A Level-Encoded Transition Signaling Protocol for High-Throughput Asynchronous Global Communication*

Peggy B. McGee, Melinda Y. Agyekum, Moustafa A. Mohamed and Steven M. Nowick
Department of Computer Science
Columbia University
New York, NY 10027
Email: {pmcgee, melinda, mmohamed, nowick}@cs.columbia.edu

Abstract

A new delay-insensitive data encoding scheme for global communication, level-encoded transition signaling (LETS), is introduced. LETS is a generalization of level-encoded dual rail (LEDR), an earlier non-return-to-zero encoding scheme where one of two wires changes value per data bit per transaction. In LETS, only one of $N = 2^n$ (1-of-N) wire changes value per n data bits per transaction. Compared to most common return-to-zero encoding schemes, LETS has potential power and throughput advantages, since fewer rails switch and no return-to-zero phase is required. Compared to existing non-return-to-zero schemes (i.e., LEDR), higher-dimension LETS codes have a potential power advantage, with significantly reduced switching activity per data bit.

Two alternative 1-of-4 LETS codes are proposed, and efficient hardware for completion detection and conversion to return-to-zero protocols is introduced. Finally, a general theoretical framework is presented which characterizes the properties of arbitrary 1-of-N LETS codes, as well as a simple procedure to generate such codes.

1. Introduction

Level-encoded dual-rail (LEDR) signaling [3] is a delay-insensitive data encoding scheme that encodes two wires, or "rails", to encode one bit of data. One rail is a data wire (rail 1), which holds the value of the bit in a standard single rail encoding, and the other rail is a parity wire (rail 0), which indicates phase by its parity relative to the data rail. LEDR uses a so-called *two-phase* or *non-return-to-zero (NRZ)* protocol, since no return-to-zero phase is required.

A distinguishing feature of LEDR is that the encoding strictly alternates between two phases: even and odd. The data value in each phase is always carried on rail 1. The parity value is always carried on rail 0; in an odd phase, the two rails have odd parity, while in an even phase they have even parity. The encoding of bit 1 is 10 in the odd phase and 11 in the even phase. The encoding of bit 0 is 01 in the odd phase and 00 in the even phase. For each new data that arrives, exactly one rail makes a transition, with no return-to-zero.

The LEDR protocol is *delay-insensitive* [14]: in the transition from one valid code (in one phase) to any valid code (in the opposite phase), it is impossible to pass transiently through another code. Hence, one can detect the completion of a data transmission unambiguously.[1]

The LEDR protocol has two potential benefits over return-to-zero (RZ) schemes for asynchronous global communication [7]: *throughput* and *power*. Unlike return-to-zero schemes, no 'spacer' or reset phase is required, hence LEDR provides a significant system-level throughput advantage. Furthermore, LEDR can provide a power advantage, since only one transition occurs on a rail per data bit transmission, while return-to-zero schemes require two transitions. These benefits have encouraged recent applications using LEDR encoding [4, 9].

The goal of this paper is to explore more efficient protocols for delay-insensitive communication. A new delay-insensitive data encoding scheme for global communication, called *level-encoded transition signaling (LETS)*, is introduced. LETS is a generalization of LEDR encoding. In LEDR, only one of two wires changes value per data bit per transaction. In contrast, in LETS, only one of $N = 2^n$ (1-of-N) wires changes value per n data bits per transaction. Hence, LEDR can be regarded as a special case: 1-of-2 LETS codes. Compared to existing non-return-to-zero schemes (LEDR), higher-dimension LETS codes have a potential power advantage, with significantly reduced switching activity per data bit. Compared to most common return-to-zero encoding schemes [1, 6, 15], LETS also has potential power and throughput advantages, since fewer rails switch per transaction and no return-to-zero phase is required.

Contributions. Two alternative practical 1-of-4 LETS codes are introduced, and their properties are defined. Efficient hardware is then presented to support 1-of-4 LETS codes, for completion detection and conversion between 1-of-4 LETS and four-phase dual-rail (i.e., return-to-zero) protocols. The converter is based on a recent efficient design for a converter between LEDR and four-phase dual-rail, which was carefully

*This work was partially supported by NSF ITR Award No. NSF-CCR-0086036 and by an Initiatives in Science and Engineering (ISE) grant from Columbia University (from the Office of the Executive Vice President for Research).

[1]Note that delay-insensitive codes usually refer to return-to-zero codes, where all rails are reset to zero between successive transmissions. However, the same concept naturally extends to non-return-to-zero codes, like LEDR, in which the completion of a data transition is unambiguous.

simulated after layout [7]. The new 1-of-4 LETS converter makes only small modifications to this existing design. Overall, this new design maintains similar performance, in terms of latency and cycle time, as the original converter design. Additionally, two alternative pipeline designs for maintaining acceptable throughput are presented. Finally, a general theoretical framework is defined which characterizes the properties of arbitrary 1-of-N LETS codes, as well as a simple procedure to generate such codes. An analytical comparison of the tradeoffs of LETS codes and existing approaches is also provided.

Several delay-insensitive 1-of-N encoding schemes have been proposed for *return-to-zero protocols*; these schemes are straightforward generalizations of dual-rail. However, LEDR codes are significantly more complex, with subtle phase and adjacency requirements on codewords. In particular, the contribution of the paper is the first 1-of-N scheme (with N>2) proposed for *level-encoded transition signaling*, which demonstrates that it is both possible and practical to encode two bits of data in four rails where only one rail makes a transition per data communication (i.e. 1-of-4), and that such a scheme is scalable to encode arbitrary higher numbers of bits (i.e. 1-of-N).

Related Work. There has been a body of work that has explored alternative encoding styles using a return-to-zero (RZ) protocol, but much less work on exploring variants for non-return-to-zero (NRZ) protocols.

In recent work on RZ encoding, the classic four-phase dual-rail protocol (1-of-2 RZ) has been extended to 1-of-4, 1-of-N, and m-of-N RZ codes [1, 6]. A key benefit is the potential for significantly lower transition power (or energy per transaction) for global communication. For example, a 1-of-4 RZ code encodes two bits on four rails, with only one rail changing value – twice – per communication. In contrast, using a classic four-phase dual-rail code, two bits are also encoded on four rails, but with two rails changing value – each twice – per communication. Hence, with the same coding efficiency (i.e., bits/wire), 1-of-4 RZ codes can have significantly lower switching activity for global data transmission.

Other general delay-insensitive encoding schemes using RZ protocols are surveyed in [14], but few of these have been shown amenable to efficient hardware implementation.

Novel variants of RZ codes have also been proposed, which effectively *overlap* the reset phase with the next *evaluate* phase, thus combining some of the throughput benefits of LEDR (i.e., two-phase) with the simplicity of RZ codes [12, 10]. However, in [10], coding efficiency is poor, with three rails per bit, and suffers from the same power overheads of other RZ schemes. In [12], the encoding scheme uses an RZ approach, but where codes depend on previous history.

Other encoding schemes for asynchronous communication include pulse mode logic [8, 5], bundled-delay signaling [13], and phase modulation signaling [2]. However, these are in general less robust than delay-insensitive encoding styles.

2. 1-of-4 LETS Codes

This section focuses on one class of LETS codes: 1-of-4. These codes encode $n = 2$ bits on $N = 2^n = 4$ rails, where exactly one of the four rails makes a single transition per data communication. As with LEDR, the codes alternate strictly between two phases of *even* and *odd codes*. 1-of-4 LETS codes have the same relationship to classic LEDR codes (i.e., 1-of-2 LETS), as 1-of-4 RZ codes have to dual-rail RZ codes.

This section first illustrates the simple derivation of a 1-of-4 LETS code. Some formal properties of 1-of-4 LETS codes are then presented, as well as an enumeration of the number of distinct codes that exist. Finally, two particular 1-of-4 LETS codes are then highlighted, and hardware support for one of these LETS codes is then presented.

