Modeling and Energy Optimization of LDPC Decoder Circuits with Timing Violations

François Leduc-Primeau, Frank R. Kschischang, and Warren J. Gross

Abstract

This paper proposes a quasi-synchronous design approach for signal processing circuits, in which timing violations are permitted, but without the need for a hardware compensation mechanism. The error-correction performance of low-density parity-check (LDPC) code ensembles is evaluated using density evolution while taking into account the effect of timing faults, and a method for accurately modeling the effect of faults at a high level of abstraction is presented. Following this, several quasi-synchronous LDPC decoder circuits are designed based on the offset min-sum algorithm, providing a 23%–40% reduction in energy consumption or energy-delay product, while achieving the same performance and occupying the same area as conventional synchronous circuits.

I. INTRODUCTION

The time required for a signal to propagate through a CMOS circuit varies depending on several factors. Some of the variation results from physical limitations: the delay depends on the initial and final charge state of the circuit. Other variations are due to the difficulty (or impossibility) of controlling the fabrication process and the operating conditions of the circuit [1]. As process technologies approach atomic scales, the magnitude of these variations is increasing, and reducing the supply voltage to save energy increases the variations even further [2].

The variation in propagation delay is a source of energy-inefficiency for synchronous circuits, and new design approaches have been proposed to tackle this. In better than worst-case (BTWC)
[3] or voltage over-scaled (VOS) circuits, efficiency is improved by allowing some timing violations to occur, while including a mechanism to compensate or recover from these faults. One such method consists in introducing special latches that can detect timing violations, and to restart the circuit when a violation is detected [4], [5]. Since the circuit’s latency is increased significantly when a timing violation occurs, this approach is only suitable for tolerating small fault rates (e.g. $10^{-7}$) and for applications where the circuit can be easily restarted, such as speculative microprocessors. In the case of signal processing circuits, we are usually interested in the average quality of the output, which creates more possibilities for dealing with timing violations. A seminal contribution in this area was the algorithmic noise tolerance (ANT) approach [6], [7], which is to allow timing violations to occur in the main processing block, while adding a separate reliable processing block with reduced precision that is used to bound the error of the main block, and provide algorithmic performance guarantees. The downside of the ANT approach is that it relies on the assumption that timing violations will first occur in the most significant bits. If that is not the case, the precision of the circuit could degrade to the precision of the auxiliary block, limiting the scheme’s usefulness. For many circuits, including some adder circuits [8], this assumption does not hold. Furthermore, the addition of the reduced precision block and of a comparison circuit increases the area requirement.

We propose a general design methodology for digital circuits with a relaxed synchronicity requirement that does not rely on any hardware compensation mechanism. Instead, we provide performance guarantees by re-analyzing the algorithm while taking into account the effect of timing violations. We call such circuits quasi-synchronous circuits.

We use the quasi-synchronous approach to design energy-optimized low-density parity-check (LDPC) decoder circuits based on a state-of-the-art soft-input algorithm and architecture. LDPC decoding algorithms are good candidates for a quasi-synchronous implementation because their throughput and energy consumption are limiting factors in many applications, and like other signal processing algorithms, their performance is determined by the average quality of their output. Furthermore, we show that the iterative nature of the decoding algorithm allows for additional energy savings.

The topic of unreliable LDPC decoders has been discussed in a number of contributions. Varshney studied the Gallager-A and the Sum-Product decoding algorithms when the computations and the message exchanges are “noisy”, and showed that the density evolution analysis
still applies [9]. The Gallager-B algorithm was also analyzed under various scenarios [10]–[12]. A model for an unreliable quantized Min-Sum decoder was proposed in [13], which provided numerical evaluation of the density evolution equations as well as simulations of a finite-length decoder. Faulty finite-alphabet decoders were studied in [14], where it was proposed to model the decoder messages using conditional distributions that depend on the ideal messages. The quantized Min-Sum decoder was also analyzed in [15] for the case where faults are the result of storing decoder messages in an unreliable memory. The specific case of faults caused by delay variations in synchronous circuits is considered in [16], where a deviation model is proposed for binary-output circuits in which a deviation occurs probabilistically when the output of a circuit changes from one clock cycle to the next, but cannot occur if the output does not change. While none of these contributions explicitly consider the relationship between the reliability of the decoder’s implementation and the energy it consumes, there have been some recent developments in the analysis of the energy consumption of reliable decoders. Lower bounds for the scaling of the energy consumption of error-correction decoders in terms of the code length are derived in [17], and tighter lower bounds that apply to LDPC decoders are derived in [18]. The power required by regular LDPC decoders is also examined in [19], as part of the study of the total power required for transmitting and decoding the codewords.

In this paper, we present a circuit design framework that captures the effect of timing violations caused by a reduced supply voltage and increased clock frequency, while simultaneously measuring the energy consumption. We show that the information provided by the framework can be used as part of a density evolution analysis to evaluate the channel threshold and iterative performance of the decoder when affected by timing faults. To perform density evolution, we propose an accurate high-level model for the effect of timing faults. Finally, we show that under mild assumptions, the problem of minimizing the energy consumption of a quasi-synchronous decoder can be simplified to the energy minimization of a small test circuit, and present an approximate optimization method similar to Gear-Shift Decoding [20] that finds sequences of quasi-synchronous decoders that minimize decoding energy.

The remainder of the paper is organized as follows. Section II reviews LDPC codes and describes the circuit architecture of the decoder that is used to measure timing faults. Section III presents the deviation model that represents the effect of timing faults on the algorithm. Section IV then discusses the use of density evolution and of the deviation model to predict
the performance of a decoder affected by timing faults. Finally, Section V presents the energy optimization strategy and results, and Section VI concludes the paper. Additional details on the CAD framework used for circuit measurements can be found in Appendix A, and Appendix B provides some details concerning the simulation of the test circuits.

II. LDPC Decoding Algorithm and Architecture

A. Code and Channel

We consider a communication scenario where a sequence of information bits is encoded using a binary LDPC code of length \( n \). The LDPC code is described by an \( m \times n \) binary parity-check matrix \( H \), or equivalently by a factor graph composed of variable nodes indexed from 1 to \( n \) and of check nodes indexed from 1 to \( m \). For the \( H \) matrix and the graph to represent the same code, the graph must contain an edge between a variable node \( i \) and a check node \( j \) if and only if the element of \( H \) at row \( j \) and column \( i \) is non-zero. We assume that the LDPC code is regular, and denote the degree of variable nodes by \( d_v \), and the degree of check nodes by \( d_c \).

Let us assume that the transmission takes place over the Additive White Gaussian Noise (AWGN) channel. A codeword \( x \in \{-1, 1\}^n \) is transmitted through the channel, which outputs the received vector \( y = x + W \), where \( W \) is a vector of \( n \) independent and identically distributed zero-mean normal random variables with variance \( \sigma_w^2 \). The AWGN channel has the property of being output symmetric, meaning that \( P(y_i = q|x_i = 1) = P(y_i = -q|x_i = -1) \).

Let the belief output \( \mu_i \) of the channel be given by

\[
\mu_i = \frac{\alpha y_i}{\sigma_w^2},
\]

with \( \alpha > 0 \). Note that if \( \alpha = 2 \) then \( \mu_i \) is in the log-likelihood ratio format. Assuming that \( x_i = 1 \) was transmitted, then \( \mu_i \) has a normal distribution with mean \( \alpha/\sigma_w^2 \) and variance \( \alpha^2/\sigma_w^2 \), and therefore when \( \alpha \) is fixed, its distribution can be described by a single parameter \( \rho = \alpha/\sigma_w^2 \), yielding \( \mu_i \sim \mathcal{N}(\rho, \alpha \rho) \). We call this distribution a one-dimensional (1-D) normal distribution. The distribution of \( \mu_i \) can also be specified using other equivalent parameters, such as the probability of error \( p_e \), given by

\[
p_e = P(\mu_i < 0|x_i = 1) = P(\mu_i > 0|x_i = -1) = \frac{1}{2} \text{erfc} \left( \frac{1}{\sqrt{2\sigma_w^2}} \right) = \frac{1}{2} \text{erfc} \left( \sqrt{\frac{\rho}{2\alpha}} \right),
\]

where \( \text{erfc}(\cdot) \) is the complementary error function.
B. Decoding Algorithm

The well-known Offset Min-Sum (OMS) algorithm is a simplified version of the Sum-Product algorithm that can usually achieve similar error-correction performance. It has been widely used in implementations of LDPC decoders [21]–[23]. To make our decoder implementation more realistic and show the flexibility of our design framework, we present an algorithm and architecture that support a row-layered message-passing schedule. Architectures optimized for this schedule have proven effective for achieving efficient implementations of LDPC decoders [22]–[24]. Using a row-layered schedule also allows to pipeline the decoder to increase the circuit’s utilization. In a row-layered LDPC decoder, the rows of the parity-check matrix are partitioned into \( L \) sets called layers. We assume that all the columns in a given layer contain at most one non-zero element, which enables some architectural simplifications. Note that as a result, \( L \geq d_v \).

