# A Practical Comparison of 2-Phase Delay Insensitve Communication Protocols

Markus Schütz, Florian Huemer, Andreas Steininger
*TU Wien, Institute of Computer Engineering*
*Treitlstrasse 3, 1040 Vienna, Austria*
{*markus.schuetz@student, florian.huemer@student, steininger@ecs*}*.tuwien.ac.at*

*Abstract*—**As modern ASIC technologies suffer from substantial delay uncertainties, delay insensitive (DI) communication protocols are receiving increasing attention. Among these, the more energy-saving 2-phase protocols are most attractive. In the literature such protocols are widely covered already, with Level-Encoded-Dual-Rail (LEDR), Level-Encoded-Transition-Signalling (LETS) and Transition Encoding (TranEnc). For the designer, however, it remains difficult to pick the right one, as the discussions in the literature are restricted to the respective protocol at hand, and implementations for the required interfaces are often missing. This paper provides a thorough comparison of existing DI 2-phase protocols with respect to speed, power and coding efficiency, that also encompasses the practical implementation. We propose efficient solutions for the required completion detection and conversion circuits and compare their static power- and dynamic energy consumptions as well as the conversion times through Spice simulations.**

## 1. Introduction

Throughout the past decades we have witnessed a tremendous miniaturization in VLSI technologies. While this has undeniably been key to the enormous advances in digital electronics, the small feature sizes also cause substantial technological challenges. One of these is the high uncertainty of circuit and interconnect delays that results mainly from random dopant fluctuation and process imprecisions in nanoscale technologies. On top of that, delays heavily depend on supply voltage and temperature. In the face of these so called PVT (process/voltage/temperature) variations synchronous design, whose timing fundamentally relies on delay predictions, is reaching its practical limits. Consequently alternative approaches are being considered, particularly for communication. Here the most pronounced one is the delay-insensitive (DI) approach, in which data is encoded in a way to allow the receiver to recognize, by decoding, when a data word is complete and valid. This allows the speed of operation to automatically adapt to the current conditions.

Examples for suitable DI codes are Level-Encoded-Dual-Rail, Level-Encoded-Transition-Signalling and Transition Encoding. These all have in common that their underlying 2-phase protocol is particularly energy efficient, which is why we put a focus on them here. All of these codes have been thoroughly studied in the literature already, and it is well understood which code, e.g., involves the fewest transitions per bit and is hence most energy efficient. However, a designer trying to identify the most appropriate code for his purpose will need a more comprehensive comparison that, most notably, encompasses the interface circuits. After all, the interface circuits of the most energy efficient code may turn out so power hungry that a different choice of code may be beneficial in the end. The key contribution of this paper will be to elaborate such a comparison. To this end we will revisit the existing schemes in the next section and outline their basic properties. In Section 3 we will, based on existing work, elaborate a representative set of code converters for all the above codes. In Section 4 we will perform extensive simulations to determine energy and power consumption for all these solutions, as well as their conversion delays. Section 5 will conclude the paper.

## 2. Background and Related Work

Asynchronous circuits can be classified with regard to the delay assumptions they impose. The class of delay insensitive (DI) circuits uses the weakest assumptions and is hence most attractive. The only requirement is that gate and wire delays are positive and finite. DI protocols can be divided into two major categories. In 4-phase DI protocols two successive valid code words are separated by a null phase that serves as a spacer and does not carry further information. A 4-phase DI code is referred to as 4-phase DI *m-of-n* code if it employs $n$ rails and for every valid codeword exactly $m$ rails are *set to* logic one. The spacer is an invalid code word with usually all bits set to zero. In contrast, 2-phase protocols do not require spacers, but alternate between two disjoint subcodes for separating successive code words. A 2-phase DI code is denoted as a 2-phase DI *m-of-n* code if it employs $n$ rails and two successive valid codewords *differ by* exactly $m$ rails. Both categories have in common, that a codeword once assigned by a transmitter must remain stable until it was acknowledged by a receiver, i.e., it must not glitch.