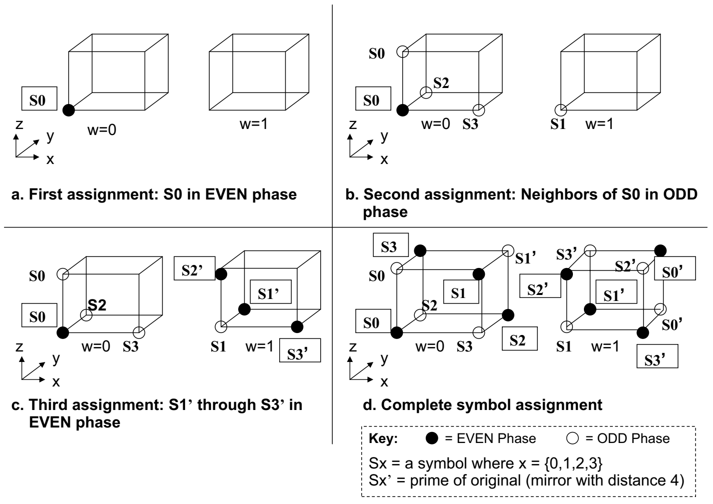

S0
w=0
w=1
a. First assignment: S0 in EVEN phase
S0
S2
S3
S1
b. Second assignment: Neighbors of S0 in ODD phase
S2'
S1'
S3'
c. Third assignment: S1' through S3' in EVEN phase
S0'
d. Complete symbol assignment
Key: ● = EVEN Phase ○ = ODD Phase
Sx = a symbol where x = {0,1,2,3}
Sx' = prime of original (mirror with distance 4)

Figure 1. Derivation of a 1-of-4 LETS Code

2.1. Deriving a 1-of-4 LETS Code: an Example

Figure 1 gives a simple stepwise derivation of a legal 1-of-4 LETS code. The figure illustrates the assignment of symbols to codes in the code space. Since 1-of-4 LETS codes encode 2 bits, there are $2^2 = 4$ symbols that must be assigned to codes: $S0$, $S1$, $S2$ and $S3$ (these correspond, for example, to respective binary data 00, 01, 10 and 11). In addition, since 4 rails will be used for the codewords, there are $2^4 = 16$ possible codewords or vertices, as shown in a 4-dimensional Boolean hypercube. Finally, codes are partitioned into even and odd phases, which must strictly alternate.

Figure 1a shows the initial unassigned code space. Arbitrarily, symbol $S0$ in the *even phase* can be assigned first to some codeword. Any codeword can be assigned this symbol; in the figure, it is arbitrarily assigned to the origin = 0000.[2]

The next step is shown in Figure 1b. Starting with the given code for symbol $S0$ in even phase, *all four symbols in the odd phase* must be immediately reachable by exactly 1 transition each. That is, symbols $S0$-$S3$ in the odd phase must be assigned to codewords at *Hamming distance 1* from the origin. There are four symbols to be assigned, and four neighboring vertices at Hamming distance 1; these codewords are all 1-hot. These odd symbols are arbitrarily assigned.

[2]The terms "odd" and "even" are used to define abstract phases, and they do not restrict the code assignment. For example, a codeword with an even number of one rails can be assigned to either "odd phase" or "even phase".

Figure 1c assigns new *even phase* symbols that are reachable from the odd symbols of Figure 1b. In particular, consider the odd symbol $S1$ as a starting point. From the codeword of this symbol, *four even symbols* must be reachable at Hamming distance one: for $S0$-$S3$. Even symbol $S0$ is already assigned to the origin and reachable with 1 wire transition; therefore, only even symbols $S1$-$S3$ must be assigned codewords. An arbitrary satisfying assignment is shown in Figure 1c. Figure 1d completes the code assignment. Interestingly, there are no degrees of freedom in assigning the remaining symbols. To see this, first, it is shown how an odd symbol $S0'$ (in Figure 1c) must be assigned to codeword $wxyz = 1110$. Currently, even symbol $S2'$ is adjacent to the existing $S0$ odd symbol (i.e., Hamming distance one). However, even symbols $S1'$ and $S3'$ are not adjacent to odd $S0$, so a new odd symbol $S0'$ must be assigned. This new odd symbol $S0'$ cannot be assigned to codewords $wxyz = 1011$; if it were, then the even symbol $S2'$ would have two of its neighbors being odd $S0/S0'$, but since it has only four neighbors, it would not immediately reach all four distinct symbols. Hence, odd symbol $S0'$ must be assigned to codeword $wxyz = 1110$. Next, starting with this new odd symbol $S0'$, it must in turn reach all 4 even symbols at distance one, so a new even $S0'$ must be assigned to codeword $wxyz = 1111$. Continuing this process, every remaining codeword is *deterministically* assigned. The result is a valid 1-of-4 LETS code.

The above assignment procedure highlights all the degrees of freedom in creating legal 1-of-4 LETS codes. The codes are "level-encoded" trivially: the target is a functional assignment of symbols to codewords in a hypercube, hence each codeword is assigned exactly one unique symbol in a designated phase (even or odd). Furthermore, interestingly, a *multicode* is generated: each symbol has 4 corresponding codes, 2 codes for even phase and 2 codes for odd phase.

2.2. Basic Properties and Rules

Several important properties hold for 1-of-4 LETS codes.

Property #1: Level Encoding. The 1-of-4 LETS code assignment problem can be formulated as a *functional mapping* from 4-dimensional codeword space (containing $2^4 = 16$ possible codewords) to symbols ($S0$-$S3$) and phases (even or odd). Because the mapping is functional, it is inherently a level-encoded assignment.

Property #2: Alternating Phases. By definition of 1-of-4 LETS, one further constraint must hold on the assignment: the code for each odd (even) symbol must reach codes for each even (odd) symbol at Hamming distance 1.

The definition of a Hamming (weight) class is as follows.

Definition. Given a code space of N-dimensions, codes in *Hamming (weight) class k* (where $0 \leq k \leq N$) are the set of all codes with exactly k one bits.

The following basic rules hold for 1-of-4 LETS codes.

Sandwich Rule #1: Distinct Phases. Given a codeword X (assigned to a symbol and phase), no adjacent codewords (i.e., Hamming distance 1) can be assigned to symbols of the same phase as X.

Since a codeword has (i) exactly 4 neighbors at Hamming distance 1, and (ii) must reach codewords for exactly 4 symbols of the opposite phase, none of its immediate neighbors neighbors can have the same phase.

Sandwich Rule #2: Distinct Symbols. Given a codeword X (assigned to a symbol and phase), no two adjacent codewords (i.e., at Hamming distance 1 from X) can be assigned to the same symbol.

A similar argument applies as for Sandwich Rule #1.

Mirror Code (1's Complement) Rule. Given a codeword X, assigned to a symbol Sx and a phase, the 1's complement (i.e., bitwise complemented) code $\bar{X}$ must also be assigned to symbol Sx and the same phase.

This mirror code rule can be seen in the 1-of-4 LETS code of Figure 1. A sketch of the proof is straightforward: starting from an arbitrary symbol (e.g. even $S0$) assigned to an arbitrary codeword (e.g. 0000), no even $S0$ code will appear at distance-2, and therefore one must appear at Hamming distance with all bits inverted.

Hamming Weight Class Rule. All codes of a given Hamming weight class will always be assigned to the same phase.

The derivation is straightforward from the above example. If some symbol (e.g. $S0$) is assigned to the origin (weight class 0), which is in the even phase, then all 1-hot codes (weight class 1) must be in the odd phase (since $S0$ needs four neighbors, and there are only four 1-hot codes of weight class 1 in total), all 2-hot codes (weight class 2) in the even phase, and so on. Initially, it might not appear obvious that all 2-hot codes must be in the even phase; however, consider the example shown in Figure 1c. After $S1$ has been assigned neighbors, three out of the six codewords in Hamming weight class 2 have been assigned even phase. According to the Mirror Code Rule, the other three codewords in Hamming weight class 2 must also be even, because these codewords are the complements of the three already assigned.

Hence the weight classes represent "slices" of the code space, where each slice is assigned to only odd or only even symbols.

2.3. Theoretical Result: Number of Legal 1-of-4 LETS Codes

Given that a legal 1-of-4 LETS code exists, as illustrated in Figure 1, it is now shown how many distinct legal codes exist. The derivation of Figure 1 provides a framework for generating any legal 1-of-4 LETS code. Each step makes binding choices on the assignment of symbols (and phases) to codewords.

In Figure 1(a), there appear to be 16 distinct assignments of an even $S0$ symbol. However, by the Mirror Code Rule above, there are in fact only 8 distinct choices: once the even $S0$ symbol is assigned to the $w = 0$ half space, its 1's complement codeword in the $w = 1$ half space is guaranteed to be assigned to the same symbol and phase. Figure 1(b) then allows arbitrary assignments of the 4 odd symbols, $S0$-$S3$, to the 1-hot codewords. That is, there are 4! legal assignments at this step. Figure 1(c) focuses on, without loss of generality, assigning even neighbors to odd symbol $S1$, and the 3 even symbols $S1$-$S3$ can be arbitrarily assigned to its immediate codeword neighbors. That is, there are $3!$ legal assignments at this step. Finally, Figure 1(d) has no degrees of freedom. In summary there are $8 \cdot 4! \cdot 3! = 1152$ possible legal 1-of-4 LETS code assignments.