Using the equivalent graph representation of the code, a layer with index \( \ell \) can be defined as a set \( \mathcal{L}_\ell \) of check nodes, \( \ell \in [1, L] \). In a given iteration, each variable node (VN) receives at most one message for each layer, and the inputs of a given VN can share the same index variable as the layers. Let \( \mu_i^{(t,\ell)} \) be the message sent by a VN \( i \) to its neighboring check node in layer \( \ell \) during iteration \( t \), and \( \lambda_i^{(t,\ell)} \) be the belief message received by VN \( i \) after evaluating layer \( \ell \) during iteration \( t \). Note that messages sent from variable nodes can also be uniquely identified by specifying the iteration index \( t \), the VN index \( i \), and the index \( j \) of the check node receiving the message, leading to the notation \( \mu_i^{(t)} \). We use the second notation when we want to avoid specifying the type of message-passing schedule being used. Finally, we denote the channel information corresponding to the \( i \)-th codeword bit by \( \mu_i^{(0)} \), since it also corresponds to the first message sent by a variable node \( i \) to all its neighboring check nodes.

The Offset Min-Sum algorithm used with a row-layered message-passing schedule is described in Algorithm 1. In the algorithm, \( v_i \) represents the VN with index \( i \), \( \mathcal{N}(c) \) denotes the set of VNs that are neighbors of a check node \( c \), and \( \Lambda_i^{(t,\ell)} \) represents the current sum of incoming messages at a VN \( i \). At all times we have that

\[
\Lambda_i^{(t,\ell)} = \mu_i^{(0)} + \sum_{\ell' = 1}^{\ell} \lambda_i^{(t,\ell')}, \quad 1 \leq \ell' \leq L.
\]

(3)

The check node (CN) function \( \text{CHK}(S) \) is defined as follows. Define a set \( S \) of VN-to-CN
Algorithm 1: OMS with a row-layered schedule.

input : \{\mu_1^{(0)}, \mu_2^{(0)}, \ldots, \mu_n^{(0)}\}, T
output: \{\hat{x}_1, \hat{x}_2, \ldots, \hat{x}_n\}

1 begin

// Initialization
2 \Lambda_i^{(0,0)} \leftarrow \mu_i^{(0)}, \forall i \in [1,n]
3 \lambda_i^{(0,\ell)} \leftarrow 0, \forall i \in [1,n], \ell \in [1,L]

// Decoding
4 for \( t \leftarrow 1 \) to \( T \) do
5  for \( \ell \leftarrow 1 \) to \( L \) do
6      // VN to CN messages
7      if \( \ell = 1 \) then
8          \mu_i^{(t,\ell)} \leftarrow \Lambda_i^{(t-1,L)} - \lambda_i^{(t-1,\ell)}, \forall i
9      else
10         \mu_i^{(t,\ell)} \leftarrow \Lambda_i^{(t-1,\ell)} - \lambda_i^{(t-1,\ell)}, \forall i
11      // CN to VN messages
12      for \( c \in \mathcal{L}_\ell \) do
13         for \( i \in \{i : v_i \in \mathcal{N}(c)\} \) do
14            \lambda_i^{t,\ell} \leftarrow \text{CHK}\left(\{\mu_k^{(t,\ell)} : v_k \in \mathcal{N}(c) \setminus \{v_i\}\}\right)
15      // VN update
16      \Lambda_i^{(t,\ell)} \leftarrow \mu_i^{(t,\ell)} + \lambda_i^{(t,\ell)}, \forall i
17      // VN decision
18      for \( i \in \{1, 2, \ldots, n\} \) do
19         if \( \Lambda_i^{(t,\ell)} > 0 \) then \( \hat{x}_i \leftarrow 1 \) else if \( \Lambda_i^{(t,\ell)} < 0 \) then \( \hat{x}_i \leftarrow -1 \) else \( \hat{x}_i \leftarrow 1 \) or \( -1 \) with equal probability

messages as \( S = \{\mu_1, \mu_2, \ldots, \mu_{d_c-1}\} \). Then

\[
\text{CHK}(S) = \max\left(0, \min_{\mu_i \in S} \left|\mu_i\right| - C\right) \cdot \prod_{\mu_i \in S} \text{sign}(\mu_i),
\] (4)

where \( C \geq 0 \) is the offset parameter, and

\[
\text{sign}(x) = \begin{cases} 
1 & \text{if } x \geq 0, \\
-1 & \text{if } x < 0.
\end{cases}
\]
C. Architecture

The factor graph of the code can also be used to represent the computations that must be performed by the decoder. At each decoding iteration, one message is sent from variable to check nodes on every edge of the graph, and again from check to variable nodes. We call a variable node processor (VNP) a circuit block that is responsible for generating messages sent by a given variable node, and similarly a check node processor (CNP) a circuit block generating messages sent by a given check node.

In a row-layered architecture in which the column weight of layer subsets is at most 1, there is at most one message to be sent and received for each variable node in a given layer. Therefore VNPs are responsible for sending and receiving one message per clock cycle. CNPs on the other hand receive and send $d_c$ messages per clock cycle. At any given time, every VNP and CNP is mapped respectively to a VN and a CN in the factor graph. The routing of messages from VNPs to CNPs and back can be posed as two equivalent problems. One can fix the mapping of VNs to VNPs and of CNs to CNPs, and find a permutation of the message sequence that matches VNP outputs to CNP inputs, and another permutation that matches CNP outputs to VNP inputs. Alternatively, if VNPs process only one message at a time, one can fix the connections between VNPs and CNPs, and choose the assignment of VN to VNPs to achieve correct message routing. We choose the later approach because it allows studying the computation circuit without being concerned by the routing of messages.

The number of CNPs instantiated in the decoder can be adjusted based on throughput requirements from 1 to $m/L$ (the number of rows in a layer). As the number of CNPs is varied, the number of VNPs will vary from $d_c$ to $n$. An architecture diagram showing one VNP and one CNP is shown in Fig. 1. In reality, a CNP is connected to $d_c - 1$ additional VNPs, which are not shown. The memories storing the belief totals $\Lambda_i^{(t,\ell)}$ and the intrinsic beliefs $\lambda_i^{(t,\ell)}$ are also not shown. The part of the VNP responsible for sending a message to the CNP is called VNP front and the part responsible for processing a message received from a CNP is called the VNP back. The VNP front and back do not have to be simultaneously mapped to the same VN. This allows to easily vary the number of pipeline stages in the VNPs and CNPs. Fig. 1 shows the circuit with two pipeline stages.

Messages exchanged in the decoder are fixed-point numbers. The position of the binary point
does not have an impact on the algorithm, and therefore the messages sent by VNs in the first iteration can be defined as rounding the result of (1) to the nearest integer, while choosing a suitable $\alpha$. The number of bits in the quantization, the scaling factor $\alpha$, and the OMS offset parameter are chosen based on a density evolution analysis of the reliable algorithm (described in Section IV). We quantize decoder messages on 6 bits, which allows a reliable decoder to have approximately the same channel threshold as a floating-point decoder.

In order to analyze a circuit that is representative of state-of-the-art architectures, we use an optimized architecture for finding the first two minima in each CNP. Our architecture is inspired by the “tree structure” approach presented in [25], but requires fewer comparators. Each pair of CNP inputs is first sorted using the Sort block shown in Fig. 2a. These sorted pairs are then merged recursively using a tree of Merge blocks, shown in Fig. 2b. If the number of CNP inputs is odd, the input that cannot be paired is fed directly into a special merge block with 3 inputs, which can be obtained from the 4-input Merge block by removing the $\min_{2b}$ input and the bottom multiplexer.

Note that it is possible that changes to the architecture could increase or decrease the robustness of the decoder (see e.g. [26]), but this is outside the scope of this paper.
III. DEVIATION MODEL

A. Quasi-Synchronous Circuit Design

We consider synchronous circuits that permit timing violations without hardware compensation, resulting in what we call quasi-synchronous circuits. Optimizing the energy consumption of these circuits requires an accurate model of the impact of timing violations, and of the energy consumption. We propose to achieve this by characterizing one or several test circuits that are subsets of the complete circuit implementation.

The term deviation refers to the effect of circuit faults on the result of a computation, and the deviation model is the bridge between the circuit characterization and the analysis of the algorithm. We reserve the term error for describing the algorithm, in the present case to refer to the incorrect detection of a transmitted symbol. We are interested in modeling deviations occurring in a synchronous circuit, and therefore the computation can be represented as a discrete-time system. In a deviation-free context, the circuit accepts an input $X[t]$ at time $t \in \{0, 1, 2, \ldots\}$ and outputs a result $Y[t]$. Note that the circuit could require one or several clock cycles to generate $Y[t]$, but this is irrelevant to the characterization of the computation. When operated in a quasi-synchronous manner, the circuit instead outputs $Z[t]$, which is a corrupted version of $Y[t]$. Note that it is important to distinguish the sequence of inputs seen by a particular processing circuit from the sequence of computations that must be performed. For example, one LDPC decoding iteration requires evaluating the outputs of $m$ check nodes, but the number of processing circuits used for this task is often much less than $m$. In order to preserve the distinction between the algorithm and circuit abstraction levels, it is convenient to assume that the sequence of algorithm-level inputs is an independent and identically distributed (i.i.d.) process, so that $X[t]$ is also an
i.i.d. process with the same distribution, regardless of the number of processing circuits used. To alleviate the restriction imposed by this assumption, we can characterize the circuit for several distributions of $X[t]$.