An important task when using DI codes is the membership test, which determines the phase of the protocol. It is performed by a completion detector that generates a signal $cd$. For 4-phase DI codes $cd$ is 0 during the null phase and 1 when a valid codeword is applied. In a 2-phase DI code,

$cd$ changes its state every time a new valid codeword is applied. 2-phase DI codes can be further subdivided into level-encoded codes, where the logic state of the rails is mapped to a symbol, and transition-encoded codes, where the transitions on the rails between two codewords determine the associated symbol.

Due to the null phase and the fixed number of ones in 4-phase *m-of-n* protocols they are well-suited to implement logic blocks for logical calculations. On the other hand, the null phase entails twice as many transitions, which decreases performance and increases energy consumption. Therefore, 2-phase protocols are typically used for intra- and inter-chip communication. A convenient method for designing asynchronous circuits is to implement local computation blocks with a 4-phase protocol, and connect them by a bus system that utilizes a 2-phase protocol. In the following the most common 2-phase protocols are shortly represented and assessed in terms of their power metrics and coding efficiency. The power metrics $P$ gives the number of rail transitions that have to be performed in order to transmit one data bit. The coding efficiency $R$ states the number of bits that can be encoded per rail.

## 2.1. Level Encoded Dual Rail (LEDR)

LEDR [1], as the name suggests, is a level-encoded 2-phase protocol. For every data bit two rails are employed, one data rail $d$ and one parity rail $p$. The data rail always carries the same logic value as the data bit. Between two successive data items the parity over $p$ and $d$ must change between $even$ and $odd$. So for two successive data items with the same logic value $d$ does not change, but $p$ will, thus introducing the single rail transition required to separate two data items. This makes completion detection for a single-bit signal as simple as parity calculation. For multi-bit data words the phases of all bits are conjuncted by a C gate to obtain the overall phase. Figure 1 illustrates the completion detector (CD) for LEDR codewords.

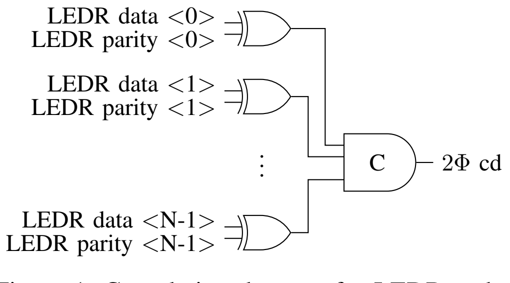


Figure 1: Completion detector for LEDR codes

$$R_{LEDR} = \frac{1}{2},\ P_{LEDR} = 1$$

## 2.2. Level Encoded Transition Signalling (LETS)

LETS [2] is a generalization of LEDR. As for LEDR, LETS transactions are divided into even and odd phases. The codewords consist of $N = 2^n$ data rails, where $n$ is the number of unencoded data bits. For every phase transition, exactly one of the $N$ rails changes its state. Therefore, LETS is a *1-of-N* level-encoded DI 2-phase protocol. LEDR can be considered as *1-of-2* LETS. The properties that make these codes DI are discussed in detail in [2], where also a procedure to derive sets of code words is given. The latter involves a complex generation by bit flip matrices and produces a large number of possible code sets. In the following, a more straightforward method is introduced that delivers one code set that is easy to implement in hardware.

Consider an arbitrary symbol $s$ consisting of $n$ bits. By flipping one of $N = 2^n$ rails it must be possible to reach an arbitrary symbol $s'$ in the complementary phase. This means, that there must be an assignment function $f_a : r_i \mapsto \{d_j | d_j(s) \neq d_j(s')\}$ that maps a change of the state of a rail $r_i$ to a set of data bits that have to change their logical state in order to obtain symbol $s'$ from $s$, where $d_j(s)$ is data bit $j$ of the symbol $s$. Since there must be a rail related to every subset of the power set $\{0,1\}^n$, the truth table with $n$ bits can be used to obtain the function $f_a$. Table 1a shows the truth table for the data bits of a *1-of-4* LETS code, where every line is labeled with the corresponding rail. A one indicates that a data bit is inverted by the corresponding rail. For instance rail $r_0$ inverts both data bits $b_0$ and $b_1$, rail $r_1$ only inverts data bit $b_1$ and so on. The fact that the inversion of one rail causes the change of a subset of data bits can be expressed in terms of XOR functions, as illustrated in Table 1a. By elaborating the truth table with all combinations of rails $r_i$ and the data bits $d_j$ as outputs, the level-encoded codewords can be derived as shown in Table 1b.