2.4. Two Practical 1-of-4 LETS Codes

Given the large range of possible 1-of-4 LETS codes, it is useful to select practical ones which are most amenable to hardware implementation. Two such codes are now presented.

a. Quasi-1-Hot 1-of-4 LETS Codes

Rails (r2, r1, r0) – symbols: S0 – 111, S1 – 100, S2 – 010, S3 – 001
Sx' – Prime of original Sx (mirror with distance 4)

1-Hot ODD Codes

symbol	r3	r2 r1 r0
S0	1	0 0 0
S1	0	1 0 0
S2	0	0 1 0
S3	0	0 0 1

1-Cold ODD Codes

symbol	r3	r2 r1 r0
S0'	0	1 1 1
S1'	1	0 1 1
S2'	1	1 0 1
S3'	1	1 1 0

1-Cold EVEN Codes

symbol	r3	r2 r1 r0
S0	1	1 1 1
S1	0	0 1 1
S2	0	1 0 1
S3	0	1 1 0

1-Hot EVEN Codes

symbol	r3	r2 r1 r0
S0'	0	0 0 0
S1'	1	1 0 0
S2'	1	0 1 0
S3'	1	0 0 1

b. Quasi-Binary 1-of-4 LETS Codes

Rails (r1, r0) – symbols: S0 – 00, S1 – 01, S2 – 10, S3 – 11
Sx' – Prime of original Sx (mirror with distance 4)

ODD Sx Codes

symbol	r3 r2	r1 r0
S0	0 1	0 0
S1	0 0	0 1
S2	0 0	1 0
S3	0 1	1 1

ODD Sx' Codes

symbol	r3 r2	r1 r0
S0'	1 0	1 1
S1'	1 1	1 0
S2'	1 1	0 1
S3'	1 0	0 0

EVEN Sx Codes

symbol	r3 r2	r1 r0
S0	0 0	1 1
S1	0 1	1 0
S2	0 1	0 1
S3	0 0	0 0

EVEN Sx' Codes

symbol	r3 r2	r1 r0
S0'	1 1	0 0
S1'	1 0	0 1
S2'	1 0	1 0
S3'	1 1	1 1

Figure 2. Two Practical 1-of-4 LETS Codes

In Figure 2(a), a **quasi-1-hot 1-of-4 LETS code** is shown. Each symbol has 4 corresponding assigned codewords, 2 in each phase. By the Mirror Code Rule, the two codewords for each odd (even) symbol are bit inversions of each other. This code uses the lower rails in many (but not all) of the codewords as a 1-hot representation. The bit-inverted versions then become analogous 1-cold representations. The only exceptions are in in the assignment of the $S0/S0'$ symbols.

In Figure 2(b), a **quasi-binary 1-of-4 LETS code** is shown. The same properties of the Mirror Code Rule apply. Here, the two lower rails in all of the codewords serve as a binary representation of the data, in either non-inverted or inverted form.

2.5. Hardware Support

This paper focuses on LETS codes to be used for global communication, not for implementing function blocks. Hence, there are only two key hardware components that need to be defined for 1-of-4 LETS codes: (i) completion detectors, and (ii) converters to and from RZ codes, to support RZ function blocks.

2.5.1 LETS Completion Detectors

In RZ codes, a *dual-rail code* has an OR2 detector for 1 bit (i.e., 2 rails), while a *1-of-4 code* has an OR4 detector for 2 bits (i.e., 4 rails). These individual completion signals are then combined with a tree of C-elements.

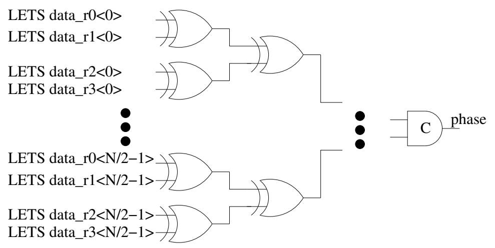


Figure 3. Completion Detector for 1-of-4 LETS Code

By analogy, with LETS codes, a *dual-rail code*, i.e., LEDR, has an XOR2 detector for 1 bit, while a *1-of-4 code* has an XOR4 detector for 2 bits. These individual completion signals are then combined with a tree of C-elements. A completion detector for 1-of-4 LETS codes is shown in Figure 3. Since the LETS completion detector uses combinational XOR2 gates in the first layer where sequential 2-input C-elements are used in the LEDR design, the LETS completion detectors are somewhat simpler than LEDR.

2.5.2 LETS Code Converters: To and from Dual-Rail RZ

A converter design is now outlined, converting between 1-of-4 LETS codes (for global communication) and dual-rail four-phase RZ codes (for implementing function blocks).

The design builds directly on a solution by Mitra et al. [7] for LEDR conversion. In this prior approach, a family of low-overhead converters was defined, converting between LEDR codes (for global communication) and RZ codes (for implementing function blocks). Several RZ function block styles were supported: dual-rail, 1-of-4, and bundled data. A full VLSI layout was completed, and detailed post-layout simulations performed, to validate the latency and throughput of the converter. It was demonstrated that the converters were efficient and had low area overhead.

In this section, the LEDR converter design is first reviewed, and then small modifications are proposed to handle 1-of-4 LETS interfaces. These modifications do not alter the concurrency of the protocol, and add little hardware to the critical paths. The focus of this design is to handle the *quasi-1-hot 1-of-4 LETS code.*

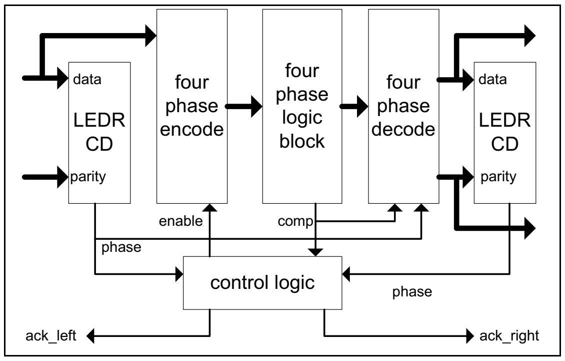


Figure 4. LEDR Protocol Converter: block diagram from [7]

Prior Work: LEDR Protocol Converters. The microarchitecture of an existing LEDR protocol converter is shown in Figure 4 (see [7]). LEDR encoding is used for global communication, while local computation is implemented using a four-phase dual-rail (i.e., RZ) protocol. The block accepts LEDR input data from the left environment, converts it to four-phase dual-rail data protocol and processes it in the logic block, then the results are converted back to LEDR and sent to the right environment. The key control signals are the *input phase*, which carries the parity of the most recent LEDR input data; the *output phase*, which carries the parity of the most recent wave of LEDR output data; the *enable*, which governs the evaluate and reset phases of the four-phase dual-rail logic block; and the logic block's completion signal, *comp*. Both *enable* and *comp* are level signals, making two transitions per computation; the *input phase* and *output phase* signals are transition signals, making one transition per computation.

The LEDR converter operates as follows. In a quiescent state, both *enable* and *comp* are low, indicating that the four-phase logic block is reset to NULL (i.e., all-0 outputs). The *input phase* and *output phase* have the same parity, indicating that the most recent LEDR input data has been fully processed and the results passed as LEDR output data. When new LEDR input data arrives, the left LEDR completion detector detects the valid input data, and transitions the *input phase* signal. The control block then asserts the *enable* signal; the LEDR inputs are then converted to four-phase dual-rail encoding, as shown in Figure 5a, and the logic block evaluates. When evaluation is complete, valid dual-rail data appears on the logic block's outputs. These outputs pass immediately to the four-phase decoding block, where each LEDR output bit can be set based solely on the value of its corresponding four-phase bit and the current input parity, as shown in Figure 5b (details in [7]). Eventually, the LEDR output completion detector will detect the new outputs, and transition the *output phase* signal; input and output phases are again matching.

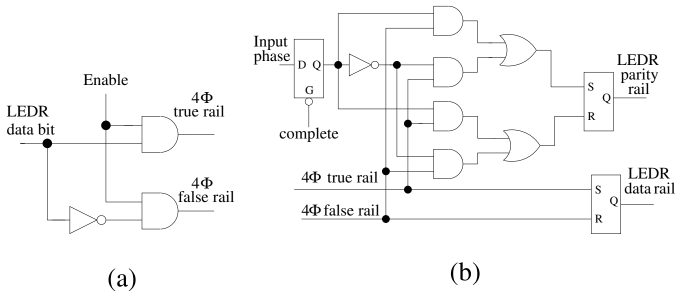


Figure 5. LEDR Protocol Converter Bit-Slice: (a) LEDR to 4-phase encoder (b) 4-phase to LEDR decoder from [7]

Concurrently, the logic block's dual-rail outputs pass through a completion detector which eventually asserts *comp*. Although the logic block is done evaluating, it will not be reset until the LEDR output completion detector makes a transition, since the four-phase output bits are needed to encode the LEDR output bits. Once the LEDR *output phase* transitions, the control block de-asserts *enable*, thus driving the logic block to its reset phase. When the block is fully reset, its completion signal *comp* is de-asserted, and the circuit returns to its initial state.

