High Throughput and Low Power Enhancements for LDPC Decoders

Erick Amador and Raymond Knopp
EURECOM
06904 Sophia Antipolis, France
name.surname@eurecom.fr

Renaud Pacalet
TELECOM ParisTech
06904 Sophia Antipolis, France
renaud.pacalet@telecom-paristech.fr

Vincent Rezard
Infineon Technologies France
06560 Sophia Antipolis, France
vincent.rezard@infineon.com

***Abstract*—Modern VLSI decoders for low-density parity-check (LDPC) codes require high throughput performance while achieving high energy efficiency on the smallest possible footprint. In this paper, we present two optimizations to enhance the throughput and reduce the power consumption for these decoders. As a first optimization, we seek to speedup the decoding task by modifying the processing step known as *syndrome check*. We partition this task and perform it in *on-the-fly* fashion. As a second optimization, we address the topic of iteration control in order to save energy and time on unnecessary decoder operation when processing undecodable blocks. We propose an iteration control policy that is driven by the combination of two decision metrics. Furthermore, we show empirically how stopping criteria should be tuned as a function of false alarm and missed detection rates. Throughout this paper we use the codes defined in the IEEE 802.11n standard to show performance results of the proposed optimizations.**

***Keywords*—LDPC codes; iterative decoding; syndrome calculation; throughput enhancement; stopping criteria; iteration control; low power.**

I. Introduction

Low-density parity-check (LDPC) codes have gained a lot of interest because of their outstanding error-correction performance. Originally proposed by Gallager in 1962 [3] and rediscovered by Mackay [4] in the 1990s, these codes exhibit a performance that comes very close to the limits imposed by Shannon.

Several communication standards have already adopted these codes, ranging from Wireless Local/Metropolitan Area Networks (IEEE 802.11n [5] and 802.16e [6]) and high-speed wireless personal area networks (IEEE 802.15.3c [7]) to Digital Video Broadcast (DVB-S2 [8] and DTMB [9]) and 10Gbit Ethernet (10GBASE-T [10]). Furthermore, these codes are currently being proposed for next generation cellular and mobile broadband systems as defined by the ITU-R to comply with the IMT-Advanced radio interface requirements: IEEE 802.16m [11] and 3GPP LTE-Advanced [12].

In the case of mobile wireless terminals high throughput and low power operation are required. Nevertheless, these goals are often contradictory due mainly to the iterative nature of the decoding algorithms used. For a successful decoding task the fulfillment of all parity-check constraints is verified, but usually for an unsuccessful task a preset maximum number of iterations is completed.

In this paper, we propose to optimize one recurrent task that is performed within each decoding iteration. Syndrome check or verification is performed in order to confirm the validity of the obtained codeblock and hence decide whether to continue or halt the decoding process. This task corresponds to the evaluation of all the parity-check constraints imposed by the parity-check matrix. We propose to perform this task *on-the-fly* so that a partially unsatisfied parity-check constraint can disable a potential useless syndrome verification on the entire matrix. We identify as benefits from this technique the elimination of several hardware elements, a reduction on the overall task latency and an increase on system throughput.

One form of the proposed technique has been identified in [13] for the purpose of improving the energy-efficiency of a decoder. This technique, nevertheless, is sub-optimal in the error-correction sense as it introduces undetected codeblock errors. In this work, we show the assumptions for this technique and a performance analysis, along with a proposal to recover the performance loss.

As a second topic, we consider iteration control for avoiding unnecessary decoder operation. Early detection of an undecodable block not only saves energy on unnecessary iterations but may improve the overall latency when an automatic repeat-request (ARQ) strategy is also in use. Previously proposed iteration control policies [14][15][16][17] differ on the decision metrics used. These decision metrics are characterized by their dependence or not upon extraneous variables that must be estimated. The parameters used within the decision rule must be tuned to particular scenarios. In the previous art it has been shown how this tuning essentially trades off error-correction performance and the average number of iterations.

In this work, we identify a decision metric provided by a specific decoding algorithm, the Self-Corrected Min-Sum algorithm [18]. This algorithm has been shown to provide quasi-optimal error-correction performance at very low complexity. We propose to combine two decision metrics in order to control the iterative decoding task. We perform comparisons among the previous art and the proposed hybrid control policy in terms of error-correction performance, average number of iterations and false alarm rate. The main advantage our work shows is the energy efficiency of the proposed policy as it exhibits empirically very low missed detection rates. Furthermore, we argue that the tuning of parameters of a

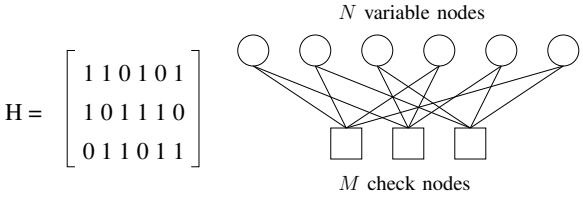


Fig. 1. LDPC code matrix and graph example.