|  | $b_1$ | $b_0$ |
|---|---|---|
| $r_3$ | 0 | 0 |
| $r_2$ | 0 | 1 |
| $r_1$ | 1 | 0 |
| $r_0$ | 1 | 1 |

$b_1 = r_0 \veebar r_1$
$b_0 = r_0 \veebar r_2$

(a)

| $r_3$ | r2 | r1 | r0 | $b_1$ | $b_0$ |
|---|---|---|---|---|---|
| X | 0 | 0 | 0 | 0 | 0 |
| X | 0 | 0 | 1 | 1 | 1 |
| X | 0 | 1 | 0 | 1 | 0 |
| X | 0 | 1 | 1 | 0 | 1 |
| X | 1 | 0 | 0 | 0 | 1 |
| X | 1 | 0 | 1 | 1 | 0 |
| X | 1 | 1 | 0 | 1 | 1 |
| X | 1 | 1 | 1 | 0 | 0 |

(b)

TABLE 1: Derivation of valid 1-of-4 LETS codewords

As described in [2], it can also be observed from this construction method, that there is more than one codeword mapped to every symbol. This property is referred to as *multicode*. Moreover, it can be seen that the whole code space has to be assigned. Based on the same arguments as for the LEDR membership test, the phase of a LETS codeword is derived by calculating its parity.

$$R_{1-of-N\ LETS} = \frac{\lfloor \log_2(N) \rfloor}{N},\ P_{1-of-N\ LETS} = \frac{1}{\lfloor \log_2(N) \rfloor}$$

## 2.3. Transition Encoding (TranEnc)

There are approaches that use the conversion from a 4-phase to a 2-phase protocol [3], but generic converter circuits have not been investigated yet. Therefore, this issue will be addressed in this work. Basically, TranEnc is no autonomous protocol, but a transformation process. Any DI 4-phase code can be transformed into a TranEnc code. Transition Encoding employs the same number of rails as the original 4-phase code, but only half of the transitions. Therefore, if a rail in the 4-phase codeword is logic one, the state of the corresponding rail in the TranEnc codeword has to be flipped. That means, logic states are mapped to state transitions and in the course of this the null phase is removed. This results in a 2-phase DI code. The proof that the resulting code is DI can be given in the same way as for the original 4-phase code, but instead of considering the states of codewords, transitions between two adjacent codewords have to be regarded.

$$R_{TranEnc} = \frac{R_{4\Phi}}{2},\ P_{TranEnc} = P_{4\Phi}$$

## 3. Code Converter Implementation

As already mentioned, 2-phase protocols are employed for global communication, whereas 4-phase protocols are preferred for calculations. The conversions between 2- and 4-phase protocols therefore are important tasks within asynchronous circuits. A simple 4-phase protocol often used for building calculation blocks is the 4-phase Dual Rail (DR) code. It employs two rails for every data bit. The true rail $b.t$ is set to one, if the corresponding data bit is 1 and the false rail $b.f$ is set to one if the data bit is 0. In the null phase both rails remain at zero. The membership test can simply be performed by ORing the two rails. In the following, conversion circuits from and to DR are given. The LEDR and LETS converters are inspired from [4] and [2], respectively. The data path of the converters is the same as for the original circuits, but we adapted the control logic in order to separate encoder and decoder. This enables a more generic usage, since it now is possible to connect one encoder to several decoders. To that end the acknowledge signals from the decoders have to be connected by a C gate and this overall acknowledge signal is than fed to the encoder. Moreover, we introduce generic converters for TranEnc, that are inspired by the work of [5] and [3].

