w`	≡ `w => v` in PRS
Repetition	`*[v+;v-]`	Loops forever

III. LINK IMPLEMENTATION

To implement the link, we expanded the communications among its blocks (see Fig. 3) into handshaking sequences. Handshaking expansion (HSE) is the second step in Martin's synthesis procedure for quasi-delay-insensitive circuits [4].[2] An active port (A) communicates with a passive (P) port as follows:

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

(for notation, see Table I). That is, A asserts its request (`ao+`) and waits for the acknowledge (`[ai]`), then deasserts its request (`ao-`) and waits for the acknowledge to clear (`[~ai]`) before initiating the next handshake. Whereas P waits for the request (`[pi]`) before asserting its acknowledge (`po+`), then waits for the request to clear (`[~pi]`) before deasserting its acknowledge (`po-`). Alternatively, two-phase handshakes can be used:

```
# Two-phase #
*[ao:=~ao; [ai=ao]]
*[[pi=~po]; po:=~po]
```

That is, A toggles its request (`ao:=~ao`) and waits for the acknowledge to match (`[ai=ao]`) while P waits for a mismatch (`[pi=~po]`) before toggling its acknowledge (`po:=~po`). When data is involved, its validity (computed by `odd()` for two-phase handshakes) replaces the appropriate handshake signal (Fig. 3, *Insert*).

We reshuffled the HSE of Sender's three blocks to minimize the number of state-variables, while retaining enough slack to avoid deadlock in the feedback loop. Our final reshufflings are shown below (see Fig. 3 for port labels).

```
# Xbar -- 2-to-1 mux #
*[[di0&cif|di1&cit]; to0+; [ti]; co+,do+;
 [~di0&~cif&~di1&~cit];to0-;[~ti];co-,do-
 ]
# Tog -- Bit-slice 0 #
```

[2] *Quasi* reflects the assumption that branches have equal delays—isochronic forks—the sole timing assumption Martin's synthesis methodology makes.

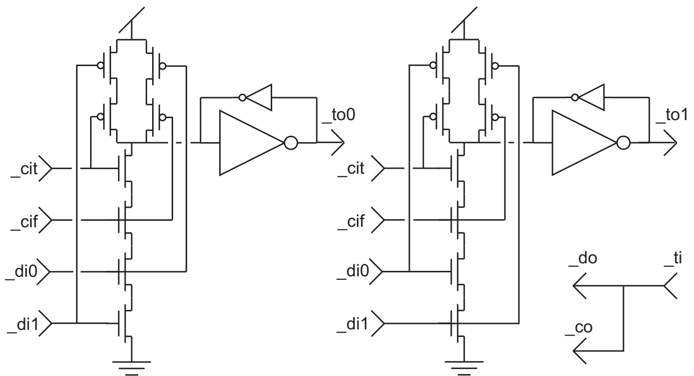

Fig. 4. Xbar. Connects _di0 and _di1 to _to0 and _to1, respectively, if _cif is low; swaps them if _cit is low. The weak feedback-inverter holds state; _to0 and _to1 are reset high (circuitry omitted for simplicity).

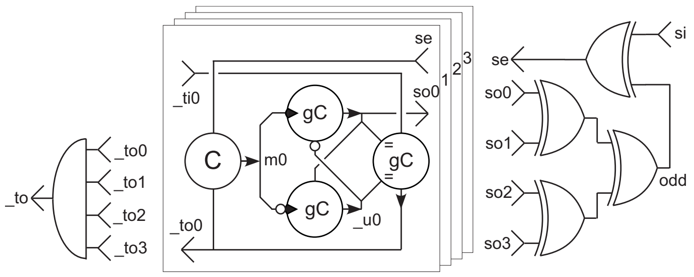

Fig. 5. Tog. Toggles _so0..3 when _ti0..3, respectively, is active; _to0..3 acknowledge requests and si enables toggles. _ti0..3 and m0 are reset high while _u0 is reset low, which forces so0..3 low.

```
*[[ti0]; to0+; [si=odd(so0..3)]; u0:=~so0;
 [~ti0]; to0-; [si=~odd(so0..3)]; so0:=u0
 ]
# Dec1 #
*[co0..3:=dec(si0..3,e); [ci]; e:=~so;
  co0..3:=dec(si0..3,e); [~ci]; so:=e
 ]
```

In this sequencing, Xbar is passive-active, Tog is passive-passive, and Dec1 is active-active (input-output). A passive input for Xbar follows convention while a passive output for Tog yields a full cycle of slack—no overlap between its T and S communications—`[si=odd(so0..3)]` is not part of S. This wait confirms that the previous cycle's toggle (`so0:=u0`) took effect, making it is safe to update `u0`. Note that it is correct to assume that `si` has not changed: It changes after Dec1 executes `so:=e`, which requires Tog to execute `to0-` (a 4-input OR-gate feeds `to0` to `ci`). This action occurs after `[si=odd(so0..3)]`.