To simplify the analysis of the algorithm (and to preserve the algorithm-level abstraction), it is also convenient to model deviations on $Y[t]$ as a transmission through a memoryless communication channel, where the deviation $D[t]$ corresponds to additive noise, such that $Z[t] = Y[t] + D[t]$. However, it is challenging to generate simple and accurate descriptions of the deviation $D[t]$. The main difficulty is that the propagation delay through the circuit depends on its internal charge state and therefore on previous inputs. In Section III-B, we present a memoryless deviation model that is sufficiently accurate to predict the behavior of the decoder for cases of interest.

The test circuit is used to characterize not only deviations, but also energy consumption. Both deviations and energy depend on the choice of some design parameters. Let $\Gamma$ be the space of design parameters that we are considering. For a probability distribution of $X[t]$ parametrized by $\pi$ and system parameters $\gamma \in \Gamma$, we define a function $f_\gamma(\pi)$ that provides a parametrization of the distribution of $Z[t]$, and a function $c_\gamma(\pi)$ that provides the average energy consumed by the circuit to produce one output.

### B. Deviation Model

In LDPC decoding, the smallest unit of computation consists in evaluating the one-iteration computation tree, shown within the dashed box in Fig. 3, in the sense that the state of the decoder remains unchanged unless all the computations required to update the computation tree are performed. Furthermore, all the computations performed in an LDPC decoder can be
represented in terms of this computation tree, and therefore the computation tree constitutes a
good basis for the test circuit, on which deviations will be measured.

There are \((d_v - 1)\) check nodes in the tree. Each of these check nodes receives \((d_c - 1)\)
messages from neighboring variable nodes, and generates a message sent to the one VN whose
message was excluded from the computation. This VN then generates the ideal extrinsic message
based on the messages received from neighboring check nodes and on the channel prior \(\mu^{(0)}\). Let
\(\nu^{(t+1)}_{i,j}\) be the message that would be sent from variable node \(i\) to check node \(j\) during iteration
\(t + 1\) if no deviations had occurred during iteration \(t\).

As discussed in Section III-A, we propose to model all the deviations that have occurred during
the computation of \(\nu^{(t+1)}_{i,j}\) by considering that the ideal output of the computation is transmitted
through an additive-noise communication channel. This is shown in Fig. 3, where a deviation
\(D^{(t)}_{i,j}\) is added to the output of iteration \(t\). For the first messages sent in the decoder at \(t = 0\),
deviations cannot occur, and therefore we simply have \(\mu^{(0)}_{i,j} = \nu^{(0)}_{i,j}\). Note that in the proposed
model, \(D^{(t)}_{i,j}\) is not independent from \(\nu^{(t+1)}_{i,j}\).

We now want to define the statistical properties of the deviation channel to represent the
behavior of the circuit as accurately as possible while retaining a memoryless channel. We
know that deviations caused by timing violations depend on the current and past inputs to a
combinational circuit. A first step in simplifying the model would be to instead assume that the
development depends on the current and past outputs of the circuit. However, at the algorithm level,
the mapping of circuit to computations is not known, and therefore it is difficult to know what the
previous output of the circuit was. It is however always known what the current ideal output is,
and therefore it is possible to construct a deviation model that depends on \(\nu^{(t+1)}_{i,j}\). This was done
in [14], where deviations are modeled by specifying the conditional probability distribution of
\(\mu^{(t+1)}_{i,j}\) given \(\nu^{(t+1)}_{i,j}\). However, such a model does not allow capturing the dependency of deviations
on the previous states of the circuit. To improve the memoryless approximation, we propose to
have it also depend on other quantities that are available at the algorithm level and that provide
information on the statistics of state transitions in the circuit. For this, we use the value of the
transmitted bit \(x_i\), and the message error probability at the beginning of the iteration \(p_e^{(t)}\), defined
as
\[
p_e^{(t)} = \mathbb{P} \left( \mu^{(t)}_{i,j} < 0 \mid x_i = 1 \right) + (1/2) \cdot \mathbb{P} \left( \mu^{(t)}_{i,j} = 0 \right).
\]

\(5\)
The deviation model is therefore described by a family of conditional distributions, indexed by \( (\gamma, p_e^{(t-1)}) \):

\[
P(\gamma, p_e^{(t-1)})_{\mu_i^{(t-1)}, \nu_i^{(t)}, x_i}(\mu | \nu, x_i).
\] (6)

Note that the message error probability parameter is used to describe the circuit’s input distribution, which is why its iteration index is \( t - 1 \). We omit the parameter \( \gamma \) when it is fixed to simplify the notation.

IV. PERFORMANCE ANALYSIS

A. Standard Analysis Methods for LDPC Decoders

Density evolution (DE) is the most common tool used for predicting the error-correction performance of an LDPC decoder. The analysis relies on the assumption that messages passed in the factor graph are mutually independent, which holds as the code length goes to infinity [27]. Given the channel output probability distribution and the probability distribution of variable node to check node messages at the start of an iteration, DE computes the updated distribution of variable node to check node messages at the end of the decoding iteration. This computation can be performed iteratively to determine the message distribution after any number of decoding iterations. The validity of the analysis rests on two properties of the LDPC decoder. The first property is the conditional independence of errors, which states that the error-correction performance of the decoder is independent from the particular codeword that was transmitted. The second property states that the error-correction performance of a particular LDPC code concentrates around the performance measured on a cycle-free graph, as the code length goes to infinity.

Both properties were shown to hold in the context of reliable implementations [27]. It was also shown that the conditional independence of errors always holds when the channel is output symmetric and the decoder has a symmetry property. We can define a sufficient symmetry property of the decoder in terms of a message-update function \( F_{i,j} \) that represents one complete iteration of the (ideal) decoding algorithm. Given a vector of all the messages \( \mu^{(t)} \) sent from variable nodes to check nodes at the start of iteration \( t \) and the channel information \( \nu_i^{(0)} \) associated with variable node \( i \), \( F_{i,j} \) returns the next ideal message to be sent from a variable node \( i \) to a check node \( j \): \( \nu_{i,j}^{(t+1)} = F_{i,j}(\mu^{(t)}, \nu_i^{(0)}) \).
Definition 1. A message-update function \( F_{i,j} \) is said to be symmetric with respect to a code \( C \) if
\[
F_{i,j} (\mu^{(t)}, \nu^{(0)}_i) = x_i F_{i,j} (x\mu^{(t)}, x_i\nu^{(0)}_i)
\]
for any \( \mu^{(t)} \), any \( \nu^{(0)}_i \), and any codeword \( x \in C \).

In other words, a decoder’s message-update function is symmetric if multiplying all the belief messages in the decoder by a valid codeword \( x \) only affects the sign of the VN-to-CN messages, and if additionally the sign of a message sent by a VN \( i \) is flipped if and only if \( x_i = -1 \). Note that the symmetry condition in Definition 1 is implied by the check node and variable node symmetry conditions in [27, Def. 1].

B. Experiments on Codeword Independence

As was discussed earlier, the behavior of a quasi-synchronous circuit depends on the particular sequence of inputs that are processed at each clock cycle. It follows that the performance of a quasi-synchronous decoder is not independent of the transmitted codeword. Nonetheless, since the performance of a reliable decoder is independent of the transmitted codeword, the effect of the transmitted codeword on performance could be negligible in practice, especially when the amount of deviation is moderate.

To evaluate the importance of the transmitted codeword on performance, we perform DE experiments on the test circuit using Monte-Carlo (MC) simulation. For every DE iteration, we start from the known message distribution at the end of the previous iteration, and determine the next message distribution using MC simulation. This evaluation does not depend on a deviation model, and can be used as a benchmark. To evaluate the impact of the transmitted codeword on the performance of the faulty decoder, we perform three MC-DE experiments, one with the all-one codeword, one where the transmitted bits alternate between \( x_i = 1 \) and \( x_i = -1 \), and one with random codewords. The CAD workflow used to perform these measurements is described in Appendix A, and additional details on the MC simulation setup are provided in Appendix B.