### 3.1. DR $\Leftrightarrow$ LEDR

Figure 2 depicts a bitslice of the 4-phase DR to LEDR converter. The associated control logic is shown in Figure 3. If the 4-phase CD sets $4\Phi cd$ to one, the D latch becomes opaque. Meanwhile, the encoder sets the LEDR data rail according to the 4-phase input. Depending on the last $2\Phi ack$, which indicates the old phase, the new parity rail is calculated. If the old phase was 0, then according to the LEDR code, the parity bit has to be the inverse of the data bit in order to get to the new phase 1. If the old phase was 1, the parity bit is equal to the new data bit.

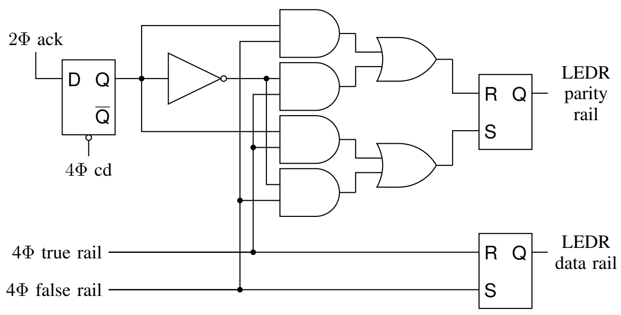


Figure 2: 4-phase-DR-to-LEDR encoder bitslice

After the 4-phase data was converted, $2\Phi cd$ changes its state and in consequence $4\Phi ack$ is set to one. The null phase is initiated and $4\Phi cd$ becomes zero. The new $2\Phi ack$ is passed through the D latch and $4\Phi ack$ becomes zero. The 4-phase transaction is completed and the encoder is ready to receive new data.

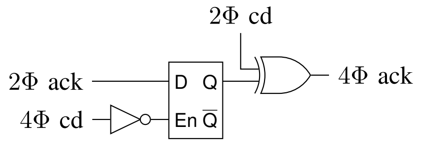


Figure 3: Control block for LEDR/LETS encoder

Figure 4 shows one bitslice of the converter from LEDR to DR and Figure 5 the associated control logic.

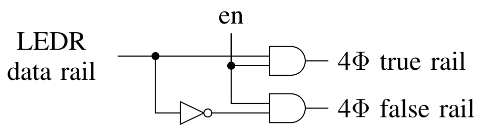


Figure 4: LEDR-to 4-phase-DR decoder bitslice

$4\Phi ack$ is initially zero and therefore the first latch is opaque. $2\Phi cd$ is obtained from a LEDR CD as depicted in Figure 1. If this signal changes (i.e. a new LEDR codeword is received), the XOR gate sets $en$ to one and the encoder logic sets the $true$ and $false$ rail corresponding to the LEDR input. After the subsequent 4-phase logic has processed the data $4\Phi ack$ is set to one. The first latch becomes transparent and $en$ is reset. A null phase is introduced by the decoder. If it is acknowledged by $4\Phi ack = 0$, the current phase $2\Phi cd$ is passed to $2\Phi ack$ through the two latches, which completes the 2-phase protocol. The decoder now is ready to receive new data.

### 3.2. DR $\Leftrightarrow$ LETS

The LETS converters use the same control circuits as the LEDR converters. Figure 7 shows the DR-to-LETS

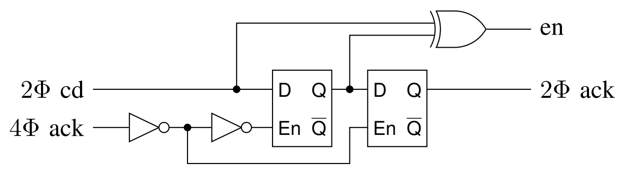


Figure 5: Control block for LEDR/LETS decoder

converter. As soon as the DR codeword enters its null phase ($4\Phi cd = 0$), the latches in the storage block store the previous LETS codeword for decoding. The comparator block then decides which LETS rails have to be flipped for obtaining the new LETS codeword from the old one so that the required 4-phase data is properly encoded. The select block then flips the states of the output rails accordingly.