The two muxes in each of Xbar's blocks (see Fig. 2) are implemented with two identical gC-elements (Fig. 4). The first one combines (C-elements) `di0` and `di1` with `cif` and `cit`, respectively, and ORs the results; collapsing these two logic-levels into a single level saved eight transistors. The second one swaps `di0` and `di1`. Wires suffice for the acknowledge.[3]

[3]This implementation is fully DI.

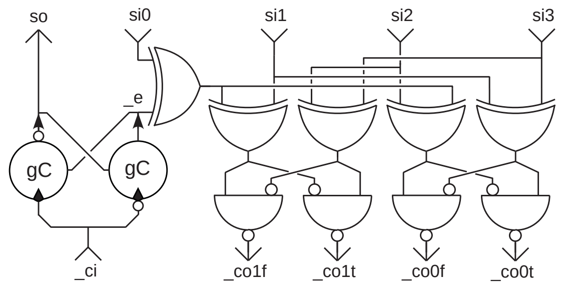

Fig. 6. Dec1. Decodes an odd word (si0..3) when _e is low and an even one when it is high. _e switches when _ci goes low, clearing the output (_co0t,_co0f,_co1t,_co1f), and so toggles when _ci goes back high, enabling the next input. _e is reset high.

Tog is implemented with four toggle elements, each implemented with two cross-coupled gC-elements, a C-element (triggers priming and toggling) and a third gC-element (confirms them) (Fig. 5).[4] The C-element ensures that a new output has been requested (`se` high) before triggering toggling (by taking `m0` high) and that this request has been serviced (`se` low) before triggering priming (by taking `m0` low)—otherwise two toggles could occur in response to the same request. When toggling occurs, `so0` complements `_u0`, enabling the third gC-element to acknowledge a new input. And when priming occurs, `_u0` matches `so0`, enabling the acknowledge to be withdrawn.[5]

Dec1's control is implemented with two cross-coupled gC-elements (similar to Tog) and its datapath (`dec()`) with four XORs and four NANDs (Fig. 6). Each NAND combines the outputs of two XORS—one extracts the encoded bit, the other checks that the parity is odd:

```
co0f := (si3= si1) & (si0p=~si2)
co0t := (si3=~si1) & (si0p= si2)
co1f := (si3= si2) & (si0p=~si1)
co1t := (si3=~si2) & (si0p= si1)
```

With `si0p := xor(_e,si0)`, the same logic decodes even codes—flipping the LSB converts them into odd codes (see Fig. 1). `_e`, which is high when an even code is requested, is used instead of `so` because `_e` switches immediately after the Dec1's output is acknowledged—thereby clearing the outputs—whereas `so` switches immediately after the acknowledge clears (see HSE). Reseting `_e` high decodes Tog's all-low outputs (even), thus Xbar receives valid data on startup.[6]

We now describe the implementation of Receiver's two blocks (see Fig. 3): Buf, a two-phase active-passive buffer (compatible with Sender's passive output port), and Dec2, a 1c4 decoder (identical to Sender's Dec1). Buf's HSE reshuf-

[4]Both inputs must be high to set and low to clear a C-element. Strobe input (dimple) must be high to set or clear a gC-element; equal (=) inputs must be equal to set and unequal to clear (XNOR); unmarked inputs function as in a C-element.

[5]Except for the isochronic fork `m0->_u0,so0`, which ensures that both gC-elements are not updated simultaneously, this implementation is DI.

[6]Except for the isochronic fork `_ci->_e,so`, which ensures that both gC-elements are not updated simultaneously, this implementation is DI.

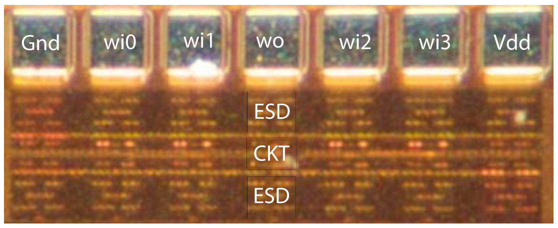

Fig. 7. Receiver Pads (271.4μm height; 100μm pitch). All logic circuitry (CKT) resides in the enable pad—taking up less than 75 × 13μm (0.18μm process)—sandwiched between electrostatic discharge devices (ESD).

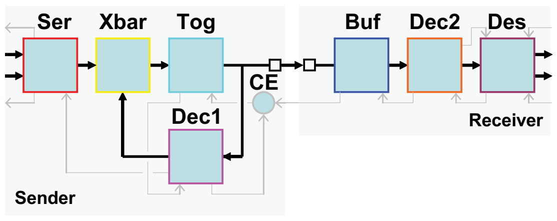

Fig. 8. Link plus Seriralizer and Deserializer. Dual-rail odd/even signals from Dec1 and Dec2 control Ser and Des, respectively.

fling allows Sender to send the next codeword while Receiver is decoding the previous one, thereby increasing throughput. Its design is similar to [5], except that there is no separate request and the acknowledge is generated by a parity circuit (XOR tree). Buf feeds Dec2, whose two active-low dual-rail outputs are converted to 1-of-4 by four OR gates.[7] Although Dec2's circuit is identical to Dec1's, it is reset differently: `_e` is reset low rather than high. As a result, Dec2's `so` goes high immediately, requesting an odd code from Sender (through Buf). The C-element combines this request with that from Dec1—which goes high once Xbar has received Sender's decoded even output. Thus, whereas Dec1 decodes Sender's even output upon reset, Dec2 ignores it.