We consider two cases that have large average deviation rates, but that are able to reach a low error probability. Each DE iteration is based on a MC simulation using \( 3 \cdot 10^7 \) decoding-iteration trials. In the first test case, shown in Fig. 4, we consider a decoder with \( d_v = 3 \) and \( d_c = 30 \), \( p_e^{(0)} = 0.015 \), operated at \( V_{dd} = 0.75 \) V and \( T_{clk} = 3.2 \) ns. Despite being affected by an average
Fig. 4. MC-DE performance evaluation of a (3, 30) decoder with $p_e^{(0)} = 0.015$ operated at $\gamma = [0.75 \text{V}, 3.2 \text{ns}]$, along with the DE prediction based on the deviation model.

deviation rate of up to 4% (depending on the message distribution), we see that the decoding performance is very similar for all three experiments. In the second test case, shown in Fig. 5, we consider a decoder with $d_v = 3$ and $d_c = 6$, with a channel error probability $p_e^{(0)} = 0.12$, operated at $V_{dd} = 0.85 \text{V}$ and $T_{clk} = 2.1 \text{ns}$. The average deviation rate observed is up to 0.8%. In this case, while the performance obtained for the alternating codeword and for random codewords are very similar, the performance of the all-one codewords is slightly different, with the decoding of the all-one codeword requiring approximately one additional iteration to reach a similar error rate. However, if we reduce the amount of deviations by changing the operating condition to $(V_{dd} = 0.85 \text{V}, T_{clk} = 2.2 \text{ns})$, bringing the maximum average deviation rate to 0.1%, the three results become similar, as shown in Fig. 6.

The experiments above provide some justification for modeling the decoding performance using a codeword-independent model when the deviation rate is not too large. Fortunately, as will be presented in Section V, it turns out that operating the decoder with very large deviation rates is not helpful for optimizing the energy consumption of a quasi-synchronous decoder. In particular, the $(0.85 \text{V}, 2.1 \text{ns})$ operating condition does not appear amongst the energy minimization solutions presented in Section V for (3,6) decoders with $p_e^{(0)} = 0.12$. Of course these experiments do not guarantee that there are not some codewords other than the ones that were simulated for which decoding performance differs. In future work, it would be
interesting to determine whether concentration results can be established for the performance of quasi-synchronous decoders.

C. Applicability of Density Evolution

In order to use density evolution to predict the performance of long finite-length codes, the decoder must satisfy the two properties stated in Section IV-A, namely the conditional indepen-

dence of errors and the convergence to the cycle-free case. We first present some properties of the decoding algorithm and of the deviation model that are sufficient to ensure the conditional independence of errors.

Let \( x \in \{-1, 1\}^n \) be the transmitted codeword, and let \( y \) denote the channel output. The AWGN channel can be modeled multiplicatively as \( y_k = x_k z_k \) \((k \in [1,n])\), where \( \{z_k\} \) is a sequence of i.i.d. random variables that represent the channel noise. In a reliable decoder, messages are completely determined by the received vector \( x z \), but in a faulty decoder, there is additional randomness that results from the deviations. Therefore, we represent messages in terms of conditional probability distributions given \( x z \). Since we are concerned with a fixed-point circuit implementation of the decoder, we can assume that messages are integers from the set \( \{-Q, -Q + 1, \ldots, Q - 1, Q\} \), where \( Q > 0 \) is the largest message magnitude that can be represented.

**Definition 2.** We say that a message distribution \( P_{\mu^{(t)}}(\mu | x z) \) is symmetric if

\[
P_{\mu^{(t)}}(\mu | x z) = P_{\mu^{(t)}}(x_i \mu | z).
\]

To maintain symmetry, let us define the probability that a message \( \mu^{(t)} \) is in error as

\[
P(\mu^{(t)} | x z) = (1/2) \cdot P(\mu^{(t)} = 0).
\]

Note that these comparisons are never actually performed by the decoding algorithm. Then, if a message has a symmetric distribution, its error probability is the same whether \( x z \) or \( z \) is received, since \( y = z \) implies that \( x_i = 1 \).

Similarly to the results presented in [14], we can show that the symmetry of message distributions is preserved when the message-update function is symmetric.

**Lemma 1.** If \( F_{i,j} \) is a symmetric message-update function and if \( \mu_i^{(0)} \) and \( \mu_{i,j}^{(t)} \) have symmetric distributions for all \((i,j)\), the next ideal messages \( \nu_{i,j}^{(t+1)} \) also have symmetric distributions.

**Proof:** We can express the distribution of the next ideal message from VN \( i \) to CN \( j \) as

\[
P_{\nu_{i,j}^{(t+1)}}(\nu | x z) = \sum_{(\mu, \mu_i^{(0)}) \in R} P_{\mu^{(t)}}(\mu | x z) P_{\mu_i^{(0)}}(\mu_i^{(0)} | x z),
\]

where \( R = \{ (\mu, \mu_i^{(0)}) : F_{i,j}(\mu, \mu_i^{(0)}) = \nu \} \).

Since the graph is cycle-free and all \( \mu_{i,j}^{(t)} \) have a symmetric distribution,

\[
P_{\mu^{(t)}}(\mu | x z) = \prod_k P_{\mu_{k}^{(t)}}(\mu_k | x z) = \prod_k P_{\mu_{k}^{(t)}}(x k \mu_k | z) = P_{\mu^{(t)}}(x \mu | z),
\]
and since the channel output $\mu_i^{(0)}$ also has a symmetric distribution,

$$
\mathbb{P}_{\mu_i^{(0)}|y}(\mu_i^{(0)} | xz) = \mathbb{P}_{\mu_i^{(0)}|y}(x_i\mu_i^{(0)} | z).
$$

Therefore, we can rewrite (7) as

$$
\mathbb{P}_{\nu_i^{(t+1)}|y}(\nu | xz) = \sum_{(\mu,\mu_i^{(0)}) \in R} \mathbb{P}_{\mu^{(t)}|y}(\mu | xz) \mathbb{P}_{\mu_i^{(0)}|y}(x_i\mu_i^{(0)} | z).
$$

(8)

Finally, letting $\mu' = x\mu^{(t)}$ and $\nu'_i = x_i\mu_i^{(0)}$, (8) becomes

$$
\mathbb{P}_{\nu_i^{(t+1)}|y}(\nu | xz) = \sum_{(\mu',\nu'_i) \in R'} \mathbb{P}_{\mu^{(t)}|y}(\mu' | xz) \mathbb{P}_{\mu_i^{(0)}|y}(\nu'_i | z),
$$

where $R' = \{(\mu',\nu'_i) : F_{i,j}(x\mu', x_i\nu'_i) = \nu\}$. Since $F_{i,j}$ is symmetric, we can also express $R'$ as

$$
R' = \{(\mu',\nu'_i) : F_{i,j}(\mu', \nu'_i) = x_i\nu\},
$$

and therefore,

$$
\mathbb{P}_{\nu_i^{(t+1)}|y}(x_i\nu | xz) = \sum_{(\mu',\nu'_i) \in R'} \mathbb{P}_{\mu^{(t)}|y}(\mu' | xz) \mathbb{P}_{\mu_i^{(0)}|y}(\nu'_i | x_i\nu | xz),
$$

indicating that the next ideal messages have symmetric distributions.

To establish the conditional independence of errors under the proposed deviation model, we first define some properties of the deviation.

**Definition 3.** We say that the deviation model is symmetric if

$$
\mathbb{P}_{\mu_i^{(t)}|\nu_i^{(t)},y}(\mu | \nu, xz) = \mathbb{P}_{\mu_i^{(t)}|\nu_i^{(t)},y}(\mu | \nu, z) = \mathbb{P}_{\mu_i^{(t)}|\nu_i^{(t)},y}(\mu | -\nu, z).
$$

**Definition 4.** We say that the deviation model is weakly symmetric (WS) if

$$
\mathbb{P}_{\mu_i^{(t)}|\nu_i^{(t)},y}(\mu | \nu, xz) = \mathbb{P}_{\mu_i^{(t)}|\nu_i^{(t)},y}(x_i\mu | x_i\nu, z).
$$

Note that if the model satisfies the symmetry condition, it also satisfies the weak symmetry condition, since $x_i \in \{-1,1\}$. We then have the following Lemma.

**Lemma 2.** If a decoder having a symmetric message-update function and taking its inputs from an output-symmetric communication channel is affected by weakly symmetric deviations, its message error probability at any iteration $t \geq 0$ is independent of the transmitted codeword.
Proof: Similarly to the approach used in [28, Lemma 4.90] and [9], we want to show that the probability that messages are in error is the same whether $xz$ or $z$ is received. This is the case if the faulty messages $\mu_{i,j}^{(t)}$ have a symmetric distribution for all $t \geq 0$ and all $(i,j)$.

Since the communication channel is output symmetric and since no deviations can occur before the first iteration, messages $\mu_{i,j}^{(0)} = \nu_{i,j}^{(0)}$ have a symmetric distribution. We proceed by induction to establish the symmetry of the messages for $t > 0$. We start by assuming that

\[ P_{\nu_{i,j}^{(t)}}(\nu | \mid xz) = P_{\nu_{i,j}^{(t)}}(x_i \nu | z) \]  

(9)

also holds for $t > 0$.