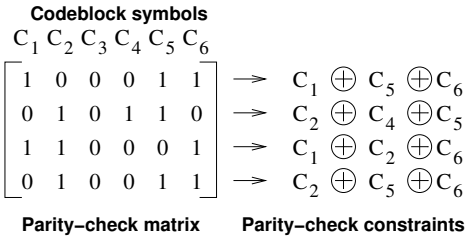


Fig. 2. Example parity-check constraints.

stopping rule should be done based upon the false alarm and missed detection rates performance.

This paper merges and extends our previous work in [1][2]. The remainder of this paper is organized as follows. Section II presents LDPC codes and their iterative decoding. Section III outlines the proposed syndrome check method and its performance while Section IV shows the system level impact along with results for a VLSI architecture. In Section V, we show prior stopping criteria and the proposed iteration control policy while Section VI shows simulation results and the tuning of stopping criteria. Section VII concludes the paper.

II. Background

In this section, we introduce the target error-correction codes along with their iterative decoding algorithm and the type of messages used in the computation kernels.

A. LDPC Codes

Binary LDPC codes are linear block codes defined by a sparse parity-check matrix $\boldsymbol{H}_{M\times N}$ over GF(2). This matrix defines M parity-check constraints among N code symbols. The number of non-zero elements in $\boldsymbol{H}$ is relatively small compared to the dimensions $M \times N$ of $\boldsymbol{H}$.

A codeword $\boldsymbol{c}$ corresponds to the null space of $\boldsymbol{H}$:

$$\boldsymbol{H} \cdot \boldsymbol{c}^T = \boldsymbol{S} = \boldsymbol{0}\ , \tag{1}$$

where $\boldsymbol{S}$ is referred to as the *syndrome*. Indeed, the condition $\boldsymbol{S} = \boldsymbol{0}$ suggests that no further decoding iterations are necessary. Typically, a maximum number of iterations is set to define an unsuccessful decoding operation.

A quasi-cyclic (QC) LDPC code is obtained if $\boldsymbol{H}$ is formed by an array of sparse circulants of the same size, [19]. If $\boldsymbol{H}$ is a single sparse circulant or a column of sparse circulants this results in a cyclic LDPC code. *Architecture-aware* [20] and QC-LDPC codes are composed of several layers of non-overlapping rows, this enables the concurrent processing of subsets of rows without conflicts.

The code can also be represented by a bipartite graph in which rows of $\boldsymbol{H}$ are mapped to *check nodes* and columns to *variable nodes*. The non-zero elements in $\boldsymbol{H}$ define the connectivity between the nodes. Figure 1 shows an example matrix (non-sparse) and the corresponding code graph representation.

The rows in $\boldsymbol{H}$ establish the parity constraints of the code as a function of the code symbols with the location of the non-zero elements of the matrix. Figure 2 shows an example correspondance between the code symbols C_n, the parity-check matrix and the parity-check constraints.

The parity-check constraints are of even parity and the $\oplus$ operation corresponds to the modulo-2 addition.

B. Decoding Algorithm

LDPC codes are typically decoded iteratively using a two-phase message-passing algorithm commonly known as *sum-product* [21] or *belief propagation*. This algorithm exchanges code symbol extrinsic reliability values between check and variable nodes. Each decoding iteration consists of two phases: variable nodes update and send messages to the neighboring check nodes, and check nodes update and send back their corresponding messages. Node operations are in general independent and may be executed in parallel. This allows the possibility to use different scheduling techniques that may impact the convergence speed of the code and the storage elements requirements. The algorithm initializes with intrinsic channel reliability values and iterates until hard-decisions upon the accumulated resulting posterior messages satisfy equation (1). Otherwise, a maximum number of iterations is completed.

The computational complexity of the decoding task resides in the operation performed at the check nodes of the code graph, indeed it is in here where the tradeoff between error-correction performance and complexity takes place. Optimal message computation is performed by the Sum-Product algorithm [21] at the expense of high complexity. The Min-Sum (MS) algorithm [22] performs a sub-optimal message computation at reduced complexity. Several correction methods have been proposed to recover the performance loss of the MS algorithm by downscaling the messages computed using a normalization or an offset value, [22].

