Soft Self-Synchronising Codes for Self-Calibrating Communication

Frédéric Worm
Processor Architecture Laboratory
Swiss Federal Institute of Technology
Lausanne (EPFL)
Lausanne, Switzerland
Frederic.Worm@epfl.ch

Paolo Ienne
Processor Architecture Laboratory
Swiss Federal Institute of Technology
Lausanne (EPFL)
Lausanne, Switzerland
Paolo.Ienne@epfl.ch

Patrick Thiran
Laboratory for Computer
Communications and Applications
Swiss Federal Institute of Technology
Lausanne (EPFL)
Lausanne, Switzerland
Patrick.Thiran@epfl.ch

Abstract

Self-calibrating designs are gaining momentum in both the computation and communication worlds. Instead of relying on the worst-case characterisation of design parameters, self-calibrating systems determine autonomously the boundary of correct behaviour, and set design parameters accordingly. In this paper, we focus on the communication task. We model errors due to over-aggressive operation and derive a channel model. We show that self-synchronising codes achieve completely reliable communication over this channel model, and study a known example, LEDR (Level Encoded 2-Phase Dual-Rail)*, which is an improvement of the well-known Dual-Rail code. Then, we introduce a family of coding schemes which are a generalisation of LEDR, and study their performance over our channel model. We observe that the wiring overhead can be significantly reduced at the expense of a limited loss in reliability. Finally, we extend our channel model to include additive noise, and show that in this more general situation a specific instance of our coding scheme has similar or better performance than LEDR, at a smaller wiring overhead.*

INTRODUCTION

Self-calibrating designs rely on two key hypothesis, namely the possibility to (i) detect that the system is not operating correctly, and (ii) improve the system reliability at some cost—e.g., energy.

In this paper, we focus on the former issue, in the context of communication. As far as communication tasks are concerned, correct operation is assessed if it is possible to determine whether a sequence of data has been received correctly. The voltage and frequency of the link can then be varied to ensure reliability. In practice, we are looking for an encoding scheme that determines, *independently of the speed* at which the link is operated, (a) if the received data is correct and (b) if it is the next piece of data to be received in sequence. The encoding has to be speed-independent since no assumption is made on signal propagation time.

As a motivational example illustrating a situation where such an encoding scheme is required, consider a communication link where the voltage and frequency are set adaptively by a self-calibrating controller [11], as depicted in Figure 1. The encoding scheme has to detect errors originated by over-aggressive operation. The difficulty does not come from the fact that the frequency is variable, but is caused by the "blind" setting of voltage at a given frequency. It may result in situations with (close to) unity bit error rate, which renders the problem quite unique.

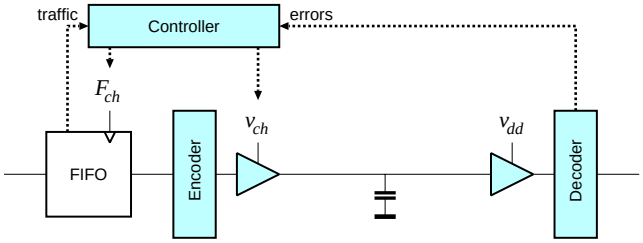

Figure 1. The controller determines link parameters, v_{ch} and F_{ch}, based on transfer errors. In case the parameters have been set too aggressively, or if an error corrupts the data, the decoder has to raise an error flag.

The contribution of this paper is twofold:

- We develop a channel model for self-calibrating communication, and show that self-synchronising codes, such as Dual-Rail and LEDR, achieve completely reliable communication over this channel.
- We introduce a new family of codes, *soft self-synchronising codes*, obtained by generalising the LEDR code. We compare one instance of soft self-synchronising code with LEDR, and recommend our coding scheme for self-calibrating communication due to (a) its lower wiring overhead and (b) its better tolerance to additive errors.

The next section gives a short survey over different pieces of work related to self-calibrating designs, and briefly introduces *self-synchronising codes*. Then, we define the type of errors occurring on a data link operated too aggressively and give a formal model of the relevant communication channel. The following section defines self-synchronising codes in the framework of the introduced model; in the same section, we recall the LEDR self-synchronising code, which is systematic and does not require a spacer. The successive section describes the novel family of encoding scheme we propose, the *alternating phase* encoding, and explains how it is obtained by a generalisation of LEDR. We conclude by extending the channel model discussed earlier in the paper to include additive errors; we compare the LEDR code

with a specific embodiment of alternating phase encoding in terms of reliability and wiring overhead. A short section summarises our achievements.

RELATED WORK

Self-Calibrating Designs

The use of adaptive design techniques in extremely aggressive designs is not new. For example, in [6] the regional clock skew is adaptively tuned at power-up using relatively complex controllers to compensate for local process variations across a single die. In a recent paper [4], the possibility of exploiting devices in subcritical regions for *Digital Signal Processing (DSP)* was presented; in that case, errors arising from the subcritical voltages are compensated by the DSP algorithms. The work published in [1] has given more momentum to self-calibration by showing that the design of a whole processor without noise margin is indeed possible. The work presented in [9, 10] is similar in the intents, but applies to a different domain (communication instead of computation).

Self-Synchronising Codes

When transferring a sequence of data, the notion of time is essential in order to determine the correct ordering of data. For example, assume that the sequence $\{d_k\}$, with k denoting the time index, has to be transferred. How to account for the fact that d_{k+1} is preceded by the data piece d_k? The so-called *synchronous* approach relies on a global time reference, namely a clock. In the considered example, the clock contains the information that, if d_k is the sampled data at time index k, d_{k+1} will be sampled at the next rising edge.