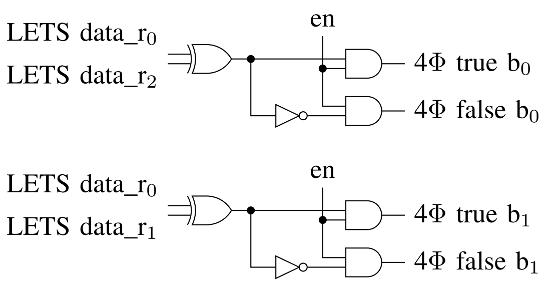


Figure 6: LETS-to 4-phase-DR decoder bitslice

The LETS-to-DR decoder depicted in Figure 6 implements the XOR functions described in Section 2.2. If its $en$ input is set, it forwards its decoded data to the DR output.

### 3.3. DR ⇔ TranEnc

Figure 8a shows the DR-to-TranEnc converter and Figure 8b the corresponding control logic. If $4\Phi cd$ becomes one and the 4-phase input is also one, the TranEnc encoder inverts the corresponding 2-phase rail. The control logic generates a logic one on $4\Phi ack$ whenever $2\Phi ack$ changes its value; that is, whenever a 2-phase request is acknowledged. If the CD indicates that the 4-phase protocol is in its null phase, $4\Phi ack$ is reset.

The TranEnc-to-DR decoder is depicted in Figure 9a and its control logic in Figure 9b. $4\Phi ack$ and the 4-phase rail initially are zero. If the 2-phase rail changes its value, the 4-phase rail is set to one. After the 4-phase request was processed, the subsequent stage sets $4\Phi ack$ to one and the 4-phase rail therefore becomes zero. The 4-phase cycle is then

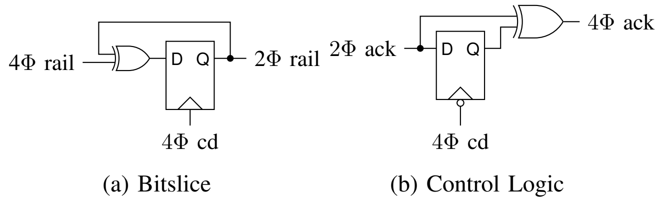


Figure 8: Bitslice and control logic of the TranEnc encoder

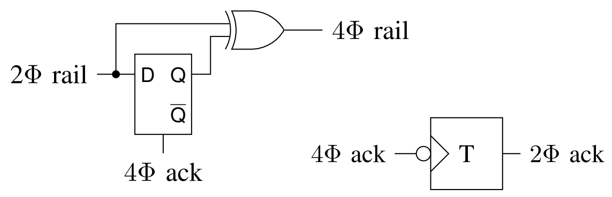


Figure 9: Bitslice and control logic of the TranEnc decoder

complete. Moreover, on the falling edge of $4\Phi ack$, $2\Phi ack$ is toggled by the toggle flip flop, which also completes the 2-phase transaction.

A generic CD for TranEnc codes is illustrated in Figure 10. The TranEnc codeword is first decoded to a 4-phase data word and then a 4-phase CD performs a membership test. The result is then fed back to the decoder's $4\Phi ack$ input. The $2\Phi ack$ output is used as $2\Phi cd$ signal that indicates the phase of the 2-phase protocol.

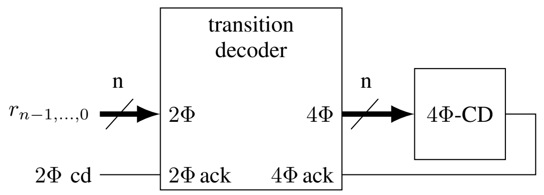


Figure 10: Generic m-of-n NRZ completion detector

## 4. Comparison by Simulation

We simulated all the converter circuits from Section 3 with the goal to compare their performance, power- and energy- consumption. Our focus was not on attaining their absolute physical characteristics, but rather on providing a fair relative comparison between the codes using the same conditions (which is exactly what is missing in the literature so far). We studied all the codes for 2 and 4 bit data width, whereby we encoded 4-bit LETS by 16 rails, not by two 1-of-4 LETS codes. For the TranEnc code, beyond the 2-bit DR and 4-bit DR code, we also implemented a version with an intermediate *2-of-7** code which will be denoted as TranEnc*. Here the DR data is first converted to an incomplete *2-of-7* code as described in [6], and then transformed into a TranEnc code.