Using Definition 4 and (9), we can write the faulty message distribution as

\[
P_{\mu_{i,j}^{(t)}}(\mu | \mid xz) = \sum_{\nu=-Q}^{Q} P_{\mu_{i,j}^{(t)}}(\mu | \nu, xz) P_{\nu_{i,j}^{(t)}}(\nu | \mid xz)
\]

\[
= \sum_{\nu=-Q}^{Q} P_{\mu_{i,j}^{(t)}}(x_i \mu | x_i \nu, z) P_{\nu_{i,j}^{(t)}}(x_i \nu | z)
\]

\[
= \sum_{\nu'=-x_iQ}^{x_iQ} P_{\mu_{i,j}^{(t)}}(x_i \mu | \nu', z) P_{\nu_{i,j}^{(t)}}(\nu' | z)
\]

\[
= \sum_{\nu'=-Q}^{Q} P_{\mu_{i,j}^{(t)}}(x_i \mu | \nu', z) P_{\nu_{i,j}^{(t)}}(\nu' | z)
\]

\[
= P_{\mu_{i,j}^{(t)}}(x_i \mu | z).
\]

where the third equality is obtained using the substitution $\nu' = x_i \nu$. We conclude that the faulty messages have a symmetric distribution. Finally, since the decoder’s message-update function is symmetric, Lemma 1 confirms the induction hypothesis in (9).

The last remaining step in establishing whether density evolution can be used with a decoder affected by WS deviations is to determine whether the error-correction performance of a code concentrates around the cycle-free case. The property has been shown to hold in [9] (Theorems 2, 3 and 4) for an LDPC decoder affected by “wire noise” and “computation noise”. The wire noise model is similar to our deviation model, in the sense that the messages are passed through an additive noise channel, and that the noise applied to one message is independent of the noise applied to other messages. The proof presented in [9] only relies on the fact that the wire noise applied to a given message can only affect messages that are included in the directed
neighborhood of the edge where it is applied, where the graph direction refers to the direction of message propagation. This clearly also holds in the case of our deviation model, and therefore the proof is the same.

Since the message error probability is independent of the transmitted codeword, and furthermore concentrates around the cycle-free case, density evolution can be used to determine the error-correction performance of a decoder perturbed by our deviation model, as long as the deviations are weakly symmetric.

D. Generating the WS Deviation Model

We have established that it is reasonable to model the performance of a quasi-synchronous decoder as being independent of the transmitted codeword when the deviation probability is not too large, and we have identified properties of the deviation model that ensure codeword independence. We now present a fast method for generating a WS deviation model, and show that the resulting model accurately predicts the behavior of the faulty circuit.

One possibility for measuring the conditional deviation distributions is to simply perform a MC-DE experiment (as in Section IV-B) and observe the deviations. However, MC-DE experiments performed on the circuit models are very computationally intensive. When analyzing reliable decoders, it is common to use a simplified DE that tracks a one-dimensional characterization of the message distribution, rather than the exact distribution. This approach is often called an “extrinsic information transfer” (ExIT) chart. An ExIT function is a 1-D function that provides the message distribution parameter at the end of a decoding iteration, given the message distribution parameter at the beginning of the iteration. One important advantage of using an ExIT chart is that progress made by the decoder at any decoding iteration can be determined without first evaluating the previous decoding iterations. Therefore, it is possible to measure only a few points on the ExIT function, and interpolate between these points to determine the progress made by the decoder at any iteration.

As mentioned in Section II-A, the belief output of an AWGN channel can be characterized exactly by a 1-D normal distribution. In a reliable decoder, a 1-D normal distribution remains an accurate approximation of the message distribution throughout the decoding, because the variance-to-mean ratio of messages remains constant [29]. However, this is not necessarily the case in a faulty decoder. Therefore, we propose to use a 1-D message distribution for the purpose
of generating the deviation model, but to evaluate the decoder’s progress using an exact DE method. We parametrize the 1-D message distribution using the error probability parameter $p_e^{(t-1)}$, yielding a deviation model that takes the form of the conditional distributions already introduced in (6).

We collect deviation measurements from the test circuits by inputting test vectors representing random codewords, and distributed according to several $p_e^{(t-1)}$ values. We then generate estimates of the conditional distributions in (6). It is interesting to visualize the distributions using an aggregate measure such as the probability of observing a non-zero deviation

$$p_{nz}(\nu_{i,j}^{(t)}, x_i) = \mathbb{P}(p_e^{(t-1)}) (\mu_{i,j}^{(t)} \neq \nu_{i,j}^{(t)}) | \nu_{i,j}^{(t)}, x_i) .$$

(10)

These conditional probabilities are shown for a $(3,30)$ circuit in Fig. 7. When $x_i = 1$, positive belief values indicate a correct decision, whereas when $x_i = -1$, negative belief values indicate a correct decision. We can see that in this example, deviations are more likely when the belief is incorrect than when it is correct, and therefore a symmetric deviation model is not consistent with these measurements. On the other hand, there is a sign symmetry between the “correct” part of the curves, and between the “incorrect” parts, that is $p_{nz}(\nu_{i,j}^{(t)}, 1) = p_{nz}(-\nu_{i,j}^{(t)}, -1)$, and for this reason a weakly symmetric model is consistent with the measurements. Note that the slight jaggedness observed for incorrect belief values of large magnitude in the $p_e^{(t-1)} = 0.008$ curves is due to the fact that these $\nu_{i,j}$ values occur only rarely. For the largest incorrect $\nu_{i,j}$ values, only about 100 deviation events are observed for each point, despite the large number of MC trials.

Figure 8 shows a similar plot for a $(3,6)$ circuit. In this case, the conditional deviation probability is almost symmetric, and a symmetric deviation model could be appropriate. Of course, since it is more general, a WS model is also appropriate.

Under the assumption that deviations are weakly symmetric, we have

$$\mathbb{P}(p_e^{(t-1)}) (\mu | \nu, 1) = \mathbb{P}(p_e^{(t-1)}) (\mu | -\nu, -1) .$$

Therefore, we can combine the $x_i = 1$ and $x_i = -1$ data to improve the accuracy of the estimated distributions.

To validate the accuracy of the WS deviation model, we compare the performance predicted by DE based on the model against MC-DE simulations performed on the circuit. Examples are
shown in Figures 4, 5, and 6. We can see that despite the use of a memoryless model and despite the 1-D approximation used when measuring deviations, the performance predicted using the WS deviation model accurately tracks the random-codeword performance measured using MC-DE.

Let $p_L$ and $p_H$ be respectively the smallest and largest $p_e^{(t-1)}$ values for which the deviations
have been characterized. We can generate a conditional distribution for any $p_e^{(t-1)} \in [p_L, p_H]$ by interpolating from the nearest distributions that have been measured. We choose $p_H \geq p_e^{(0)}$ to make sure that the first iteration’s deviation is within the characterized range. Because messages in the decoder are saturated once they reach the largest magnitude that can be represented, the circuit’s switching activity decreases when the message error probability becomes very small. Since timing faults cannot occur when the circuit does not switch, we can expect deviations to be equally or less likely at $p_e^{(t-1)}$ values below $p_L$. Therefore, to define the deviation model for $p_e^{(t-1)} < p_L$, we make the pessimistic assumption that the deviation distribution remains the same as for $p_e^{(t-1)} = p_L$.

E. DE and Energy Curves

We evaluate the progress of the decoder affected by timing violations using quantized density evolution [30]. For the Offset Min-Sum algorithm, a DE iteration can be split into the following steps: 1-a) evaluating the distribution of the CN minimum, 1-b) evaluating the distribution of the CN output, after subtracting the offset, 2) evaluating the distribution of the ideal VN-to-CN message, and 3) evaluating the distribution of the faulty VN-to-CN messages. Step 1-a is given in [15], while the others are straightforward. In the context of DE, we write the message distribution as $\pi(t) = \mathbb{P}(\mu_{i,j}^{(t)} | x_i = 1)$, and the channel output distribution as $\pi(0) = \mathbb{P}(\mu_{i}^{(0)} | x_i = 1)$. We write a DE iteration as $\pi(t+1) = f_\gamma(\pi(t), \pi(0))$.

The energy consumption is measured on the test circuit using a 1-D input message distribution, and therefore it is best described in terms of the message error probability. We write the energy function as $c_\gamma(p_e^{(t)})$. As for the deviation model, we use interpolation to define $c_\gamma(p_e^{(t)})$ for $p_e^{(t)} \in [p_L, p_H]$, and assume that $c_\gamma(p_e^{(t)}) = c_\gamma(p_L)$ for $p_e^{(t)} < p_L$. To display $f_\gamma(\pi(t), \pi(0))$ and $c_\gamma(p_e^{(t)})$ on the same plot, we project $\pi(t)$ onto the message error probability space.