Self-synchronisation [7] constitutes an alternative approach that samples time by the events taking place in the system or environment. Self-synchronising codes are embodiments of this concept and enable to transfer a sequence of data without the need of a clock. Such codes have been developed and used extensively by the asynchronous community. Notice that self-synchronising codes are of direct interest for self-calibrating communication. For example, a signal sampled too aggressively does not have time to transition. As a result, the missing transition event prevents time from advancing. A very natural way of preserving data ordering consists in separating explicitly two consecutive data pieces with a spacing symbol. This is in fact the idea of Dual-Rail, a well known self-synchronising code, that dedicates a codeword to data separation [7]. Dual-Rail encodes binary data into 2-bit codewords: logic-0 as $(0, 1)$, logic-1 as $(1, 0)$, while the codeword $(0, 0)$ is reserved as a spacer (see Table 1).

The minimum wiring overhead required by a self-synchronising code is given in [7] as a function of K, the number of information bits. For example, at least 3 redundant bits are required for a self-synchronising code encoding 16 information bits. However, such self-synchronising codes are not systematic, which increases significantly the encoder complexity, or require a spacer, which is bandwidth inefficient. (Codes are said *systematic* when the information bits are transmitted as such.) LEDR [3] is a systematic code derived by an optimisation of Dual-Rail and it avoids the insertion of an explicit spacer token in the dataflow. We discuss LEDR more in depth in a later section: our encoding scheme borrows some ideas from it.

Data	Encoding	Meaning
0	$(0, 1)$	logic-1
1	$(1, 0)$	logic-0
N.A.	$(0, 0)$	spacer
N.A.	$(1, 1)$	not used

Table 1. The three logic states used by Dual-Rail. Because the spacer is the all-zero vector and valid codewords have weight 1, completion of bit transitions is unambiguously detected.

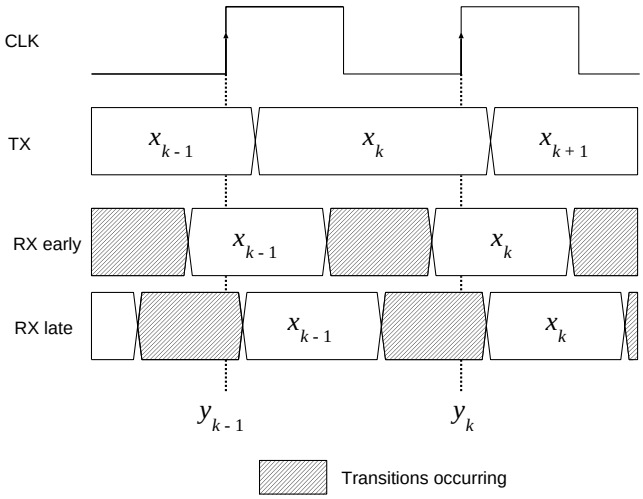

Figure 2. Due to propagation delay of the signal "RX early", the sampling times are correct. However, if the clock is too aggressive with respect to the signal propagation delay, as in the case of "RX late", the sampled data is wrong.

MODELLING TIMING ERRORS

We would like to capture the following phenomenon: assume a binary signal is sampled synchronously, but with a clock whose period is not set in function of a worst-case characterisation of the signal propagation delay. That is, there is a risk that the signal is sampled while the data has not yet transitioned correctly: the receiver samples either the previous data or has metastability problems. Figure 2 illustrates such a situation.

Throughout the paper, we use the following notations and definitions: k denotes the time index, x_k designates the channel input, and y_k the channel output. Def. 1 defines a signal transition. As for any channel, an error occurs when $y_k \neq x_k$.

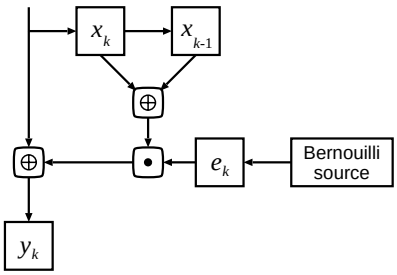

Figure 3. Timing-error channel. The dot operator denotes bitwise *and* of input values.

Definition 1 (Transition) *Let $z_k \in \{0,1\}$ be a binary signal. We define a transition on the signal z at time index k if and only if $z_k \neq z_{k-1}$ or, equivalently, $z_k = z_{k-1} \oplus 1$, with $\oplus$ denoting* exclusive or.

We denote by $e_k \in \{0,1\}$ the incorrect data sampling at time index k—i.e., $e_k = 1$ means that the sampling has been too early, while $e_k = 0$ indicates a correct sampling. Notice that $e_k = 1$ does not necessarily mean that an error occurs; an error occurs only if, in addition, a transition occurs. An error occurs if and only if $y_k \neq x_k$, which is the case if and only if (i) a transition occurs—i.e., $x_k \neq x_{k-1}$—and (ii) the sampling has been too early—i.e., $e_k = 1$.

We denote by *timing error* such an error process, and model the success of sampling with a Bernouilli random variable. Namely, e_k is described by a sequence of *independent and identically distributed (i.i.d.)* Bernouilli random variables:

$$e_k = \begin{cases} 1 & \text{with probability } \varepsilon \\ 0 & \text{with probability } 1-\varepsilon, \end{cases} \tag{1}$$

We have: $P\left(y_k \neq x_k \mid x_k \neq x_{k-1}\right) = \varepsilon$. For an N-bit wide channel with timing errors, we write $\vec{e}_k = (e_{k,1}, \ldots, e_{k,N})$, where the terms $e_{k,i}$ (for $i = 1, \ldots, N$) are N i.i.d. random variables defined by Eq. (1).

The relation between the channel output $\vec{y}_k$and the channel input $\vec{x}_k$is given by

$$\vec{y}_k = \vec{x}_k \oplus \vec{e}_k \cdot (\vec{x}_k \oplus \vec{x}_{k-1}), \tag{2}$$

with $\cdot$ denoting the bitwise *and* operation between binary values and $\oplus$ denoting the bitwise *exclusive or*. Eq. (2) simply models the fact that, for every component i of the vectors $\vec{y}_k$, $\vec{x}_k$, and $\vec{e}_k$, it is $y_{k,i} = x_{k,i}$ if $e_{k,i} = 0$ or $x_{k,i} \oplus x_{k-1,i} = 0$, and $y_{k,i} = x_{k-1,i}$ otherwise. We call a channel described with Eq. (2) a *Timing-Error Channel*, which we denote TEC (ε). A graphical representation of the timing-error channel is in Figure 3.