The converter allows a good degree of parallelism with its input environment. In particular, a left acknowledge is sent on as soon as results are computed and converted to LEDR on the right interface (i.e., after the falling edge of the *enable*). This event occurs after the completion of the logic block's evaluation phase, rather than the completion of the reset phase.

For applications where the node is used in a feed-forward pipeline (Figure 1, [7]), an additional synchronization point is added, to ensure that the node does not begin to process new input data (from the left neighbor) until the current result has been acknowledged (from the right neighbor). The current data is thus protected from overwriting until the right acknowledge is received.

A circuit layout was implemented in 0.18 micron TSMC process, with five metal layers and a 1.8 V supply, using standard cell gates. The design was not heavily optimized. The area for a 16-bit converter with no logic block (i.e., a FIFO stage, simply converting from LEDR to four-phase dual-rail back to LEDR) was quite small: only 0.018 mm. The performance of the converter was also quite reasonable: 2.7-3.1 ns latency and 3.9-4.2 ns minimum cycle time.

New Approach: 1-of-4 LETS Protocol Converters.

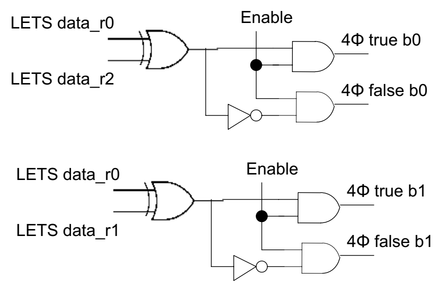
The 1-of-4 LETS protocol converter uses an identical micro-architecture as the previous LEDR converter of Figure 4, but with changes to only three different components: (i) the two LEDR completion detectors (on input and output data); (ii) the left LEDR to four-phase dual-rail encoder (Figure 5a); and (iii) the right four-phase dual-rail to LEDR decoder (Figure 5b).

(i) 1-of-4 LETS Completion Detectors. Each of the two LEDR completion detectors is replaced by a 1-of-4 LETS completion detector, shown in Figure 3.

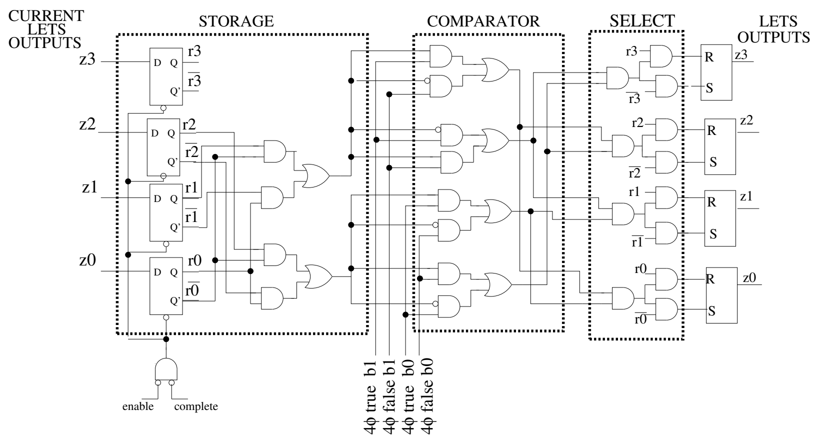
(ii) 1-of-4 LETS to Four-Phase Dual-Rail Encoder. The new encoder is shown in Figure 6a. It replaces the LEDR to four-phase dual-rail encoder of Figure 5a. The design is simple, exploiting the properties of the quasi-1-hot 1-of-4 LETS code. Unlike a transition-signaling to four-phase dual-rail translator, it is combinational, since a level-encoded input can be functionally (i.e., combinationally) translated.

(iii) Four-Phase Dual-Rail to 1-of-4 LETS Decoder. The new decoder is shown in Figure 6b. It replaces the LEDR to four-phase dual-rail encoder of Figure 5b. As with the earlier decoder, four-phase dual-rail inputs are received on the left, and a code is generated on the right through SR latches. In this case, however, two four-phase dual-rail bits (i.e., 4 rails), instead of one bit, is supplied on the left, and a two-bit 1-of-4 LETS code (i.e., 4 rails), instead of a one bit LEDR code (i.e., 2 rails), is generated on the right.

A key difference from the earlier LEDR decoder is that the LETS decoder is no longer combinational, but requires state. In particular, the 1-of-4 LETS code is a *multicode:* there are two distinct codewords for each symbol per phase (see Figures 1 and 2). The left encoder of Figure 6a is combinational, since the LETS code is 'level-encoded'. However, the right converter of Figure 6b must include sequential state (i.e., store the current LETS output code) to distinguish which of the two



(a)



(b)

Figure 6. LETS Protocol Converter Bit-Slice: (a) LETS to 4-phase dual-rail encoder (b) 4-phase dual-rail to 1-of-4 LETS decoder

multicodes to generate on new data inputs. Figure 6b shows how the LETS output code is fedback through transparent D-latches, to be used as state, to determine which of the two multicodes to generate, given the new four-phase dual-rail inputs. Also, unlike the circuit in Figure 5b, no explicit "input phase" control signal is used in the LETS converter. Instead, phase information is implicitly extracted from the stored current LETS output in the "Select" block to generate the new LETS output.

The decoder logic implementation also differs from that of Figure 5b. The new logic performs the following operations, from left to right: (a) converts the current 1-of-4 LETS code to a two-bit binary value; (b) compares the above value with a binary version of the new two-bit dual-rail input, and indicates in which bits (in binary) the two data items differ, if in any; and finally, (c) using this difference, resolves which of the two multicodes to generate for the new data, and activates the appropriate SR latch.

The new LETS decoder operates as follows. After a data item is processed and a LETS output is generated, the logic block resets. When fully reset, the *comp* signal makes the D-latches transparent, feeding the current LETS output to the decoder logic inputs. At this point, the four-phase dual-rail inputs are NULL, hence the SR latches are in the hold state. When *enable* is asserted high and the dual-rail logic block is activated to start evaluation, the D-latches are made opaque. Finally, when new four-phase dual-rail inputs arrive, the logic will activate the set or reset of exactly one SR latch, corresponding to the one bit which will flip in the current 1-of-4 LETS output code. The new 1-of-4 LETS code is thereby directly generated to the right environment.

Note that the "Storage" block component of Figure 8 is *not* on the critical path of operation. In particular, when the dual-rail logic block completes its reset, the current LETS output can immediately pass through the D-latches and logic of "Storage", to reach the Comparator inputs. The critical dual-rail to LETS output delay is therefore from the new dual-rail inputs (bottom of Figure 8) to the SR latch outputs.

The area and delay overhead of the combinational logic for this 1-of-4 LETS right decoder (which converts two bits) should not be significantly different per bit than that of the LEDR right decoder of Figure 5b (which converts only one bit). Beyond the logic, only 4 transparent D-latches are added, which are not on the critical path. Given the detailed breakdown of post-layout area, latency and throughput for the previous LEDR converter [7], the modifications to create a LETS converter are expected to have only small difference in area, and to have little effect on overall latency and throughput.

(iv) Summary. Because 1-of-4 LETS codes are *level-encoded*, they can be functionally translated by the left encoders – with only combinational logic – to any other RZ code. This is a clear advantage over transition-signaling codes, which require state to translate.

However, because the 1-of-4 LETS codes are *multicodes*, the right decoders must include state. Note, this disambiguation is a simpler problem than translating to transition-signaling codes: with 1-of-4 LETS codes, there are only two alternative multicodes in each phase to translate a given symbol. However, in transition-signaling, any codeword may be used for any symbol, depending on the previous history.

The hardware complexity presented in this section was derived using 1 of the 1152 possible legal 1-of-4 LETS code assignments *(see Section 2.3)*. The converter hardware implementation for other 1-of-4 codes requires only minor modifications to the circuits shown in Figures 6a and 6b. More specifically, for the encoder, the only modification needed is the replacement of the two XOR gates to left of the figure by some other 3-input combinational logic; likewise, modifications to the combinational logic in the "Storage" block of the decoder may also be necessary. Other than these, only minor wiring changes are needed in the "Comparator" and "Select" blocks of the decoder.