We decided to use analog circuit-level simulations in Spice, since this allows us to conveniently measure power and energy. We generally neglect the influence of routing within our small elementary function blocks, but of course when reasoning about the codes' energy efficiency we do consider the influence of the interconnect over which the 2-phase communication runs. More specifically, we use the wire model shown in Figure 11, with an interconnect hop consisting of a buffer and an RC element. This emulates the wires in an integrated circuit, which are buffered in regular distances.

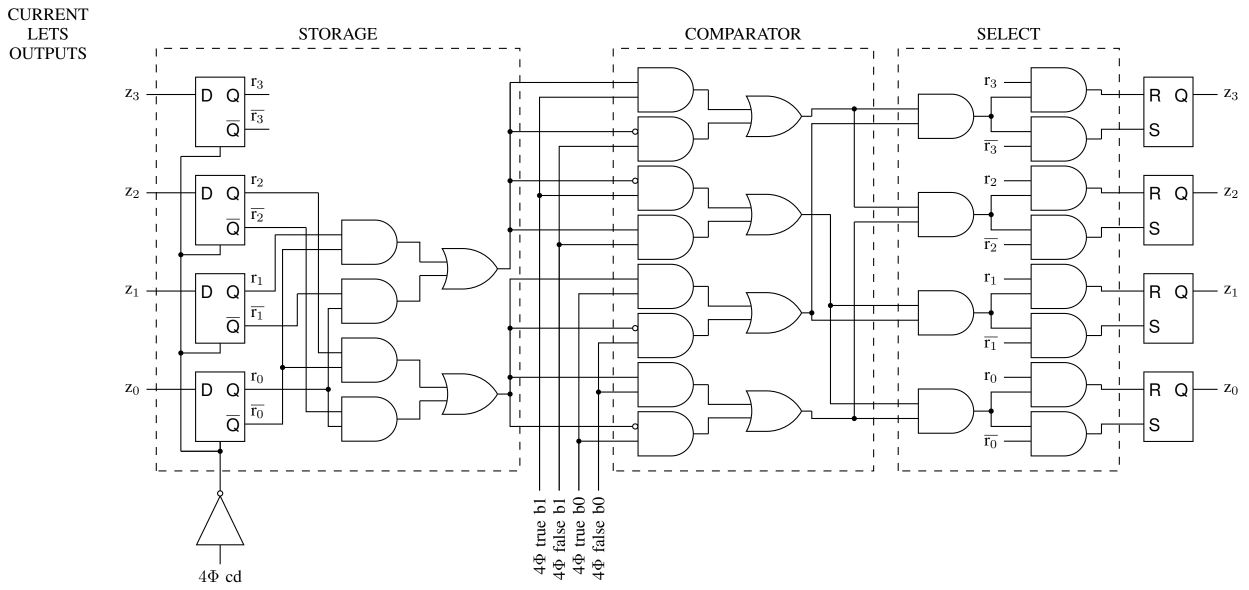


Figure 7: 4-phase-to-LETS encoder

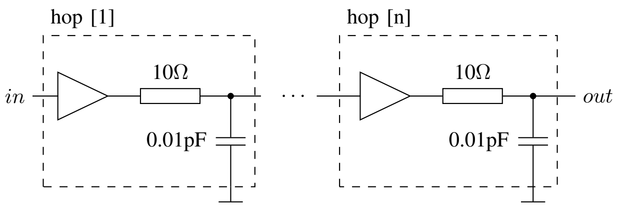


Figure 11: Interconnect Model

For the simulations random input vectors were applied to all circuits, with the interconnect attached, and the static power- and dynamic- energy consumption were measured, as well as the forward latency.

The results of our simulations are summarized in Figure 12. The number of interconnect hops was chosen as an index on the x-axis. As can be seen, LETS has the highest static power consumption and the lowest dynamic energy consumption. The latter indicates that this code is very efficient, using only few transitions, but at the same time its complex encoder causes high static power consumption. So LETS will leverage its advantages for long wires.