With full-custom layout, Sender and Receiver's circuitry (0.18μm CMOS) were each small enough to fit in a single pad (Fig. 7). In addition to these circuits, we designed a 2-to-1 serializer and a 1-to-2 deserializer, controlled by phase signals from Dec1 and Dec2, respectively (Fig. 8). These phase signals were computed as follows:

```
codd  := and( odd(si0..3), e)
ceven := and(~odd(si0..3),~e)
```

The respective output goes high when an odd or even code is received, and goes low when the acknowledge (`ci`) causes `e` to switch (see Fig. 6). Dec1 and Dec2 obtain `odd(si0..3)` from Tog's and Buf's parity circuits, respectively, avoiding duplication. Thus, we transmitted two pairs of bits over four I/O pads—one pair on the odd phase, the other on the even phase—taking advantage of the doubled throughput 1c4 offers to cut pad-count in half.

[7]As the first DR input that clears will clear the 1-in-4 output, this implementation is not DI.

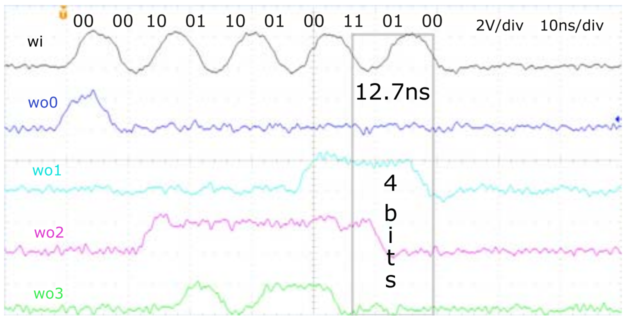

Fig. 9. Transmission rate. Two dataline transitions occur in 12.7ns, transmitting four bits, a 315 Mb/s data-rate. Bit values are indicated. Notice that the inverted request (wi) behaves just like an uninverted acknowledge.

IV. Test Results and Conclusions

Ours is the first full-custom VLSI implementation of 1c4. The chip, fabricated in 0.18μm CMOS, was tested with 3cm-long PCB traces connecting a pair of 1c4 sender and receiver pad-groups (on the same chip). Each two-phase communication took 6.3 ns, yielding 315 Mb/s (Fig 9). Power dissipation was 17.7mW at 197Mb/s, or 89.8 pJ/bit.[8] The two previous implementations were done with a CPLD [1] and with standard cells [3]. The latter's cycle-time was estimated to be 4.5-5.0 ns, based on simulation results, but this was for an on-chip link. Whereas our 6.3 ns cycle-time was measured from a fabricated chip, and is for a chip-to-chip link, which has much higher capacitance.

V. Acknowledgements

Shih-Chii Liu and Fopefolu Folowosele contributed to a CPLD implementation of this link at the 2007 Telluride Neuromorphic Engineering Workshop. Paul Merolla and John Arthur performed the test measurements. The NIH Director's Pioneer Award provided funding (DPI-OD000965).

References

[1] A. Chandrasekaran, F. Folowosele, S. Liu, and K. Boahen, "One-change-in-four encoders and decoders," in *2007 Telluride Neuromorphic Engineering Workshop Report*. Institute for Neuromorphic Engineering, 2007, pp. 37–47, http://ine-web.org/workshops/past-workshops.

[2] M. E. Dean, T. E. Williams, and D. L. Dill, "Efficient self-timing with level-encoded 2-phase dual-rail," in *Proc. of the 1991 UC Santa Cruz Conference on Advanced Research in VLSI*, C. H. Sequin, Ed. Cambridge MA: MIT Press, 1991, pp. 55–70.

[3] P. B. McGee, M. Y. Agyekum, M. A. Mohamed, and S. M. Nowick, "A level-encoding transition signaling protocol for high-throughput asynchronous global communication," in *14th IEEE International Symposium on Asynchronous Circuits and Systems*, 2008, pp. 116–127.

[4] A. Martin, *Programming in VLSI: From Communicating Processes to Delay-Insensitive Circuits*, ser. UT Year of Programming Series. Addison-Wesley, 1990, pp. 1–64, http://resolver.caltech.edu/CaltechCSTR:1989.cs-tr-89-01.

[5] M. Singh and S. M. Nowick, "Mousetrap: High-speed transition-signaling asynchronous pipelines," *IEEE Trans. VLSI*, vol. 15(6), pp. 684–698, 2007.

[8]Simulations including layout-extracted parasitics, bonding-wire inductances (3nH), package pin capacitance (0.5pF), and PCB transmission-lines (50 Ω,2 cm) predicted 4.8 ns per two-phase communication, yielding 414 Mb/s at 11.3mW, or 27.3pJ/bit.