The next section introduces the LEDR code, and shows that, as any self-synchronising code, it detects all timing errors.

LEDR CODE

We formulate the following definition of self-synchronising codes, which is equivalent to the one of [7].

u	ϕ	r	$\vec{x} = (u \mid r)$
0	0	0	$(0,0)$
0	1	1	$(0,1)$
1	1	0	$(1,0)$
1	0	1	$(1,1)$

Table 2. Possible codewords of the LEDR code. The encoder outputs a sequence of codewords with alternating phase.

Definition 2 (Self-synchronising codes) *A code is self-synchronising if, and only if, it detects any possible error over a timing-error channel* TEC (ε) *with* $0 \leq \varepsilon \leq 1$.

We focus on a specific self-synchronising code: the LEDR code [3]. LEDR is a systematic code, henceforth enabling low-complexity encoders, and, contrary to codes like Dual-Rail, does not use any spacer token.

Dual-Rail, which we have described in the Related Work section, has a high overhead in wiring (100%, as each information bit is encoded into a 2-bit codeword) as well as in timing (100%, due to the explicit spacer). LEDR is an improvement of Dual-Rail that avoids using an explicit spacer token. Before describing the code, we introduce the following notations. We denote the information bits by $\vec{u}$. Throughout the paper, we only consider systematic codes. We write therefore $\vec{x} = (\vec{u} \mid \vec{r})$ to indicate that a codeword $\vec{x}$ consists of the information bits $\vec{u}$ concatenated to the redundant bits $\vec{r}$. We refer to a code that encodes K bits of information into an N-bit codeword as a (N, K) code.

Dual-Rail and LEDR are both $(N = 2, K = 1)$ codes—i.e., they encode one information bit into a 2-bit codeword. The explicit spacer symbol of Dual-Rail is implicitly expressed in the LEDR codeword sequencing. Namely, each codeword is said to have a *phase*, which we define as follows.

Definition 3 (Phase of a data sequence) *Let $\{\vec{u}_k\}$, with k denoting the time index, be a sequence of data. The phase, ϕ_k, of the data $\vec{u}_k$ is a binary information obtained as the parity of the sequence index k:*

$$\phi_k = k \bmod 2 \in \{0,1\}.$$

By extension, the phase of a codeword is the phase of the encoded information.

Notice that, by assumption, we restrict the phase to a binary value. LEDR makes the spacer symbol implicit by exploiting the opposite phase of two consecutive codewords and encoding the phase in the codeword: The redundant bit r is computed as the sum of the information bit u, and the phase ϕ (i.e., $r = u \oplus \phi$). As a result, the phase of an LEDR codeword can be obtained directly by checking whether the information and redundant bit are equal ($\phi = 0$) or different ($\phi = 1$). Table 2 summarises the encoding scheme of LEDR. We verify, in the following example, that LEDR detects all timing errors.

Example 1 (LEDR is self-synchronising) *Consider a 2-bit timing-error channel over which information is transmitted*

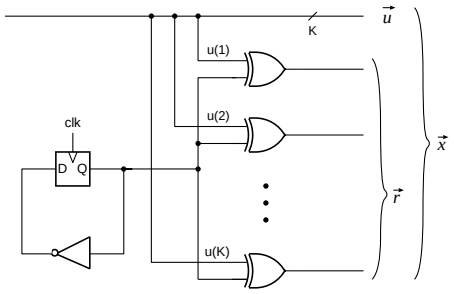

Figure 4. A possible implementation of a LEDR encoder. The flip-flop generates the alternating phase bit.

with the LEDR code. Analysis of Table 2 shows that exactly *one bit transitions between any two consecutive codewords. As a result, the only error that can occur is that a bit that should transition does not. In this case, LEDR declares an error, since the phase of the received data is unchanged.*

When transmitting more than one information bit, each redundant bit is computed independently as the sum of its information bit and the phase. Figure 4 depicts a possible implementation of an LEDR encoder: the hardware cost is very low. For asynchronous communication, the phase bit would off course not be inverted with a flip-flop, but after receiving an acknowledgement from the decoder. On the contrary, if LEDR had to be used as an encoding scheme for the system of Figure 1, the encoder and decoder would be provided with a synchronous clock, so that any over-aggressive sampling would be detected by the decoder.

Self-synchronisation is a crucial property for asynchronous communication. Nonetheless, there are two reasons why codes with lower overhead than 100% are of interest. First, there are applications (e.g., self-calibrating communication) that may not tolerate a 100% wiring overhead, while they may accommodate a nonideal detection capability. Secondly, the timing-error model is ideal in the sense that it neglects additive errors: In reality, there are noise sources other than late bit transitions, and it is unclear whether LEDR performs better or not than other encoding schemes, including those with a lighter overhead. We defer this question to a later section and continue, in the next section, with the derivation of a low overhead encoding scheme.

ALTERNATING PHASE ENCODING

As shown in Figure 4, each information bit is involved, with the alternating phase bit, in exactly one parity check relation. A very straightforward modification enabling to reduce the overhead of redundant bits consists in including more than one information bit into a parity check relation. For example, consider the transfer of 32 information bits. Instead of computing 32 redundant bits with 32 independent encoders, as LEDR would do, one could very well use, for instance an 8-bit CRC, to generate only 8 redundant bits. Each redundant bit is computed by only one encoder as a sum of some information bits, and the alternating phase bit.