2.5.3 Performance Analysis

The performance analysis of the LETS protocol converter follows the same approach used for the earlier LEDR converters (Section 4, [7]). Three key metrics, *(i) latency, (ii) stabilization time,* and *(iii) minimum cycle time*, are derived and the results are compared to those of the LEDR design. Since a layout was performed on the LEDR implementation and accurate simulation results are available, this analytical comparison provides a rough estimate of the performance of the LETS design. For the LETS converter analysis, a simplified design was assumed without function blocks. In practical applications where function blocks are used, overheads of the LETS converter should be significantly reduced (c.f., LEDR analysis in [7]). More details on each metrics are summarized as follows.

(i) Latency. The latency of the protocol converter circuit is defined as the delay between the arrival of a full set of LETS inputs from the left environment to the circuit at quiescent state, to the output of a full set of LETS data from the circuit to the right environment. Referring to the micro-architecture diagram in Figure 4, the latency is the delay through the left completion detector, followed by the delay through the logic in the control logic block to generate the *enable* signal, and the delay through the four-phase encode, logic and decode blocks.

Both the LEDR and LETS protocol converters have the same blocks on the critical path, therefore the difference in their latency metrics is only due to the different implementation of the following two blocks: (a) *LETS completion detector:* replacement of a layer of 2-input C-elements by XOR2 gates; and (b) *four-phase decode:* two extra layers of AND2 gates. Note that even though the implementation of the encoder block is different for the LEDR and LETS designs (compare Figure 5a to 6a), the extra layer of XOR gates in the LETS design does not contribute to extra delays, as they are not on the critical path: outputs at the XOR gates are always expected to be available *before* the arrival of the enable signal at the AND gates under normal operating conditions.

From the extracted-layout simulations of the LEDR design, the latency (assuming no function blocks) was 2.7-3.1 ns.[3] The implementation differences in the LETS design is expected to add an overhead of roughly 15%-20% to these numbers.

(ii) Minimum Pipelined Cycle Time. The minimum pipelined cycle time is defined as the steady-state processing time per data item in a pipelined environment where the left environment acts as a continuous data source, and the right environment a continuous sink, assuming a set of *identical* stages. The minimum pipelined cycle time of the LETS protocol design is the same as that of the LEDR design, except for the difference in latency (as previously described), and the delays through the completion detection due to implementation differences.

In the extracted-layout using the LEDR converter design, the minimum pipelined cycle time was 3.9-4.2 ns. The extra logic in the LETS design is expected to impose a 15%-20% overhead on the cycle time (assuming no function blocks).

(iii) Stabilization Time. The stabilization time of the LETS protocol converter is defined as the delay between the arrival of a full set of LETS input from the left environment to the circuit in quiescent state, to the time when the circuit returns to its initial quiescent state. It includes the latency of the circuit, plus the delay through the output completion detector block, followed by the delay though the control logic block, the reset time of the four-phase function block, and the delay through the "Storage" block of the LETS decoder logic.

[3]The range is due to different data inputs for the simulation runs in the LEDR design.

Compared to the original LEDR design, the stabilization time of the LETS design differs in its latency (as described above), the delay through the output completion detector (due to the different internal implementation), and in an additional delay through the storage block of the LETS decoder logic.

In the extracted-layout using the LEDR converter design, the stabilization time was 5.9-6.7 ns. Given that the completion detectors have comparable performance, it is expected that the overall difference for the stabilization time, due to latency and "Storage" block overhead, is roughly 30% over the LEDR stabilization time (assuming no function blocks).

2.5.4 Timing Constraints

As in the LEDR design, the LETS design is nearly quasi-delay insensitive (QDI), but needs a few one-sided timing constraints to ensure its correct operation. When compared, the LETS design eliminates one of the two LEDR design timing constraints, but adds two easy-to-satisfy constraints.

The previous LEDR design observes two one-sided timing constraints: (i) a pulse-width requirement for the control signal of the SR-latch that is used to generate the *enable* signal (Figure 9, [7]); (ii) a setup and hold time requirement for the D-latches in the four-phase to LEDR decoder block (Figure 5). In the LETS design, (i) is still necessary since the LETS converter uses the same logic for generating the enable signal; however, (ii) is eliminated due to implementation differences in the LEDR and LETS converters. Although there are setup and hold time requirements for all SR-latches, these constraints can be avoided by replacing the SR-latches with C-elements.

The two new timing constraints for the LETS converter design are: (i) the control input at the D-latches used in the four-phase decoder (Figure 6b) must be long enough to latch the input data properly, and must end before the new four-phase input arrives; (ii) in the four-phase decoder (Figure 6b), by the time the new four-phase input arrives, the LETS output from the previous iteration must have propagated through the logic in the storage stage and arrived at the comparator stage.

2.6. Improving Throughput: Pipelining LETS Channels

The focus of this paper is to use LETS codes for robust point-to-point global communication. However, when communicating over long wires, one problem that may arise is maintaining acceptable throughput. Pipelining can be introduced to address this problem. Two LETS pipeline designs for sustaining throughput are presented; the first uses a simple one-sided timing constraint, while the latter is more robust (QDI).

The design in Figure 7 is based on the MOUSETRAP pipelines in [11]. The structure of the LETS pipeline design is the same as in the original MOUSETRAP design, except that the bundled data path and bundled delay are now replaced with the 1-of-4 LETS data path and completion detectors, respectively. The LETS design is more timing-robust than MOUSETRAP; it eliminates that latter's one-sided bundling constraint, but still retains the one-sided *data overrun* constraint (stage N must close its latch before N-1 sends new data).

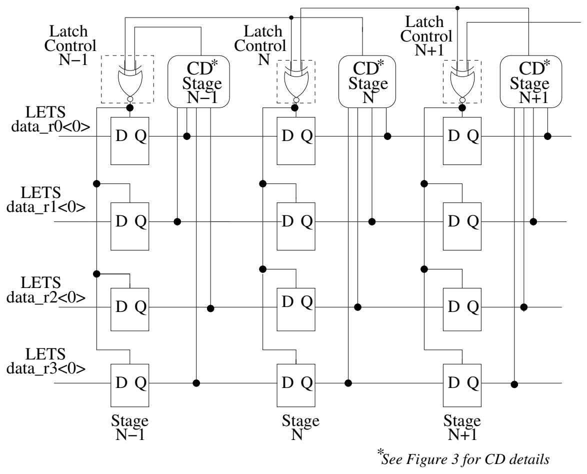


Figure 7. LETS-Style MOUSETRAP Pipeline Structure

Alternatively, a more timing-robust LETS pipeline can be used, which is based on Dean's LEDR-style pipelines (see [3], Fig. 11). As in the LEDR-style design, special customized identity function latches are created to selectively store even and odd phase data, which allow "self-latching" of data. The LEDR-style latches can be modified to LETS-style structures (designs omitted due to space limitations).

3. 1-of-N LETS Codes

This section presents a general framework for encoding n bits of data in $N = 2^n$ rails in LETS, i.e., 1-of-N LETS. This framework serves two purposes. First, it allows for the characterization of the entire class of LETS codes – including LEDR and any N beyond $N = 4$ – in terms of its code space and general properties. Second, it provides a formal method for constructing LETS codes based on fundamental properties of level encoding and transition signaling. Both the properties and the method can be used as a basis for exploring other codes.

Three key results are presented: (i) a characterization of the minimum requirements, in terms of the number of rails and the size of code space, required to encode n bits of data (Section 3.1); (ii) a procedure for generating legal 1-of-N LETS codes for any given N (Sections 3.2-1 and 3.2-2); and finally, (iii) a proof of the correctness of the procedure (Section 3.2-3).

3.1. Basic Properties

Some basic properties and theorems on 1-of-N LETS codes are presented in this section.

Informally, a LETS code is one that satisfies two properties: *(A) Level encoding:* datum value can be extracted directly from a codeword; and *(B) Transition signaling:* there is no return-to-zero phase, or equivalently, each symbol can reach all symbols by a single transition on exactly one rail, and can likewise be reached by all symbols.

This is more formally stated as follows.

Definition 1 (*1-of-N LETS code) A 1-of-N LETS code is triple (S, C, h) where S is a set of symbols, $C \subseteq \{0,1\}^N$ a set of codes and $h : C \rightarrow S$ a function which maps a code $c \in C$ to a symbol $s \in S$, such that the following two properties are satisfied:*

A. Level encoding: h maps each code $c \in C$ to at most one unique symbol $s \in S$,

B. Transition signaling: if c is assigned a symbol, then for each symbol $s_j \in S$, there exists exactly one code of Hamming distance 1 from c which maps to s_j.