Several regular code ensembles were evaluated, with rates $\frac{1}{2}$ and $\frac{9}{10}$. Fig. 9 shows examples of projected DE curves and energy curves for rate-$\frac{1}{2}$ code ensembles with $d_v \in \{3, 4, 5\}$ and various operating conditions. The energy is measured as described in Appendix A and corresponds to one complete decoding iteration performed with the test circuit, requiring $d_v$ uses of the test circuit. The nominal operating condition is $V_{dd} = 1.0 \text{ V}$, $T_{clk} = 2.0 \text{ ns}$ and therefore these curves correspond to a reliable implementation. With a reliable implementation, these ensembles have a channel threshold of $p_e^{(0)} \leq 0.12$ for the $(3, 6)$ ensemble, $p_e^{(0)} \leq 0.11$ for $(4, 8)$, and $p_e^{(0)} \leq 0.09$ for

April 19, 2016
(5, 10). We use $p_e^{(0)} = 0.09$ for all the curves shown in Fig. 9 to allow comparing the ensembles. As can be expected, a larger variable node degree results in faster convergence towards zero error rate, and it is natural to ask whether this property might provide greater fault tolerance and ultimately better energy efficiency. However, we can see that the energy per iteration increases rapidly as $d_v$ increases, and our energy optimization results show that for minimizing energy or
EDP it is best to choose the ensemble with the smallest possible variable node degree.

Fig. 10 is a similar plot for the \((3, 30)\) and \((4, 40)\) ensembles. The channel threshold of both ensembles is approximately \(p_e^{(0)} \leq 0.019\). For these curves, the nominal operating condition is \(V_{dd} = 1.0\) V and \(T_{clk} = 3\) ns. As we can see, the energy consumption per iteration of the \((4, 40)\) decoder is roughly double that of the \((3, 30)\) decoder. We note that in the case of the \((3, 30)\) ensemble, the reliable decoder stops making progress at an error probability of approximately \(10^{-8}\). This floor is the result of the message saturation limit chosen for the circuit.

V. ENERGY OPTIMIZATION

A. Design Parameters

As in a standard LDPC code-decoder design, the first parameter to be optimized is the choice of code ensemble. In this paper we restrict the discussion to regular codes, and therefore we need only to choose a degree pair \((d_v, d_c)\), where \(R = 1 - d_v/d_c\) is the design rate of the code. For a fixed \(R\), we can observe that both the energy consumption and the circuit area of the decoding circuit grow rapidly with \(d_v\), and therefore it is only necessary to consider a few of the lowest \(d_v\) values.

Besides the choice of ensemble, we are interested in finding the optimal choice of operating parameters for the quasi-synchronous circuit. We consider here the supply voltage \((V_{dd})\) and the clock period \((T_{clk})\). Generally speaking, the supply voltage affects the energy consumption, while the clock period affects the decoding time, or latency. The energy and latency are also affected by the choice of code ensemble, since the number of operations to be performed depends on the node degrees. The operating parameters of a decoder are denoted as a vector \(\gamma = [V_{dd}, T_{clk}]\).

The decoding of LDPC codes proceeds in an iterative fashion, and it is therefore possible to adjust the operating parameters on an iteration-by-iteration basis. In practice, this could be implemented in various ways, for example by using a pipelined sequence of decoder circuits, where each decoder is responsible for only a portion of the decoding iterations. It is also possible to rapidly vary the clock frequency of a given circuit by using a digital clock divider circuit [31]. We denote by \(\vec{\gamma}\) the sequence of parameters used at each iteration throughout the decoding, and we use \(\vec{\gamma} = [\gamma_1^{N_1}, \gamma_2^{N_2}, \ldots]\) to denote a specific sequence in which the parameter vector \(\gamma_1\) is used for the first \(N_1\) iterations, followed by \(\gamma_2\) for the next \(N_2\) iterations, and so on.
B. Objective

The performance of the LDPC code and of its decoder can be described by specifying a vector \( P = (p_e^{(0)}, p_{\text{res}}, T_{\text{dec}}) \), where \( p_e^{(0)} \) is the output error rate of the communication channel, \( p_{\text{res}} \) the residual error rate of VN-to-CN messages when the decoder terminates, and \( T_{\text{dec}} \) the expected decoding latency.

The decoder’s performance \( P \) and energy consumption \( E \) are controlled by \( \bar{\gamma} \). The energy minimization problem can be stated as follows. Given a performance constraint \( P = (a, b, c) \), we wish to find the value of \( \bar{\gamma} \) that minimizes \( E \), subject to \( p_e^{(0)} \geq a \), \( p_{\text{res}} \leq b \), \( T_{\text{dec}} \leq c \). As in the standard DE method, we propose to use the code’s computation tree as a proxy for the entire decoder, and furthermore to use the energy consumption of the test circuit described in Appendix B as the optimization objective. To be able to replace the energy minimization of the complete decoder with the energy minimization of the test circuit, we make the following assumptions:

1) The ordering of the energy consumption is the same for the test circuit and for the complete decoder, that is, for any \( \gamma_1 \) and \( \gamma_2 \), \( E_{\text{test}}(\gamma_1) \leq E_{\text{test}}(\gamma_2) \) implies \( E_{\text{dec}}(\gamma_1) \leq E_{\text{dec}}(\gamma_2) \), where \( E_{\text{test}}(\gamma) \) and \( E_{\text{dec}}(\gamma) \) are respectively the energy consumption of the test circuit and of the complete decoder when using parameter \( \gamma \).

2) The average message error rate in the test circuit and in the complete decoder is the same for all decoding iterations.

3) The latency of the complete decoder is proportional to the latency of the test circuit, that is, if \( T_{\text{dec}}(\gamma) \) is the latency measured using the test circuit with parameter \( \gamma \), the latency of the complete decoder is given by \( \beta T_{\text{dec}}(\gamma) \), where \( \beta \) does not depend on \( \gamma \).

Assumption 1 is reasonable because the test circuit is very similar to a computation unit used in the complete decoder. The difference between the two is that the test circuit only instantiates one full VNP, the remaining \((d_c - 1)\) VNPs being reduced to only their “front” part (as seen in Fig. 11), whereas the complete decoder has \(d_c\) full VNPs for every CNP. Assumption 2 is the standard DE assumption, which is reasonable for sufficiently long codes. Finally, it is possible for the clock period to be slower in the complete decoder, because the increased area could result in longer interconnections between circuit blocks. Even if this is the case, the interconnect length only depends on the area of the complete decoder, which is not affected by the parameters.
we are optimizing, and hence $\beta$ does not depend on $\gamma$.

Clearly, if Assumption 1 holds and the performance of the test circuit is the same as the performance of the complete decoder, then the solution of the energy minimization is also the same. The performance is composed of the three components $(p_e^{(0)}, p_{res}, T_{dec})$. The channel error rate $p_e^{(0)}$ does not depend on the decoder and is clearly the same in both cases. Because of Assumption 2, the complete decoder can achieve the same residual error rate as the test circuit when $p_e^{(0)}$ is the same. The latencies measured on the test circuit and on the complete decoder are not necessarily the same, but if Assumption 3 holds, and if we assume that the constant $\beta$ is known, then we can find the solution to the energy minimization of the complete decoder subject to constraints $(p_e^{(0)}, p_{res}, T_{dec})$ by instead minimizing the energy of the test circuit with constraints $(p_e^{(0)}, p_{res}, T_{dec}/\beta)$.

We also consider another interesting optimization problem. It is well known that for a fixed degree of parallelism, processing speed (represented here by $T_{dec}$) and energy consumption have a proportional relationship, which is observed both in the physical energy limit stemming from Heisenberg’s uncertainty principle [32], as well as in practical CMOS circuits [33]. In situations where both throughput normalized to area and low energy consumption are desired, optimizing the product of energy and latency or energy-delay product (EDP) for a fixed circuit area can be a better objective. In that case the performance constraint is stated in terms of $P = (p_e^{(0)}, p_{res})$, and the optimization problem becomes the following: given a performance constraint $P = (a, b)$, minimize $E(\bar{\gamma}) \cdot T_{dec}(\bar{\gamma})$ subject to $p_e^{(0)} \geq a$, $p_{res} \leq b$, and a fixed circuit area.

C. Dynamic Programming

To solve the iteration-by-iteration energy and EDP minimization problems stated above, we adapt the “Gear-Shift” dynamic programming approach proposed in [20]. The original method relies on the fact that the message distribution has a 1-D characterization, which is chosen to be the error probability. By quantizing the error probability space, a trellis graph can be constructed in which each node is associated with a pair $(\bar{p}_e^{(t)}, t)$. Quantized quantities are marked with tildes. A particular choice of $\bar{\gamma}$ corresponds to a path $P$ through the graph, and the optimization is transformed into finding the least expensive path that starts from the initial state $(\bar{p}_e^{(0)}, 0)$ and reaches any state $(\bar{p}_e^{(t)}, t)$ such that $\bar{p}_e^{(t)} \leq p_{res}$ and the latency constraint is satisfied, if there is one. Note that to ensure that the solutions remain achievable in the original continuous space,
the message error rates $p_e(t)$ are quantized by rounding up. To maintain a good resolution at low error rates, we use a logarithmic quantization, with 1000 points per decade.