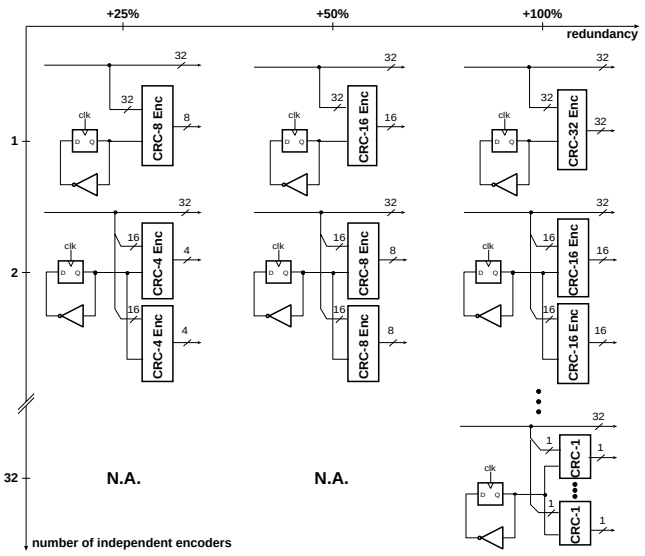

Figure 5. Different options of alternating phase encoding. On the bottom right, we have the LEDR code. All options with more than one independent encoder are particular cases of the top row.

This example underlines that two parameters can be varied, namely: (i) the number of independent encoders, and (ii) the amount of redundancy added per encoder. Since LEDR is a $(N = 2, K = 1)$ code, the number of independent encoders is necessarily equal to the number of information bits to transfer.

Figure 5 shows different encoding options, including LEDR. For the rest of the paper, we focus on the most general case: the alternating phase encoding with only one independent encoder. When giving numerical examples, the encoder assumed consists of an 8-bit CRC generated by the polynomial $x^8 + x^2 + x + 1$. This choice is motivated by a piece of work which studied the performance of 8-bit CRC over the binary symmetric channel and concluded that this polynomial has the best overall performance [2]. We have observed on a few examples that this property seems to be preserved for the timing-error channel. We name the design option with only one encoder computing the CRC generated by this polynomial the *CRC-8 alternating phase encoding*.

Before describing more precisely the alternating phase encoding, we need some notations to describe the operation of a single encoder. By encoding the information bits $\vec{u}$ with a CRC encoder, one gets the redundant bits $\vec{r}$; we write: $\vec{r} = \text{CRC-Enc}\,(\vec{u})$. Decoding consists in computing, from the received data $\vec{y}_k$, the syndrome $\vec{s}$ [5], which we write $\vec{s} = \text{CRC-Dec}\,(\vec{y}_k)$. Eq. (3) shows how a codeword $\vec{x}_k$ is obtained from the information bits $\vec{u}_k$ with the alternating phase encoding:

$$\vec{x}_k = (\vec{u}_k \mid \text{CRC-Enc}\,(\phi_k \mid \vec{u}_k))\,, \qquad (3)$$

where ϕ_k is the phase of the sequence $\vec{u}_k$ provided by the

encoder. Eq. (4) describes the decoding procedure:

$$\vec{s}_k = \text{CRC-Dec}\left(\phi_k \mid \vec{y}_k\right), \tag{4}$$

where ϕ_k is the phase of the sequence $\vec{u}_k$ provided by the decoder. The decoder declares an error if and only if the syndrome is not the all-zero vector—i.e., if and only if the received data does not belong to the code. To summarise, the alternating phase bit, ϕ_k,

- flips each time new data has to be encoded, or decoded,
- is not transmitted over the channel, but generated both by the encoder and decoder, and
- is involved both in the computation of the redundant bits and the syndrome.

Example 2 shows that the alternating phase encoding detects all timing errors when the bit error rate ε is 1.

Example 2 (Timing-error channel TEC ($\varepsilon = 1$)) *Consider a timing-error channel,* TEC $(\varepsilon = 1)$*, this time using the alternating phase encoding scheme. Since any transition fails, the received data is equal to the preceding one ($\vec{y}_k = \vec{y}_{k-1}$), but the decoder appends the alternating phase bit that has flipped. As a result, the decoder encounters a weight-1 error, which is deterministically detected.*

The residual word error rate, a standard reliability metric, is the probability that the decoder declares a word correct, while it is actually corrupted. We indicate it as $\mathrm{P}_{\text{u.w.e.}}$. We derive in [8] an upper-bound and an approximation for alternating phase encoding with 1 encoder, over a timing-error channel. We obtain the following expressions:

$$\mathrm{P}_{\text{u.w.e.}} \leq \frac{1}{2^K} \sum_{i=1}^{N} A_i \varepsilon^i \left((1-\varepsilon)^{d-1} + \left((2-\varepsilon)^{C(i)} - 1 \right) \right) \tag{5}$$

and

$$\mathrm{P}_{\text{u.w.e.}} \cong \frac{1}{2^K} \sum_{i=1}^{K} A_i \varepsilon^i \cdot \left\{ (1-\varepsilon)^{d-1} + (1-\varepsilon)^{\left\lfloor \frac{N-K}{2} \right\rfloor} \left((2-\varepsilon)^{K-i} - 1 \right) \right\}, \tag{6}$$

with d the minimum distance of the code, A_i the number of codewords of weight i, and $C(i)$ a constant that depends on the code and the weight i. (The weight of a codeword is the number of 1s it contains.) The A_i and d are well-known characteristics of a code which can be readily obtained [5].

The upper bound and approximation of Eqs. (5) and (6) are plotted in Figure 6, for the CRC-8 alternating phase encoding mentioned at the beginning of this section. The residual word error rate decreases very rapidly with the raw bit error rate: for example, a residual word error rate of 10^{-10} is already achieved for a raw bit error rate ε as high as 10^{-3}. As it can be seen from the equations and from Figure 6, both the upper bound and approximation of the residual word error rate amount to 0 for a raw bit error rate $\varepsilon = 1$. We point out that this feature is essential for a self-calibrating system that may very well operate with values of ε as high as 1.