In the 2 bit case, the other codes have similar characteristics. In the 4 bit case, the dynamic energy consumption of the TranEnc* circuit is lower than for LEDR and TranEnc but higher than the consumption of LETS. This can be ascribed to the good power metrics of the *2-of-7** code ($P_{TranEnc*} = \frac{1}{2}$). Moreover, the scaling effects on these codes can be inferred from comparing the respective 2- and 4-bit graphs.

Table 2 shows the corresponding encoder forward latencies, i.e. the time interval from applying a data word at the input of the encoder until the codeword is processed at the output. It indicates LEDR as the clear winner.

Table 3 gives the decoder characteristics in terms of their forward latencies, power- and energy consumptions. Here, the TranEnc offers a very low latency, but compared to the other decoder circuits a higher energy consumption.

| **LEDR** | | **LETS** | | **TranEnc** | | |
|---|---|---|---|---|---|---|
| 2-bit | 4-bit | 2-bit | 4-bit | 2-bit | 4-bit | 2-of-7* |
| 15.94 | 17.06 | 23.93 | 36.46 | 25.85 | 37.69 | 61.41 |

TABLE 2: Encoder forward latencies [ps]

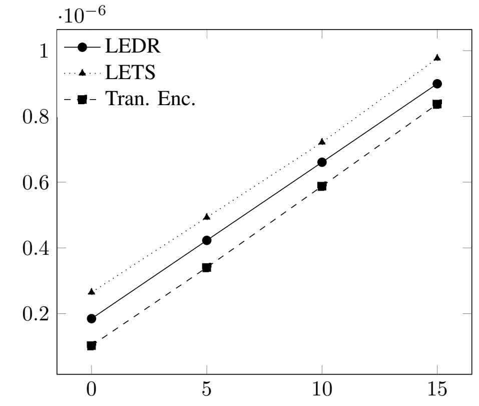
| **Code** | | **Latency** | **Power Cons.** | **Energy Cons.** |
|---|---|---|---|---|
| **LEDR** | 2 bit | 165.19 ps | 62.67 nW | 48.26 fJ |
| | 4 bit | 250.9 ps | 105.0 nW | 84.21 fJ |
| **LETS** | 2 bit | 169.64 ps | 66.04 nW | 40.9 fJ |
| | 4 bit | 377.3 ps | 198.77 nW | 85.21 fJ |
| **TranEnc** | 2 bit | 63.68 ps | 69.82 nW | 34.59 fJ |
| | 4 bit | 76.5 ps | 119.43 nW | 60.84 fJ |
| | 2-of-7* | 156.3 ps | 210.15 nW | 106.86 fJ |

TABLE 3: Decoder characteristics

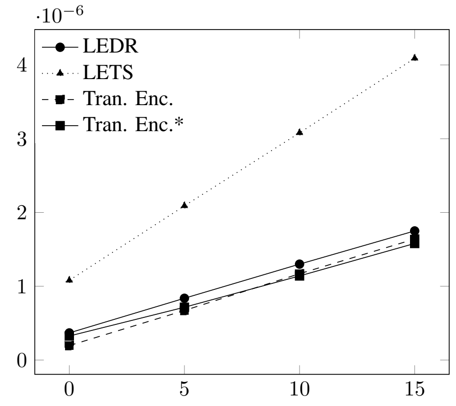
| **Code** | | **Latency** | **Power Cons.** | **Energy Cons.** |
|---|---|---|---|---|
| **LEDR** | 2 bit | 44.59 ps | 15.63 nW | 4.91 fJ |
| | 4 bit | 89.89 ps | 36.15 nW | 15.01 fJ |
| **LETS** | 2 bit | 30.7 ps | 8.04 nW | 1.77 fJ |
| | 4 bit | 143.1 ps | 44.54 nW | 3.6 fJ |
| **TranEnc** | 2 bit | 387.89 ps | 91.33 nW | 62.39 fJ |
| | 4 bit | 554.64 ps | 164.57 nW | 125.03 fJ |
| | 2-of-7* | 597.1 ps | 249.06 nW | 149.51 fJ |