The 1-of-N LETS code assignment problem can now be formally defined.

Definition 2 (*1-of-N LETS code assignment problem) Given n bits of data to encode on N wires, the LETS code assignment problem is to define a set of mappings $h : C \rightarrow S$, where $C \subseteq \{0,1\}^N$ is a set of codes and $S = \{s_1, s_2, ..., s_{2^n}\}$ a set of symbols representing n bits of data, such that both the level-encoding and transition-signaling properties of LETS in Definition 1 are satisfied.*

Theorems 1 and 2 show the minimum requirements for the number of rails and the size of the code space needed to encode n bits of data in $LETS$, respectively.

Theorem 1 (*Non-existence of solution for $N < 2^n$) Given n bits of data to encode on N rails, there does not exist a LETS code for any $N < 2^n$.*

Proof: For any LETS code, a symbol must be able to reach all symbols by flipping a distinct bit of a codeword that is assigned to it (transition signaling property of LETS). If fewer than N rails are used to encode a symbol, then the symbol must have fewer than N adjacent neighbors of Hamming distance one, therefore it cannot reach all symbols. It follows that there does not exist a LETS code to encode n bits of data using $N < 2^n$ rails. □

Theorem 2 (*Complete code assignment) If there exists a LETS code to encode n bits of data in $N = 2^n$ rails, the entire code space $C = \{0,1\}^N$ must be assigned.*

Proof: This is a proof by induction.

Base case: Suppose an all-zero code $\{0\}^N$ (Hamming class zero) is assigned a symbol. In order for this code to reach all N symbols (to satisfy the transition-signaling property of LETS), all N codes of Hamming class one must be assigned unique symbols.

Hypothesis: Assume all codes in Hamming class m were assigned symbols, then all codes in Hamming class $m+1$ must also be assigned symbols.

Induction: Every $m+2$ codes in Hamming class $m+1$ form a group which reaches a unique code in Hamming class $m+2$. For example, suppose $m = 1$, then the codes 0011, 0101 and 0110 from Hamming weight class two form a group with a common neighbor 0111 in Hamming weight class three. There are $\binom{N}{m+2}$ ways to form such groups in Hamming weight class $m+1$, if all codes in this class were assigned; therefore, $\binom{N}{m+2}$ codes in Hamming class $m+2$ must be assigned symbols. However, in total, there are only a maximum number of $\binom{N}{m+2}$ possible codes in Hamming class $m+2$. As a result, all codes in Hamming class $m+2$ must be assigned.

This proof assumes the assignment begins with a codeword in Hamming class zero. However, it also applies if any arbitrary codeword is chosen, as the code space is isomorphic up to the re-labeling of symbols. □

Interestingly, the complete code assignment property of Theorem 2 implies the multi-code property of higher-dimensional LETS codes, as in 1-of-4 LETS.

3.2. Generating 1-of-N LETS Codes

Having shown that there does not exist any 1-of-N LETS code to encode any n bits of data for $N < 2^n$, this section introduces a general procedure for deriving arbitrary LETS codes for $N = 2^n$. More specifically, a solution to the *1-of-N LETS code assignment problem* is presented.

3.2.1. Overview of Approach

The strategy to solve the LETS code assignment problem follows three main steps. First, a data structure, called the *bit-flip matrix*, which defines a mapping between symbols, is introduced. Second, it is shown that the bit-flip matrix can be used to generate a consistent code. Finally, it is shown that any code it generates is also *always* a LETS code.

More concretely, the bit-flip matrix is an $N \times N$ matrix where each row and column represent a symbol in a 1-of-N code, and each matrix entry represents the bit position which differs between a codeword assigned to the symbol in the corresponding row, and that in the corresponding column. The bit-flip matrix can be used as a code generator: when given a symbol which has already been assigned a codeword, the matrix can be used as a lookup table to assign further codewords to symbols, by indicating which bit to flip in order to get to the new symbol.

The rest of this section is organized as follows. Subsection 3.2.2 presents a formal definition of the bit-flip matrix problem. It also shows that a solution to the problem always exists, and a procedure to derive the solution is introduced. Subsection 3.2.3 presents a simple and direct procedure to generate a code from a bit-flip matrix. In Subsection 3.2.4, the relationship between the bit-flip matrix problem and the 1-of-N LETS code assignment problem is formally established. More specifically, it is shown that a legal bit-flip matrix always generates a legal 1-of-N LETS code.

3.2.2 The bit-flip matrix problem

The bit-flip matrix constraint-satisfaction problem is defined as follows.

Definition 3 (*The bit-flip matrix problem) Given an $N \times N$ matrix, assign integers $0...N-1$ to the matrix such that the following three properties are satisfied:*

1. *For each row, all row entries must be distinct.*
2. *For each column, all column entries must be distinct.*
3. *Given column i and two entries in the column, $m_{ki} = a$ and $m_{li} = b$, there must exist another column, j, such that $m_{kj} = b$ and $m_{lj} = a$.*

Figure 8 shows an example bit-flip matrix, $\mathbf{A}$, for $N = 8$. A quick inspection verifies that this matrix satisfies all three constraints for the bit-flip matrix problem, namely, all rows and columns have distinct entries, and for every pair of entries in a column, there is always another column with the same pair in the corresponding row positions, in reverse order.

Intuitively, the bit-flip matrix can be understood as follows. The rows and columns of the matrix represent a set of symbols $s_0...s_{N-1}$, and the matrix entries $m_{ij} = \{0...N-1\}$ a set of operations for mapping s_i to s_j. Given a set of bit vectors, they can be partitioned into N non-overlapping sets, each associated with a single symbol. In this way, the operation defined by each matrix entry m_{ij} can be considered a “bit flip” function, determining the bit position to flip in a bit vector belonging to symbol class s_i, in order to map it to a bit vector belonging to symbol class s_j.

	s_0	s_1	s_2	s_3	s_4	s_5	s_6	s_7
s_0	0	1	2	3	4	5	6	7
s_1	1	0	3	2	5	4	7	6
s_2	2	3	0	1	6	7	4	5
s_3	3	2	1	0	7	6	5	4
s_4	4	5	6	7	0	1	2	3
s_5	5	4	7	6	1	0	3	2
s_6	6	7	4	5	2	3	0	1
s_7	7	6	5	4	3	2	1	0

Figure 8. A Bit-Flip Matrix

Having formally defined the bit-flip matrix problem, it is now shown that there always exists a matrix which satisfies all the constraints defined in Definition 3 for all $N = 2^n$, and that there is a mechanical procedure to construct such a matrix.

Theorem 3 (*Existence of legal bit-flip matrices) There always exists a solution to the bit-flip matrix problem for any $N = 2^n$.*

Instead of a formal proof, a general method which can be used to generate a legal bit-flip matrix for any $N = 2^n$ is sketched below. Since such a general method exists, it indirectly proves that there always exists a solution to the bit-flip matrix problem for $N = 2^n$.

Consider again the matrix shown in Figure 8. Notice that the upper left quadrant (rows and columns 1 to 4) is a legal bit-flip matrix for $N = 4$. Further notice that the upper left corner with rows and columns 1 to 2 is also a legal matrix for $N = 2$. It is also not difficult to see that all four quadrants of the $N = 8$ matrix have similar properties. It is obvious then, given a legal matrix for $N = 2^k$ code, one can construct a legal $N = 2^{k+1}$ matrix by repeating this recursive structure. Moreover, a systematic re-labeling of the matrix entries will also always result in a legal matrix. It should be noted that not all legal matrices can be constructed by this method. However, by following this method, one can always generate a legal matrix.

3.2.3 Generating a consistent code from a bit-flip matrix

It is now shown how any legal bit-flip matrix satisfying all rules in Definition 3 of the bit-flip matrix problem can be used to generate a consistent code, i.e., one in which every codeword is assigned to no more than one symbol.

Let the row and column indices of a bit-flip matrix, $\mathbf{A}$, represent symbols in a code, and each matrix entry $m(i,j)$ a bit-flip function which maps the symbol in row i, s_i, to that in

```
1    GEN_CODE($A, c_0, s_0$)
2    $c'_0$ = complement($c_0$)
3    GEN_NEXT($A, c_0, s_0, c'_0$)
4    return $h$

5    GEN_NEXT($A, c, s, c'_0$)
6    if ($c = c'_0$)
7        return $h$
8    else
9        for $j = 0$ to 1 do
10           $c^+ \leftarrow$ bit_invert($A[s][j], c$)
11               $h(c^+) \leftarrow s_j$
12           GEN_NEXT($A^T, c^+, h(c^+), c'_0$)
```

Figure 9. Pseudocode: generating a code from the bit-flip matrix

column j, s_j, by flipping the bit at position $m(i,j)$ of a codeword assigned to s_i, then given a codeword c that has already been assigned to symbol s_i, the bit-flip matrix can be used as a look up table to assign symbols to the set of N codes which are of Hamming distance 1 from c. Each of these N new codeword/symbol pairs (c_i, s_i) can then be used to generate N new codeword/symbol pairs each using the transpose of matrix A, i.e., A^T, as a look up table.