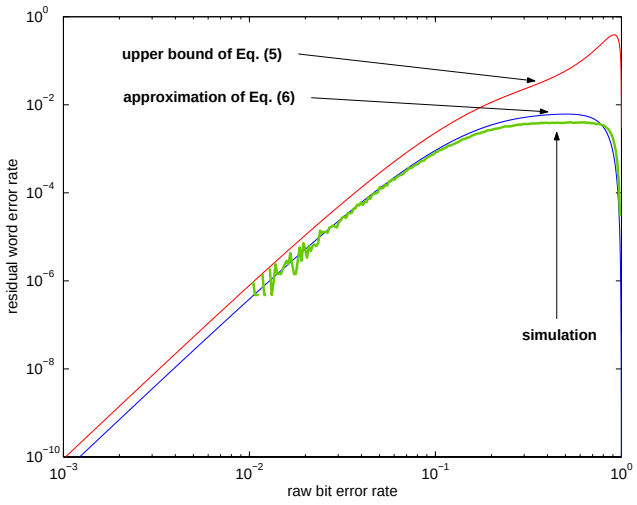

Figure 6. Residual word error rate ($\mathrm{P}_{\text{u.w.e.}}$) vs. raw bit error rate (ε) for the CRC-8 alternating phase encoding over a timing-error channel. The thick curve has been obtained by simulation, while the others are plotted from Eqs. (5) and (6).

Figure 6 shows that the maximum $\mathrm{P}_{\text{u.w.e.}}$ is nearly 10^{-2}. This maximum value can be decreased by exploiting a latency-reliability trade-off. We observe that the residual word error rate is maximum when the raw bit error rate is fairly high (in the 0.1 to 0.9 range). As a result, although it is in this bit error rate range that the maximum undetection probability is found, the code still detects many of the errors that occur. The idea consists in considering correctly received a word only if also the following word has no detected errors. Namely, $\vec{y}_k$ is marked correct only if also $\vec{y}_{k+1}$ has no detected errors. This rule brings the benefit of cancelling undetected errors affecting $\vec{y}_k$ when $\vec{y}_{k+1}$ has detected errors. The rule is easily implemented with a one-word pipeline appended to the decoder output.

We call $\mathrm{P}_{\text{u.e.}}^{\text{pipe}}$ the residual word error rate of this scheme, and give its analytical expression in Property 1.

Property 1 (Residual word error rate) *The residual word error rate of the scheme with a single word pipeline is*

$$\mathrm{P}_{\text{u.e.}}^{\text{pipe}} = \mathrm{P}_{\text{u.w.e.}} \left(1 - \left(1 - \mathrm{P}_{\text{u.w.e.}}\right)\left(1 - \mathrm{P}_{\text{n.w.e.}}\right)\right),$$

with $\mathrm{P}_{\text{n.w.e.}}$ *the probability that no error occurs on a word, and* $\mathrm{P}_{\text{u.w.e.}}$ *the residual word error rate of the encoding used.*

The proof of this property is in appendix. Figure 7 compares the residual word error rate of the one word pipeline scheme with the one already plotted in Figure 6. Figure 7 reveals that the maximum residual word error rate has been decreased by nearly two orders of magnitude. As expected, the difference between the two schemes is only significant for large bit error rates.

TIMING ERRORS AND ADDITIVE NOISE

We have seen in the last section that some low overhead

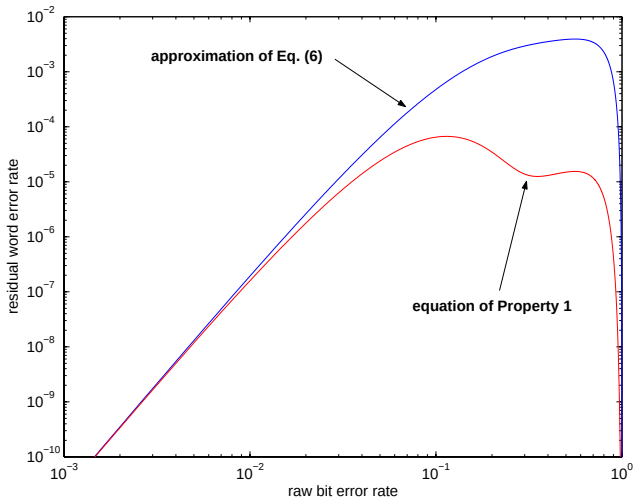

Figure 7. Residual word error rate ($P_{u.w.e.}$) as a function of the raw bit error rate (ε), for the CRC-8 alternating phase encoding over a timing-error channel. The upper curve is obtained from Eq. (6), while the lower one is plotted from Property 1.

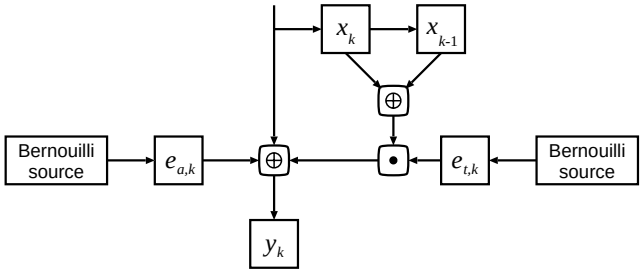

Figure 8. Extension of the timing-error channel: not only bit transitions may cause errors, but also additive white noise.

versions of alternating phase encoding, although not ensuring self-synchronisation in the sense of Def. 2, still offers a reasonable residual word error rate over the whole range of raw bit error rate. Moreover, they detect all errors occurring with a unity raw bit error rate. The situation considered in the last section is in a sense ideal, as it assumes that the only error source are delayed transitions. We extend therefore the timing-error model with additive noise, as depicted in Figure 8. This more realistic channel model now includes two independent white noise sources—modelled by Bernouilli sources as in Eq. (1). We denote such a channel a *Timing-Error Additive-Noise Channel*, and abbreviate it TEANC $(\varepsilon_t, \varepsilon_a)$, with ε_t and ε_a describing the respective parameters of the two independent and i.i.d. Bernouilli sources of timing and additive noise. We compare now the reliability of the CRC-8 alternating phase encoding with the one of LEDR, under a TEANC $(\varepsilon_t, \varepsilon_a)$. Notice that, over such a channel, LEDR and other classic self-synchronising codes do not guarantee anymore the detection of all errors.