In the case of a faulty decoder, we want to evaluate the decoder’s progress by tracking a complete message distribution using DE, rather than simply tracking the message error probability. In this case, the Gear-Shift method can be used as an approximate solver by projecting the message distribution $\pi(t) = \mathbb{P}(\mu_{i,j}|x_i = 1)$ onto the error probability space. We refer to this method as DE-Gear-Shift. Any path through the graph is evaluated by performing DE on the entire path using exact distributions, but different paths are compared in the projection space. As a result, the solutions that are found are not guaranteed to be optimal, but they are guaranteed to accurately represent the progress of the decoder.

In the DE-Gear-Shift method, a path $P$ is a sequence of states $\{\pi(t)\}$. As in the original Gear-Shift method, any sequence of decoder parameters $\gamma$ corresponds to a path. We denote the projection of a state onto the error probability space as $p_e(t) = \Theta(\pi(t))$. To each path $P$, we associate an energy cost $E_P$ and a latency cost $T_P$. A path ending at a state $\pi(t)$ can be extended with one additional decoding iteration using parameter $\gamma$ by evaluating one DE iteration to obtain $\pi(t+1) = f_\gamma(\pi(t), \pi(0))$. Performing this additional iteration adds an energy cost $c_\gamma(\tilde{p}_e(t), p_e(0))$ and a latency cost $T_\gamma$ to the path’s cost. When optimizing EDP, we define the overall cost of a path

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>(3,6)</td>
<td>2.0 ns</td>
<td>1.066</td>
<td>0.12 $\dagger$</td>
<td>$\leq 10^{-8}$</td>
<td>198</td>
<td>750</td>
<td>149</td>
<td>575 (-23%)</td>
<td>114 (-23%)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0.09</td>
<td>$\leq 10^{-8}$</td>
<td>66</td>
<td>185</td>
<td>13.5</td>
<td>135 (-34%)</td>
<td>8.8 (-35%)</td>
</tr>
<tr>
<td>(4,8)</td>
<td>2.0 ns</td>
<td>1.44</td>
<td>0.09</td>
<td>$\leq 10^{-8}$</td>
<td>72</td>
<td>394</td>
<td>28.4</td>
<td>299 (-24%)</td>
<td>21.2 (-25%)</td>
</tr>
<tr>
<td>(3,30)</td>
<td>3.0 ns</td>
<td>1.099</td>
<td>0.019 $\dagger$</td>
<td>$\leq 10^{-8}$</td>
<td>252</td>
<td>2650</td>
<td>668</td>
<td>1816 (-31%)</td>
<td>437 (-35%)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>2.5 ns</td>
<td>1.135</td>
<td>0.019 $\dagger$</td>
<td>$\leq 10^{-8}$</td>
<td>210</td>
<td>2748</td>
<td>577 2004 (-27%)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>3.0 ns</td>
<td>1.099</td>
<td>0.015</td>
<td>$\leq 10^{-8}$</td>
<td>117</td>
<td>918</td>
<td>107 588 (-36%)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>2.5 ns</td>
<td>1.135</td>
<td>0.015</td>
<td>$\leq 10^{-8}$</td>
<td>97.5</td>
<td>972</td>
<td>95 653 (-33%)</td>
</tr>
<tr>
<td>(4,40)</td>
<td>3.0 ns</td>
<td>1.522</td>
<td>0.015</td>
<td>$\leq 10^{-8}$</td>
<td>108</td>
<td>1456</td>
<td>157</td>
<td>897 (-38%)</td>
<td>95 (-40%)</td>
</tr>
</tbody>
</table>

$\dagger$ Cell area divided by the minimal area of the smallest decoder having the same code rate. $\dagger$ Approx. threshold.
$C_P$ as $C_P = E_P \cdot T_P$. When optimizing energy under a latency constraint, we define the path cost as a two-dimensional vector $C_P = (E_P, T_P)$.

We use the following rules to discard paths that are suboptimal in the error probability space. Rule 1: Paths for which the message error rate is not monotonically decreasing are discarded. Rule 2: A path $P$ with cost $C_P$ is said to dominate another path $P'$ with cost $C_{P'}$ if all the following conditions hold: 1) an ordering exists between $C_P$ and $C_{P'}$, 2) $C_P \leq C_{P'}$, 3) $\Theta(\pi_P) \leq \Theta(\pi_{P'})$, where $\pi_P$ denotes the last state reached by path $P$. The search for the least expensive path is performed breadth-first. After each traversal of the graph, any path that is dominated by another is discarded.

When the path cost is one-dimensional, the optimization requires evaluating $O(|\Gamma| N_s)$ DE iterations, where $|\Gamma|$ is the number of operating points being considered and $N_s$ the number of quantization levels used for $\tilde{p}_e^{(t)}$. This can be seen from the fact that with a 1-D cost, Rule 2 implies that at most one path can reach a given state $\tilde{p}_e^{(t)}$. Therefore, $O(|\Gamma| N_s)$ DE iterations are required for each decoding iteration. In addition, upper bounds can be derived for the number of decoding iterations spanned by the trellis graph in terms of the smallest latency and energy cost of the parameters in $\Gamma$, and therefore it is a constant that does not depend on $|\Gamma|$ or $N_s$. On the other hand, when the cost is two-dimensional, the number of DE iterations could grow exponentially in terms of the number of decoding iterations. However, even in the case of a 2-D cost, an ordering exists between the costs of paths $P$ and $P'$ if $(E_P \geq E_{P'} \land T_P \geq T_{P'}) \lor (E_P \leq E_{P'} \land T_P \leq T_{P'})$, and in that case Rule 2 can be applied. In practice, for the cases presented in this paper, the discarding rules allowed to keep the number of paths down to a manageable level, even when using a 2-D cost. Note that an alternative to the use of a 2-D cost is to define a 1-D cost as $C_P = E_P + \kappa T_P$, and to perform a binary search for the value of $\kappa$ that yields an optimal solution with the desired latency.

The algorithm can also be modified to search for parameter sequences that have other desirable properties beyond minimal energy or EDP. For example, if the decoder is implemented as a pipelined sequence of decoders, it can be desirable to favor solutions that do not require the decoder to switch its parameters too often. We can find good approximate solutions by adding a penalty to $E_P$ when the algorithm used in the current and next steps is different.
D. Results

We use DE-Gear-Shift to find good parameter sequences $\gamma$ for several regular ensembles with rates $\frac{1}{2}$ and $\frac{9}{10}$. The parameter space $\Gamma$ consists of $(V_{dd}, T_{clk})$ points with $V_{dd}$ from 0.70 V to 1.0 V in steps of 0.05 V and several $T_{clk}$ values depending on $V_{dd}$, in steps of 0.1 ns. The standard and quasi-synchronous decoders use the same circuits. Parameter $\alpha$ in (1) is set to $\alpha = 4$ for the $(3, 6)$, $(3, 30)$, and $(4, 40)$ decoders, and to $\alpha = 2$ for the $(4, 8)$ decoder. Parameter $C$ in (4) is set to $C = 2$ for the $(4, 40)$ decoder and to $C = 1$ for all other decoders. As part of our best effort to design a good standard circuit, in the case of the $(3, 30)$ decoder we present results for two circuits synthesized with different nominal $T_{clk}$ values. The standard circuit has a lower energy consumption using the first circuit, while it has a lower EDP with the second circuit.

We first run the DE-Gear-Shift solver without any path penalties to obtain the best possible parameter sequences, for both the energy and the EDP objectives. We also noticed that in some cases, adding a small algorithm change penalty allows to discover slightly better sequences. Note that when the objective is EDP, there is no constraint on latency. These results are summarized in Table I. Overall, we see that significant gains are possible while achieving the same channel noise, latency, and residual error requirements. The results confirm that among these regular ensembles, increasing $d_v$ causes a significant increase in the decoding energy, both for a standard decoder and for a quasi-synchronous decoder, in addition to an increase in circuit area. Furthermore, we see that much more energy is required when operating the decoder close to the channel threshold. Nonetheless, significant energy or EDP gains are still possible close to the threshold, although the gains increase with a better channel.