Starting with an arbitrary "seed assignment" of a codeword of length N, $c_0 = \{0,1\}^N$, to a symbol s_0, and repeating the procedure described above, an $N \times N$ bit-flip matrix $\mathbf{A}$ (and its transpose, $\mathbf{A^T}$) can be used to assign symbols to the entire code space $C = \{0,1\}^N$.

The pseudo-code for this procedure is shown in Figure 9. The procedure accepts as parameters a bit-flip matrix $\mathbf{A}$ and a seed code/symbol assignment (c_0, s_0), and returns a mapping function h which maps each code $c \in \{0,1\}^N$ to a symbol $s \in \{s_1, s_2, ..., s_N\}$. The procedure GEN_CODE first computes the bitwise complement of c_0 (line 2), which is used as terminator for the recursive procedure GEN_NEXT, which it calls in line 3. Given a codeword/symbol pair, GEN_NEXT assigns N next codeword/symbol pairs by using the matrix $\mathbf{A}$ (and its transpose, $\mathbf{A^T}$) as a lookup table, and recursively calls itself. The recursive call terminates when the bitwise complement of the seed code is reached (line 6). At this point, the mapping function h is returned.

It is now proved that a code generated by this method is always *consistent*.

The entries in a bit-flip matrix can be viewed as 'bit-flip' functions, defined as follows.

Definition 4 (*Bit-flip function) Given a set of codes C and a set of symbols S, and a pair $(c \in C, s \in S)$, where c is a codeword assigned to symbol s, i.e., $h(c) = s$, the bit flip function $f_j : C \times S \to C \times S$ maps the pair (c, s) to (c', s'), where c' is obtained from c by flipping its jth bit, and s' any symbol in S.*

In order for the bit flip function to always generate a consistent code, it must have the following properties:

Property 3 (*Consistency properties of bit flips) A bit flip function must have the following properties in order to generate a consistent code:*

A. Symmetry: *if $f_i(c,s) = (c', s')$, then $f_i(c', s') = (c, s)$*
Flipping the same bit of a codeword assigned to a symbol twice always results in the symbol itself.

B. Commutativity: $f_i f_j(c,s) = f_j f_i(c,s)$
From a codeword assigned to a symbol, flipping two bits in succession always lead to the same symbol, regardless of the order in which the bits are flipped.

It is now shown that any code generated by a legal bit-flip matrix indeed satisfies Properties 3A and 3B.

Theorem 4 (*Consistency of codes) Any code generated by a legal bit-flip matrix is always consistent, i.e., any codeword is assigned to at most one symbol.*

Proof: Since a matrix $\mathbf{A}$ and its transpose $\mathbf{A^T}$ are used to generate codes of opposite phases, the symmetry property of bit-flips is satisfied. Constraint 3 of the bit-flip matrix, in addition, ensures that any code it generates satisfy the commutativity property. Since the algorithm shown in Figure 9 starts the code assignment process with a code that is already consistent (i.e., a code with a single member, the seed code/symbol pair), and the symmetry and commutativity properties of bit-flips *preserve* the consistency (i.e., level-encoding) property of a code that is already consistent, the code generated from a legal bit-flip matrix must also be consistent. $\square$

3.2.4 Existence and generation of LETS codes

Having shown how a legal bit-flip matrix is constructed, and how it can be used to generate a consistent code, it is proved in this section that this method can be used to *generate a LETS code*. More specifically, it is shown in Theorem 5 that a legal bit-flip matrix can be used to generate a LETS code, and furthermore, in Theorem 6 that *any* code generated by this method is always a legal LETS code.

It has been shown in Property 3 that in order for the bit-flip function to generate a consistent code, it must process symmetry and commutativity properties. It is now shown in Property 4 that in order for the bit-flip function to generate a LETS code, it must possess two additional properties:

Property 4 (*Transition signaling properties of bit-flips) To satisfy its transition signaling properties, the bit-flip function of a LETS code must have the following properties:*

A. $i \neq j \Rightarrow f_i(c,s) \neq f_j(c,s)$
Flipping different bits of a codeword always results in codewords that are assigned to different symbols.

B. for each $s \in S$, $s = f(c, s')$ for any $s' \in S$
Every symbol can be reached by another symbol with a single bit flip.

A formal relationship between the constraints in the bit-flip matrix problem (Definition 3) with the LETS properties (Definition 1) can now be established. More formally, it is shown in Lemma 1 that the matrix constraints imply the LETS properties. This Lemma will be used to prove Theorems 5 and 6.

Lemma 1 *The constraints for the bit-flip matrix problem (Definition 3) implies the LETS properties (Definition 1), i.e., any code that satisfies the constraints imposed by the code generator matrix must also satisfy the LETS properties.*

Proof: Matrix Constraint 1 (distinct row entries) implies Properties 4A and 4B (transition signaling) of the bit flip function. Matrix Constraints 1 and 2 (distinct column row entries) together further implies Property 3A (symmetry) of the bit flip function, which requires that the bit-flip matrix for opposite phases (odd and even) of a code be *transposes* of each other. Finally, matrix Constraint 3 (reversal of column pairs) implies Property 3B (commutativity) of the bit flip function. Together, the set of constraints for the matrix constraint satisfaction problem implies the bit-flip function, which in turn guarantees that the LETS properties in Definition 1 are satisfied. □

Theorems 5 and 6 can now be directly proven.

Theorem 5 (*Existence of a 1-of-N LETS code) A code generated by a legal bit-flip matrix is always consistent (i.e., any codeword is assigned only one symbol), and that it is also a LETS code.*

Proof: The consistency of any code generated by a bit-flip matrix has already been proven in Theorem 4. This implies that the code satisfies the level-encoding property of LETS (Definition 1). Constraints 1 and 2 of the bit-flip matrix further ensure that Properties 4A and 4B of the bit-flips are satisfied. Therefore the transition signaling property of LETS is also satisfied. It follows that the code is also a LETS code. □

Theorem 6 (*Safety of LETS code generation method) All codes generated by a legal bit-flip matrix are legal LETS codes.*

Proof: Since all codes generated by a legal bit-flip matrix must satisfy all of constraints 1 to 3 of the bit-flip matrix problem (Definition 3), then by Lemma 1 and Theorem 4, they must all be legal LETS codes.

3.2.5 Summary

This section presented a method for generating arbitrary 1-of-N LETS codes based on the properties of level encoding and transition signaling. It is proved that this method is *safe*, i.e., all codes generated by the method are LETS codes. It is conjectured that this method is also *complete*, i.e., all LETS codes can be generated by this method, though a proof would require further investigation.

It should also be noted that the LETS converter designs presented in Figure 6 can be directly extended to handle 1-of-N codes for $N > 4$. The complexity of the hardware is expected to scale linearly with respect to N.

4. Analytical Evaluation

A comparison table of global communication coding schemes is presented in Table 1. The columns are divided into two parts. In the left-most part, the "Encoding Schemes" column lists the general categories and names of several protocols used for asynchronous global communication; these protocols were then evaluated using the metrics listed in the right-most part of the table.

For the evaluation metrics, the "Throughput" column, indicates if a 4-phase (RZ) protocol or a 2-phase (NRZ) protocol is used. In the next two columns, "Coding Efficiency" and "Transition Power Metric" are reported based on the number of bits per wire and the number of transitions per bit (bit-flips) in a single transaction, respectively. The "Latency Overhead Metric" column gives the implementation of the completion detectors, from which one can infer the overhead on latency. The final column, "Timing Constraints", lists details of the timing robustness for each protocol.

The rows of the table are divided into three parts. The first part, "Return-to-Zero", contains four classes of codes which include 1-of-2 Dual Rail, 1-of-4, 1-of-8, 1-of-N, and M-of-N. Similarly, in the second part, 1-of-2, 1-of-4, 1-of-8, and 1-of-N are analyzed for the LETS encoding protocol. The last part, "Miscellaneous", shows details of the three encoding styles for transition signaling, bundled data,[4] and pulse mode.