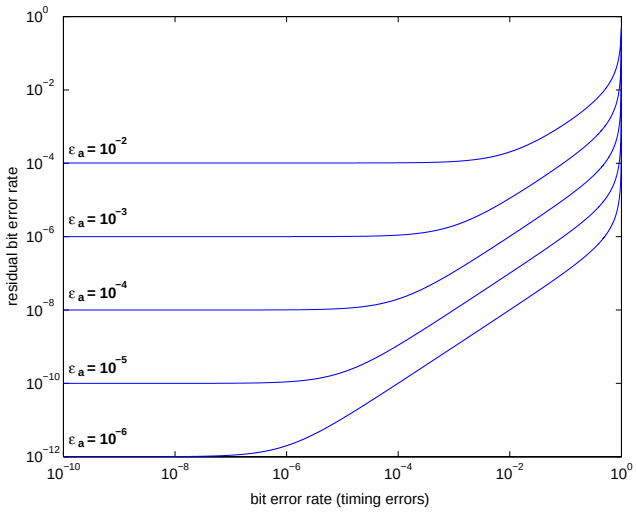

Figure 9. Residual bit error rate ($\varepsilon_{res.}^{bit}$) of LEDR as a function of ε_t and for several values of ε_a, when transferring $K = 20$ information bits over a TEANC $(\varepsilon_t, \varepsilon_a)$ channel. The residual bit error rate goes up to 0.5 as ε_t becomes 1, for any $\varepsilon_a > 0$.

As a reliability metric, we use the residual bit error rate, $\varepsilon_{res.}^{bit}$, because the two codes have different overhead. The residual bit error is the probability that, at the decoder output, a bit is erroneous. It is actually difficult to obtain analytically the residual bit error rate of an alternating phase encoding over a timing-error channel with additive noise. Therefore, we obtain this quantity by simulation. On the contrary, the residual bit error is easily obtained in the special case of LEDR, as shown in the next property, whose proof is in appendix.

Property 2 (Residual bit error rate of LEDR) *Consider the transfer of K information bits using the LEDR code over a timing-error additive-noise channel* TEANC $(\varepsilon_t, \varepsilon_a)$. *Then, the probability of an undetected bit error is independent of K, and is given by*

$$\varepsilon_{res.}^{bit} = \frac{P_{u.b.e.}}{P_{n.b.e.} + P_{u.b.e.}},$$

with $P_{u.b.e.} = \varepsilon_t\varepsilon_a(1-\varepsilon_a) + (1-\varepsilon_t)\varepsilon_a^2$ *and* $P_{n.b.e.} = \varepsilon_t\varepsilon_a \cdot (1-\varepsilon_a) + (1-\varepsilon_t)(1-\varepsilon_a)^2$.

We have plotted in Figure 9 the expression of $\varepsilon_{res.}^{bit}$ as a function of ε_t, for different values of ε_a, and assuming that $K = 20$ information bits are transferred. As expected, we observe a plateau as ε_t goes to zero. On the contrary, we see that the residual bit error rate becomes as high as 0.5, when ε_t goes to 1. Due to additive noise, LEDR has lost the self-synchronising property. We compare now the residual bit error rate of LEDR with the one of the CRC-8 alternating phase encoding. Again, we do so by fixing the additive bit error rate ε_a to some relatively small values, as expected in reality. We perform the comparison at equal wiring resources, that is considering an LEDR code with $K = 20$ information bits (i.e., a total of $2K = 40$ bits) and an alter-

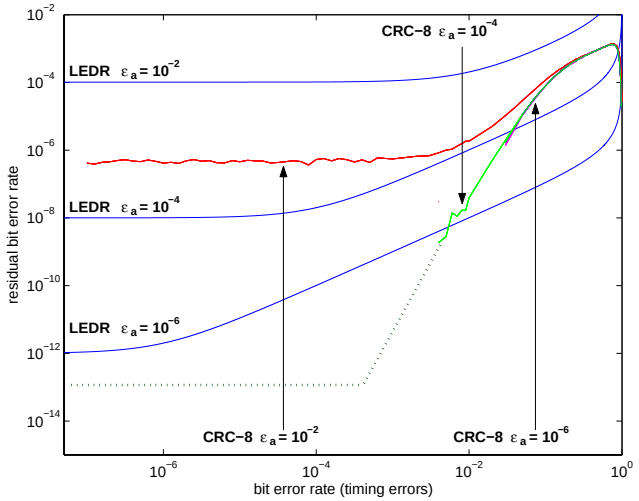

Figure 10. Comparison of the residual bit error rate of LEDR and the CRC-8 alternating phase encoding, as a function of ε_t and for several values of ε_a. The dotted horizontal line is an upper bound on the residual bit error rate of the CRC-8 alternating phase encoding, for $\varepsilon_a = 10^{-4}$ and ε_t close to 0.

nating phase encoding that adds 8 redundant bits to $K = 32$ information bits.

Figure 10 reveals that, over most of the timing bit error rate range, the CRC-8 alternating phase encoding outperforms LEDR. In particular, the CRC-8 alternating phase encoding performs significantly better than LEDR for the two cases (i) ε_t small ($< 10^{-4}$), and (ii) ε_t large (> 0.5). These two cases are especially relevant for self-calibrating systems: the system controller will mostly operate on the low bit error rate range, and, sporadically, will push the system to the high bit error rate range. Indeed, for most silicon-based systems, the transition from correct to wrong behaviour is pretty steep. In conclusion, for self-calibrating communication, the CRC-8 alternating phase encoding looks like a promising choice since (a) it has a similar or better reliability than LEDR and (b) it costs less energy due to both lower wiring overhead and switching activity.