By applying a cost penalty to parameter switches, it is possible to find parameter sequences with few switches, without a large increase in cost. For example, for a $(3, 6)$ decoder starting at $p_e^{(0)} = 0.09$, two $(V_{dd}, T_{clk})$ pairs are sufficient to provide a 33% EDP improvement, using $\gamma = [[0.8 V, 2.1 ns]^8, [1.0 V, 1.4 ns]^3]$. The average deviation probability in that schedule ranges from 1.6% to 7.2%. In the case of a $(3, 30)$ decoder, at $p_e^{(0)} = 0.015$, the sequence $\gamma = [[0.8 V, 3 ns]^{12}, [1.0 V, 2.5 ns]]$ provides a 36% EDP improvement, with average deviation probabilities from 0 to 0.9%. For a $(4, 40)$ decoder, the single-parameter sequence $\gamma = [[0.8 V, 2.8 ns]^9]$ provides a 39% EDP improvement, with average deviation probabilities from 0.4 to 3%.
VI. Conclusion

We presented a method for the design of synchronous circuit implementations of signal processing algorithms that permits timing violations without the need for hardware compensation. The method uses small test circuits to obtain accurate deviation statistics and energy estimation. We introduced a model for the deviations occurring in LDPC decoder circuits affected by timing faults that is sufficiently general to accurately represent the circuit behavior, while still being memoryless. In addition, we showed that in order to use density evolution to predict the performance of the faulty decoder, it is sufficient for the deviation model to have a weak symmetry property, which is more general than previously proposed sufficient properties.

We then presented an approximate optimization method called DE-Gear-Shift to find sequences of circuit operating parameters that minimize the energy or the energy-delay product. The method is similar to the previously proposed Gear-Shift method, but relies on density evolution rather than ExIT charts to evaluate the average iterative progress of the decoder. Our results show that the best energy or EDP reduction is achieved by operating the circuit with a large number of timing violations (often with an average deviation rate above 1%). Furthermore, important savings can be achieved with few parameter switches, and without any compromise on circuit area or decoding performance.

APPENDIX A

CAD WORKFLOW

To make the characterization as accurate as possible, we measure the deviations and the energy consumption directly on optimized circuit models generated by a commercial synthesis tool (Cadence Encounter [34]). We use TSMC’s 65 nm process with the tcbn65gplus cell library [35]. In order to provide a fair assessment of the improvements provided by the quasi-synchronous circuit, we first synthesize a benchmark circuit that represents a best effort at optimizing the metric of interest, for example energy consumption. Since we do not have a specific throughput constraint for the design, we synthesize the benchmark circuit at the standard supply voltage of the library ($V_{dd} = 1.0V$), while the clock period is chosen as small as possible without causing a degradation of the target metric. Second, we synthesize a nominal circuit that will serve as the basis for the quasi-synchronous design. In this work, we use a standard synthesis algorithm for the nominal circuit, and in all the cases that we report on, the nominal
and the benchmark circuits are actually the same. Using a standard synthesis method for the nominal circuit allows using off-the-shelf tools, but is not ideal since the objective of a standard synthesis algorithm (to make all paths only as fast as the clock period) differs from the objective pursued when some timing violations are permitted. For example, results in [36] show that the power consumption of a circuit can be reduced by up to 32% when the gate-sizing optimization takes into account the frequency at which the clock constraint can be violated. Therefore it is possible that our results could be improved by using a more specialized synthesis algorithm.

Once the circuit is synthesized, we perform a static timing analysis of the gate-level model at various supply voltages. All timing analyses (including at the nominal supply) are performed using timing libraries generated by the Cadence Encounter Library Characterization tool. We then use this timing information in a functional simulation of the gate-level circuit to observe the dynamic effect of path delay variations and measure the deviation statistics. Any source of delay variation that can be simulated can be studied, but in this paper we focus on variations due to path activation, that is the variations in delay caused by the different propagation times required by different input transitions. Note that other methods could be used to obtain the propagation delays, such as the method described in [37] based on analytical models. In addition to speeding up the characterization, such methods allow considering the effect of process variations.

Power estimation is performed by collecting switching activity data in the functional simulation and using the power estimation engine in Cadence Encounter. However, because the circuit is operated in a quasi-synchronous manner, the clock period used to run the circuit is not necessarily the same as the nominal clock period. When that is the case, the power estimation generated by the synthesis tool cannot be used directly. First, the switching activity recorded during the functional simulation must be scaled so that it corresponds to the nominal clock period. The tool’s power estimation then reports the dynamic power $P_{\text{dyn}}$ and the static power $P_{\text{stat}}$. The dynamic energy consumed during one clock cycle does not depend on the clock period, whereas the static energy does. Therefore, the total energy consumed during one cycle by the quasi-synchronous circuit is given by $E_{\text{cycle}} = P_{\text{dyn}} T_{\text{clk,nom}} + P_{\text{stat}} T_{\text{clk}}$, where $T_{\text{clk,nom}}$ is the nominal clock period and $T_{\text{clk}}$ is the actual clock period used to run the circuit.
APPENDIX B
TEST CIRCUIT MONTE-CARLO SIMULATION

A suitable test circuit for a row-layered decoder architecture consists in implementing a single check node processor, as well as the necessary logic taken from the variable node processor block to send $d_v$ messages to the CNP, and receive one message from the CNP. This test circuit is shown in Fig. 11. It re-uses logic blocks that are found in the complete decoder, ensuring the accuracy of the deviation and energy measurements, and minimizing design time. At any given clock cycle, a VNP front block is mapped to a particular VN $i$. Each VNP front block takes as input the previous belief total of that VN, denoted $\Lambda'_{i}$, and the previous CN-to-VN message for the current layer $\ell$, denoted $\lambda_{i}^{(t-1,\ell)}$. The previous belief total is defined as

$$
\Lambda'_{i} = \begin{cases} 
\Lambda_{i}^{(t-1,L)} & \text{if } \ell = 1, \\
\Lambda_{i}^{(t,\ell-1)} & \text{if } \ell > 1.
\end{cases}
$$

As part of the proposed design framework, we require that the input of the test circuit be accurately modeled as an i.i.d. process when the test circuit is used in the final system. To determine whether the circuit’s input can realistically be modeled as an i.i.d. process (with respect to the clock cycles), we must consider the way the circuit is used when instantiated in the complete decoder. The input of a given VNP front is determined by the VN it is associated with, as well as the iteration and layer indices. If we assume that the code graph is cycle free, the messages sent by all VNs at iteration $t$ and layer $\ell$ are i.i.d. Therefore, when a VNP is assigned to process a sequence of distinct VNs belonging to the same $(t, \ell)$, its inputs can be represented by an i.i.d. process. If the number of CNPs instantiated in the decoder is significantly less than $m/L$, then for most cycles the current and the next inputs of a VNP do belong to the same $(t, \ell)$, making this a reasonable approximation.

To perform the Monte-Carlo simulation, a VNP front circuit block with index $i$ must send a message $\mu_{i}^{(t)}$, randomly generated according to either a 1-D normal distribution with error probability $p_{e}^{(t)}$ (for deviation measurements), or a general discrete distribution (for MC-DE). However, the only inputs that are controllable are $\Lambda'_{i}$ and $\lambda_{i}^{(t-1,\ell)}$. To simplify the Monte-Carlo simulation, we disregard the true distribution of $\lambda_{i}^{(t-1,\ell)}$ and generate it according to a 1-D normal distribution. We also introduce another simplification: we assume that messages received at a VN only modify the total belief at the end of the iteration, as would be the case when using
a flooding schedule. As a result, the messages $\mu^{(t,\ell)}_i$ are identically distributed with error rate parameter $p_e(t)$ for all $\ell$. Note that these simplifications are not necessary, and they could be removed at the cost of a slightly more cumbersome Monte-Carlo simulation.

We have that $\Lambda'_i = \mu_1^{(t,\ell)} + \lambda_1^{(t-1,\ell)}$. On a cycle-free graph, $\mu_1^{(t,\ell)}$ and $\lambda_1^{(t-1,\ell)}$ are independent, but naturally $\Lambda'_i$ and $\lambda_i^{(t-1,\ell)}$ are not. Therefore, we generate $\mu_1^{(t,\ell)}$ and $\lambda_i^{(t-1,\ell)}$, sum them to obtain $\Lambda'_i$, and then discard $\mu_1^{(t,\ell)}$.

In the test circuit, the VNP with index 1 (shown at the top of Fig. 11) is at the root of the computation tree. To complete the DE iteration, we want to measure

$$\mu_1^{(t+1,d_v)} = \lambda_1^{(t,1)} + \lambda_1^{(t,2)} + \cdots + \lambda_1^{(t,d_v-1)}.$$  \hspace{1cm} (11)

To achieve this, we set $\Lambda_1^{(t-1,L)} \leftarrow 0$ and $\lambda_1^{(t-1,\ell)} \leftarrow 0$ for all $\ell$. We then have $\mu^{(t+1,d_v)} = \Lambda_1^{(t,d_v-1)}$, that is the next extrinsic message corresponds directly to the circuit’s total belief output after it has been used $d_v - 1$ times.

**ACKNOWLEDGEMENTS**

The authors wish to thank CMC Microsystems for providing access to the Cadence tools and TSMC 65nm CMOS technology, and Gilles Rust for advice on Cadence tools and cell library characterization.
REFERENCES


[35] TCBN65GPLUS TSMC 65nm Core Library Databook, Taiwan Semiconductor Manufacturing Company, Ltd.