TABLE 4: Completion detector characteristics

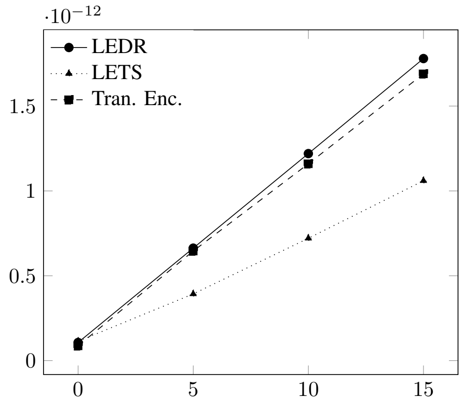
Table 4 gives the characteristics of the CDs. Since LEDR and LETS CDs consist only of few combinational gates while the TrancEnc CDs need to convert their inputs before performing the membership test, the latter have a longer latency as well as a higher energy consumption.



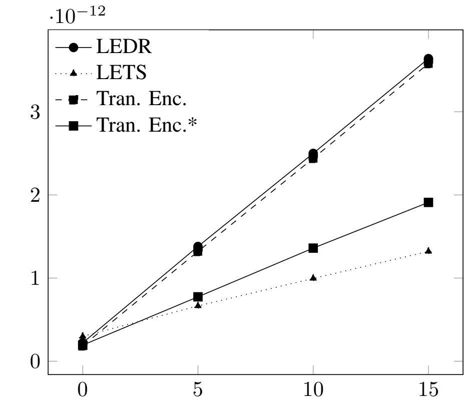
(a) Static power: 2-bit circuits



(b) Static power: 4-bit circuits



(c) Dynamic energy: 2-bit circuits



(d) Dynamic energy: 4-bit circuits

Figure 12: Static power and dynamic energy consumption of encoder circuits and interconnect over interconnect length (in number of hops)

## 5. Conclusion

Delay insensitive codes are an attractive means for dealing with the PVT variations seen in modern CMOS technologies; in conjunction with 2-phase protocols they can further achieve high energy efficiency. In this paper we have reviewed the most relevant 2-phase DI codes, namely Level-Encoded-Dual-Rail, Level-Encoded-Transition-Signalling and Transition Encoding. Our contribution is to present encoder/decoder circuits for all these codes (actually converters to/from 4-phase Dual-Rail) most of which either did not yet exist or have been optimized. Furthermore we have performed a comprehensive comparison of these codes, including the encoder/decoder circuits, with respect to static power, dynamic energy and propagation delay. This shall aid the designer to pick the right code for his purpose, and provide him implementation templates.

Our results indicate (and quantify) that LETS has the most energy-efficient encoding, but requires the most power hungry encoder. So its use is justified only for long interconnect lines. LEDR, on the other hand, has the most lightweight and fastest implementation, while TranEnc, due to its demanding completion detection, does not seem to offer specific advantages in general.

## References

[1] M. E. Dean, T. E. Williams, and D. L. Dill, "Efficient self-timing with level-encoded 2-phase dual-rail (ledr)," in *Proceedings of the 1991 University of California/Santa Cruz Conference on Advanced Research in VLSI.* Cambridge, MA, USA: MIT Press, 1991, pp. 55–70.

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

[3] Y. Shi, S. Furber, J. Garside, and L. Plana, "Fault tolerant delay insensitive inter-chip communication," in *Asynchronous Circuits and Systems, 2009. ASYNC '09. 15th IEEE Symposium on*, May 2009, pp. 77–84.

[4] A. Mitra, W. McLaughlin, and S. Nowick, "Efficient asynchronous protocol converters for two-phase delay-insensitive global communication," in *13th IEEE International Symposium on Asynchronous Circuits and Systems*, 2007, pp. 186–195.

[5] M. Cannizzaro, W. Jiang, and S. Nowick, "Practical completion detection for 2-of-n delay-insensitive codes," in *2010 IEEE International Conference on Computer Design (ICCD)*, 2010, pp. 151–158.

[6] W. Bainbridge, W. B. Toms, D. Edwards, and S. Furber, "Delay-insensitive, point-to-point interconnect using m-of-n codes," in *Ninth International Symposium on Asynchronous Circuits and Systems*, 2003, pp. 132–140.