CONCLUSION

In this paper, we study communication over a self-calibrated link. We formally model errors due to over-aggressive setting of the link parameters (voltage and frequency). We show that self-synchronising codes detect all such errors. Then, we contrast two different coding schemes, one used in asynchronous communication (LEDR) and an original one based on well-known linear codes (CRC-8 *alternating phase* encoding). Compared to LEDR, we show that the CRC-8 alternating phase encoding offers a tolerable degradation of reliability, while the wiring overhead is reduced by a factor four (25% vs. 100%). We call these codes *soft self-synchronising codes*. In addition, we describe both schemes under a common coding framework and we extend our channel model to account for additive noise. We show that, with this model, the CRC-8 alternating phase encoding, a low wiring overhead embodiment of alternating phase encoding, offers better or similar reliability than LEDR, while being more energy efficient due to the lower wiring overhead and switching activity. As a result, we suggest that soft self-synchronising codes are promising encoding schemes for self-calibrating communication: they offer a better trade-off between various types of overhead and reliability, compared to traditional hard self-synchronising codes (such as Dual-Rail and LEDR).

REFERENCES

[1] AUSTIN, T., BLAAUW, D., MUDGE, T., AND FLAUTNER, K. Making typical silicon matter with Razor. *Computer 37*, 3 (Mar. 2004), 57–65.

[2] BAICHEVA, T., DODUNEKOV, S., AND KAZAKOV, P. On the cyclic redundancy-check codes with 8-bits redundancy. *Computer Communications 21*, 11 (Aug. 1998), 1030–33.

[3] DEAN, M. E., WILLIAMS, T. E., AND DILL, D. L. Efficient self-timing with level-encoded two-phase dual-rail (LEDR). In *Proceedings of the 1991 University of California/Santa Cruz Conference on Advanced Research in VLSI* (Mar. 1991), MIT Press, pp. 55–70.

[4] HEGDE, R., AND SHANBHAG, N. R. Soft digital signal processing. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems VLSI-9*, 6 (Dec. 2001), 813–23.

[5] MACWILLIAMS, F. J., AND SLOANE, N. J. A. *The Theory of Error-Correcting Codes*, vol. 19 of *Mathematical Library*. North-Holland, Amsterdam, 1977.

[6] TAM, S., RUSU, S., DESAI, U. N., KIM, R., ZHANG, J., AND YOUNG, I. Clock generation and distribution for the first IA-64 microprocessor. *IEEE Journal of Solid-State Circuits 35*, 11 (Nov. 2000), 1545–52.

[7] VARSHAVSKY, V. I., Ed. *Self-Timed Control of Concurrent Processes*. Kluwer Academic, Dordrecht, The Netherlands, 1990.

[8] WORM, F., IENNE, P., AND THIRAN, P. Undetection error probability of linear and alternating-phase codes over a timing error channel. Technical report, Swiss Federal Institute of Technology Lausanne (EPFL), School of Computer and Communication Sciences (I&C), Lausanne, Switzerland, Sept. 2004.

[9] WORM, F., IENNE, P., THIRAN, P., AND DE MICHELI, G. An adaptive low-power transmission scheme for on-chip networks. In *Proceedings of the 15th International Symposium on System Synthesis* (Kyoto, Oct. 2002), pp. 92–100.

[10] WORM, F., IENNE, P., THIRAN, P., AND DE MICHELI, G. A robust self-calibrating transmission scheme for on-chip networks. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems VLSI-12*, 10 (Oct. 2004). To appear.

[11] WORM, F., IENNE, P., THIRAN, P., AND DE MICHELI, G. Self-calibrating communication for on-chip networks. *IEEE Design and Test of Computers* (2004). To appear.

APPENDICES

Proof of Property 1

A word at the channel output has either (i) no error, with probability $\mathrm{P}_{\text{n.w.e.}}$, (ii) an undetected error, with probability $(1-\mathrm{P}_{\text{n.w.e.}})\,\mathrm{P}_{\text{u.w.e.}}$, or (iii) a detected error, with probability $(1-\mathrm{P}_{\text{n.w.e.}})(1-\mathrm{P}_{\text{u.w.e.}})$. We denote by w_k the word at the pipeline output, at time k, by $\mathrm{I}_k^{\text{w.e.}}$ the indicator function of an error on w_k, and by $\mathrm{I}_k^{\text{d.w.e.}}$ the indicator function of a detected error on w_k. We have that

$$\begin{aligned}\mathrm{P}_{\text{u.e.}}^{\text{pipe}} &= P\left(\mathrm{I}_k^{\text{d.w.e.}}=0, \mathrm{I}_{k+1}^{\text{d.w.e.}}=0 \mid \mathrm{I}_k^{\text{w.e.}}=1\right)\\ &= \frac{P\left(\mathrm{I}_k^{\text{d.w.e.}}=0, \mathrm{I}_{k+1}^{\text{d.w.e.}}=0, \mathrm{I}_k^{\text{w.e.}}=1\right)}{P\left(\mathrm{I}_k^{\text{w.e.}}=1\right)}.\end{aligned} \tag{7}$$

We obtain directly that

$$P\left(\mathrm{I}_k^{\text{w.e.}}=1\right) = 1-\mathrm{P}_{\text{n.w.e.}}. \tag{8}$$

Also, the event $\left(\mathrm{I}_k^{\text{d.w.e.}}=0, \mathrm{I}_{k+1}^{\text{d.w.e.}}=0, \mathrm{I}_k^{\text{w.e.}}=1\right)$ is equivalent to requiring that (i) w_k has an undetected error and (ii) w_{k+1} has no detected error. The two events are independent because the timing-error channel introduces independent errors. We can therefore write that

$$\begin{aligned}&P\left(\mathrm{I}_k^{\text{d.w.e.}}=0, \mathrm{I}_{k+1}^{\text{d.w.e.}}=0, \mathrm{I}_k^{\text{w.e.}}=1\right) =\\ &(1-\mathrm{P}_{\text{n.w.e.}})\,\mathrm{P}_{\text{u.w.e.}}\cdot\left(1-(1-\mathrm{P}_{\text{u.w.e.}})(1-\mathrm{P}_{\text{n.w.e.}})\right).\end{aligned} \tag{9}$$