Based on this table, LETS shows potential benefits in transition power and coding efficiency when compared to other schemes. More specifically, the table indicates that 1-of-4 LETS has coding efficiency comparable to that of 1-of-2 RZ, 1-of-4 RZ, and LEDR, and at the same time, yields maximal throughput since it does not use a RZ phase. In addition, it has significant transition power benefits over 1-of-2/1-of-4 RZ and LEDR. In Section 2.5, it has been shown that the converter hardware for 1-of-4 LETS requires only moderate changes to the LEDR converter hardware; therefore, there should be no significant performance penalties using a 1-of-4 LETS converter over an LEDR converter.

An interesting observation about 1-of-N LETS encoding is that, like 1-of-N RZ, the power begins to experience diminishing return for $N > 4$. For example, the transition power for a 1-of-4 LETS code is 50% of 1-of-2 LETS code, while a 1-of-8 LETS code uses 66% of the transition power of 1-of-4 LETS. This indicates that within the LETS codes, 1-of-4 appears to be an ideal coding design or "sweet spot" because of its increase in coding density without degradation in other metrics.

Finally, when compared to the three miscellaneous encoding schemes, LETS appears to have comparable results to that of transition signaling, and shows clear advantages over pulse mode, which uses two transitions for each data transaction, and requires a complex completion detector, as two pulses, rather than two edges, need to be caught [5, 8]. The LETS protocol is also more robust than the bundled data scheme.

5. Conclusions and Future Work

A new general delay-insensitive data encoding scheme for global communication, called level-encoded transition signaling (LETS) is introduced. LETS is a generalization of an existing scheme, LEDR, and encompasses a wide family of codes. Two alternative practical 1-of-4 LETS codes are proposed, along with efficient hardware for completion detection and conversion between 1-of-4 LETS and four-phase dual-rail protocols. A general theoretical framework is presented, which characterizes the properties of arbitrary 1-of-N LETS codes, as well as a simple procedure to generate legal LETS codes. An analytical comparison of the tradeoffs between LETS codes and existing approaches is also provided.

[4]The "Transition Power Metric" for bundled data is calculated based on the assumption that data is random, i.e., the probability of a bit flipping on a wire per data transaction is 0.5.

Encoding Scheme		**Throughput Metric** ***(RZ vs. NRZ)***	**Coding Efficiency** ***(# of bits/wire)***	**Transition Power Metric** ***(# of transitions/ bit/transaction)***	**Latency Overhead Metric** ***(due to Completion Detector)***	**Timing Constraints**
Return-to-Zero (RZ)	*1-of-2 (dual-rail)*	RZ	$\frac{1}{2}$	2	OR2/bit into n-input C-element	Delay insensitive
	1-of-4	RZ	$\frac{1}{2}$	1	OR4/bit into $\left\lceil \frac{N}{2} \right\rceil$-input C-element	Delay insensitive
	1-of-8	RZ	$\frac{3}{8}$	$\frac{2}{3}$	OR8/bit into $\left\lceil \frac{N}{3} \right\rceil$-input C-element	Delay insensitive
	1-of-N	RZ	$\frac{\lceil \log N \rceil}{N}$	$\frac{2}{\lceil \log N \rceil}$	ORn/bit into $\left\lceil \frac{N}{\lceil \log N \rceil} \right\rceil$-input C-element	Delay insensitive
	M-of-N	RZ	$\frac{\left\lceil \log \binom{n}{m} \right\rceil}{N}$	$\frac{2}{\left\lceil \log \binom{n}{m} \right\rceil}$	M-of-N majority function into C-element	Delay insensitive
Level-Encoded Transition Signaling (LETS)	*1-of-2 (LEDR)*	NRZ	$\frac{1}{2}$	1	XOR2/bit into n-input C-element	Delay insensitive
	1-of-4	NRZ	$\frac{1}{2}$	$\frac{1}{2}$	OR4/bit into $\left\lceil \frac{N}{2} \right\rceil$-input C-element	Delay insensitive
	1-of-8	NRZ	$\frac{3}{8}$	$\frac{1}{3}$	XOR8/bit into $\left\lceil \frac{N}{3} \right\rceil$-input C-element	Delay insensitive
	1-of-N	NRZ	$\frac{\lceil \log N \rceil}{N}$	$\frac{1}{\lceil \log N \rceil}$	XORn/bit into $\left\lceil \frac{N}{\lceil \log N \rceil} \right\rceil$-input C-element	Delay insensitive
Miscellaneous	*Transition signaling*	NRZ	$\frac{1}{2}$	1	XOR2/bit into n-input C-element	Delay insensitive
	Bundled data	RZ or NRZ	1	$\frac{1}{2}$	Wire with delay	Bundled path delay
	Pulse-mode	RZ	$\frac{1}{2}$	2	Complex ([5, 8])	Pulse-width timing

Table 1. Comparison of Asynchronous Global Communication Coding Schemes

LETS has potential throughput and power advantages over many delay-insensitive schemes, since only one rail switches per data transaction, and no return-to-zero phase is required. It was shown that LETS is practical for implementation.

For future work, a layout of the proposed circuits and post-layout simulation will be necessary to better evaluate area overhead and performance, especially in terms of throughput and power consumption, as well as the evaulation of the effects of cross-coupled capacitance, which may also have an impact on power. Novel designs for functional blocks implemented to directly use LETS data without the need for conversion can also be explored. Finally, as in [7], modified LETS converters can be derived that support other common RZ function blocks (e.g., bundled data and 1-of-4).

References

[1] W. J. Bainbridge, W. B. Toms, D. A. Edwards, and S. B. Furber. Delay-insensitive, point-to-point interconnect using M-of-N codes. In *Proc. of the 9th International Symposium on Asynchronous Circuits and Systems*, pages 132–141, 2003.

[2] C. D'Alessandro, D. Shang, A. Bystrov, A. Yakovlev, and O. Maevsky. Multiple-rail phase-encoding for noc. In *Proc. of the 12th IEEE International Symposium on Asynchronous Circuits and Systems*, pages 107–116, 2006.

[3] M. E. Dean, T. E. Williams, and D. L. Dill. Efficient self-timing with level-encoded 2-phase dual-rail (LEDR). In *Proc. of the 1991 University of California/Santa Cruz Conference on Advanced Research in VLSI*, pages 55–70, 1991.

[4] R. R. Dobkin, R. Ginosar, and A. Kolodny. High rate wave-pipelined asynchronous on-chip bit-serial data link. In *Proc. of the 13th IEEE International Symposium on Asynchronous Circuits and Systems*, pages 3–14, 2007.

[5] J. Ebergen, S. Furber, and A. Saifhashemi. Notes on pulse signaling. In *Proc. of the 13th IEEE International Symposium on Asynchronous Circuits and Systems*, pages 15–24, 2007.

[6] A. J. Martin. Compiling communicating processes int delay-insensitive VLSI circuits. *Distributed Computing*, 1:226–234, 1986.

[7] A. Mitra, W. F. McLaughlin, and S. M. Nowick. Efficient asynchronous protocol converters for two-phase delay-insensitive global communication. In *Proc. of the 13th IEEE International Symposium on Asynchronous Circuits and Systems*, pages 186–185, 2007.

[8] L. A. Plana and S. H. Unger. Pulse-mode macromodular systems. In *Proc. of the International Conference on Computer Design*, pages 348–353, October 1998.

[9] B. R. Quinton, M. R. Greenstreet, and S. J. E. Wilton. Asynchronous ic interconnect network design and implementation using a standard asic flow. In *Proc. of the 2005 International Conference on Computer Design (ICCD 2005)*, pages 267–274, 2005.

[10] P. T. Røine. A system for asynchronous high-speed chip to chip communication. In *Proc. of the 2nd International Symposium on Advanced Research in Asynchronous Circuits and Systems*, pages 2–11, 1996.

[11] M. Singh and S. M. Nowick. MOUSETRAP: High-speed transition-signaling asynchronous pipelines. *IEEE Trans. Very Large Scale Integr. Syst.*, 15(6):684–698, 2007.

[12] C. Svensson and J. Yuan. A 3-level asynchronous protocol for a differential two-wire communication link. *IEEE Journal of Solid-State Circuits*, 29(9):1129–1132, September 1994.

[13] H. van Gageldonk, K. van Berkel, A. Peeters, D. Baumann, D. Gloor, and G. Stegmann. An asynchronous low-power 80C51 microcontroller. In *Proc. of the 4th IEEE International Symposium on Asynchronous Circuits and Systems*, pages 96–107, 1998.

[14] T. Verhoeff. Delay-insensitive codes—an overview. *Distributed Computing*, 3(1):1–8, 1988.

[15] T. E. Williams. *Self-Timed Rings and Their Applications to Division*. PhD thesis, Stanford University, June 1991.