The result is obtained by combining Eqs. (7), (8), and (9). Lastly, remark that, when using a (N,K) linear code over a timing-error channel of bit error rate ε, the probability of no error, $\mathrm{P}_{\text{n.w.e.}}$, is given by

$$\mathrm{P}_{\text{n.w.e.}} = \frac{1}{2^K}\sum_{i=0}^{N} A_i\,(1-\varepsilon)^i,$$

with A_i the number of codewords of weight i in the code. □

Proof of Property 2

We denote by word the $2K$ bits resulting of the encoding of K information bits, and use the same notations of Property 1. We say that an undetected word error has a weight-i when i information bits are affected by the undetected error. We compute $\varepsilon_{\text{res.}}^{\text{bit}}$ by a weighted (with the probability of occurrence) sum of fraction of erroneous bits at the decoder output:

$$\varepsilon_{\text{res.}}^{\text{bit}} = \sum_{i=1}^{K}\frac{i}{K}P\left(\text{weight-}i\text{ u.w.e.} \mid \mathrm{I}^{\text{d.w.e.}}=0\right). \tag{10}$$

Through standard probability manipulations, we have that

$$\begin{aligned}&P\left(\text{weight-}i\text{ u.w.e.} \mid \mathrm{I}^{\text{d.w.e.}}=0\right) =\\ &\frac{P\left(\text{weight-}i\text{ u.w.e.}, \mathrm{I}^{\text{d.w.e.}}=0\right)}{P\left(\mathrm{I}^{\text{d.w.e.}}=0\right)} =\\ &\frac{P\left(\text{weight-}i\text{ u.w.e.}\right)}{P\left(\mathrm{I}^{\text{d.w.e.}}=0\right)}.\end{aligned} \tag{11}$$

Let $\vec{x}_k$ and $\vec{y}_k$ be the input and the output, respectively, of a 2-bits TEANC $(\varepsilon_{\text{t}},\varepsilon_{\text{a}})$ channel. We denote by $\mathrm{P}_{\text{n.b.e.}}$ the probability that no error occurs, i.e., $\mathrm{P}_{\text{n.b.e.}} = P\left(\vec{x}_k=\vec{y}_k\right)$. Moreover, we denote by $\mathrm{P}_{\text{u.b.e.}}$ the probability that an undetected bit error occurs, i.e., $\mathrm{P}_{\text{u.b.e.}} = P\left(\vec{x}_k\neq\vec{y}_k, \mathrm{I}^{\text{d.w.e.}}=0\right)$. Notice that an undetected error implies that the bit output by the decoder is corrupted since no weight-1 error is undetected with LEDR.
We remark that

$$P\left(\text{weight-}i\text{ u.w.e.}\right) = \binom{K}{i}\left(\mathrm{P}_{\text{u.b.e.}}\right)^i\left(\mathrm{P}_{\text{n.b.e.}}\right)^{(K-i)} \tag{12}$$

and

$$P\left(\mathrm{I}^{\text{d.w.e.}}=0\right) = \left(\mathrm{P}_{\text{n.b.e.}}+\mathrm{P}_{\text{u.b.e.}}\right)^K. \tag{13}$$

Combining Eqs. (10), (11), (12), and (13), we obtain that

$$\begin{aligned}\varepsilon_{\text{res.}}^{\text{bit}} &= \sum_{i=1}^{K}\frac{i}{K}P\left(\text{weight-}i\text{ u.w.e.} \mid \mathrm{I}^{\text{d.w.e.}}=0\right)\\ &= \frac{\sum_{i=1}^{K}\frac{i}{K}\binom{K}{i}\left(\mathrm{P}_{\text{u.b.e.}}\right)^i\left(\mathrm{P}_{\text{n.b.e.}}\right)^{K-i}}{\left(\mathrm{P}_{\text{n.b.e.}}+\mathrm{P}_{\text{u.b.e.}}\right)^K}\\ &= \frac{\sum_{i=1}^{K}\binom{K-1}{i-1}\left(\mathrm{P}_{\text{u.b.e.}}\right)^i\left(\mathrm{P}_{\text{n.b.e.}}\right)^{((K-1)-(i-1))}}{\left(\mathrm{P}_{\text{n.b.e.}}+\mathrm{P}_{\text{u.b.e.}}\right)^K}\\ &= \frac{\mathrm{P}_{\text{u.b.e.}}\left(\mathrm{P}_{\text{n.b.e.}}+\mathrm{P}_{\text{u.b.e.}}\right)^{K-1}}{\left(\mathrm{P}_{\text{n.b.e.}}+\mathrm{P}_{\text{u.b.e.}}\right)^K}\\ &= \frac{\mathrm{P}_{\text{u.b.e.}}}{\mathrm{P}_{\text{n.b.e.}}+\mathrm{P}_{\text{u.b.e.}}}.\end{aligned} \tag{14}$$

Expressions for $\mathrm{P}_{\text{u.b.e.}}$ and $\mathrm{P}_{\text{n.b.e.}}$ are easily obtained by recalling that an error is undetected with LEDR if and only if both bits are corrupted. Therefore,

$$\mathrm{P}_{\text{u.b.e.}} = \varepsilon_{\text{t}}\varepsilon_{\text{a}}\left(1-\varepsilon_{\text{a}}\right)+\left(1-\varepsilon_{\text{t}}\right)\varepsilon_{\text{a}}^2. \tag{15}$$

The first term is the probability that, on the transitioning line, a timing error and no additive error occur, while, on the other line, an additive error occurs. The second term is the probability that, on the transitioning line, no timing error, but an additive error occurs, while, on the other line, an additive error occurs. By a similar reasoning, one gets that

$$\mathrm{P}_{\text{n.b.e.}} = \varepsilon_{\text{t}}\varepsilon_{\text{a}}\left(1-\varepsilon_{\text{a}}\right)+\left(1-\varepsilon_{\text{t}}\right)\left(1-\varepsilon_{\text{a}}\right)^2. \tag{16}$$

□