It has been argued in [18] that the sub-optimality of MS decoding is not due to the overestimation of the check node messages, but instead to the loss of the symmetric Gaussian distribution of these messages. This symmetry can be recovered by eliminating unreliable variable node messages or *cleaning* the inputs of the check node operation. In [18] the Self-Corrected MS (SCMS) decoding is introduced, which exhibits quasi-optimal error-correction performance. An input to the check node operation is identified as *unreliable* if it has changed its sign with respect to the previous iteration. Unreliable messages are *erased* and are no longer propagated along the code graph. In [23] a comparison is performed in terms of energy efficiency among the most prominent message computation kernels.

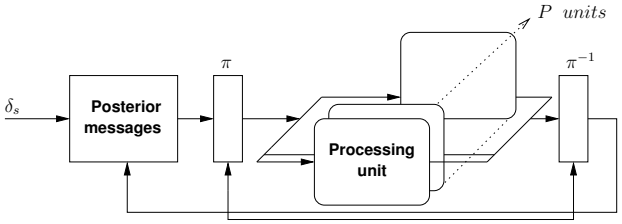


Fig. 3. General structure of an LDPC decoder.

Motivated by the outstanding error-correction performance and low complexity of the SCMS kernel, in Section V we look closely at the behavior of this kernel in order to assist the early detection of undecodable blocks.

The general structure of an LDPC decoder is shown in Figure 3. *Intrinsic* channel values δ_s are initially used to generate *extrinsic* messages, these are messages generated after processing each row in $\boldsymbol{H}$. The sum of all extrinsic messages generated constitutes the posterior messages that are used to perform a hard-decision and obtain the final decoded message. The posterior messages are distributed to and from P processing units by interleaving units (π and π^{-1}) that correspond to the code graph connectivity.

Log-likelihood (LLR) messages are commonly used since their arithmetic [24] exhibits very low complexity (e.g., additions instead of multiplications). For every received code symbol x the corresponding LLR is given by:

$$L(x) = \log \frac{P(x=0)}{P(x=1)} \ , \tag{2}$$

where $P(A = y)$ defines the probability that A takes the value y. LLR values with a positive sign would imply the presence of a logic 0 whereas a negative sign would imply a logic 1. The magnitude of the LLR provides a measure of reliability for the hypothesis regarding the presence of a logic 0 or 1. Considering the messages involved in the decoding process, the LLR of an information bit x is given by:

$$L(x) = L_c(x) + L_a(x) + L_e(x) \ , \tag{3}$$

where $L_c(x)$ is the intrinsic message received from the channel, $L_a(x)$ is the a-priori value and $L_e(x)$ is the extrinsic value estimated using the code characteristics and constraints. $L(x)$ is the *a posteriori* value and a hard-decision upon it (extraction of the mathematical sign) is used to deduce the binary decoded value. Figure 4 shows the evolution of the posterior messages LLRs for both a converging and a non-converging codeblock. These figures correspond to instances of decoding the LDPC code defined in [5] with block length of 648 and code rate 1/2 over the additive white Gaussian noise (AWGN) channel with quadrature phase-shift keying (QPSK) modulation at a signal-to-noise ratio (SNR) of $E_b/N_0 = 1dB$ with 60 maximum iterations. Works in [25][26] have shown how the LLR values evolve within the decoding process. Depending upon the operating signal-to-noise ratio (SNR) regime these values will initially fluctuate or enter right away a strictly monotonic behavior.

III. On-the-fly Syndrome Check

A hard-decision vector upon the posterior messages is required after each decoding iteration in order to calculate the syndrome. Syndrome calculation involves the product in equation (1), but this is equivalent to the evaluation of each parity constraint with the corresponding code symbols.

The arguments of each constraint correspond to the hard-decision of each LLR. A non-zero syndrome would correspond to any parity-check constraint resulting in odd parity. This condition suggests that a new decoding iteration must be triggered. The calculation of the syndrome in this way is synonymous to the verification of all parity-check constraints and we refer to this as *syndrome check*.

The typical syndrome check requires a separate memory for the hard-decision symbols and a separate unit for the syndrome calculation (or verification of parity-check constraints), this consumes time in which no decoding is involved. In this context, we use the word *typical* in two senses: one referring to the calculation of the syndrome with stable values and the other referring to the evaluation of the syndrome after the end of a decoding iteration.

A. Proposed Method

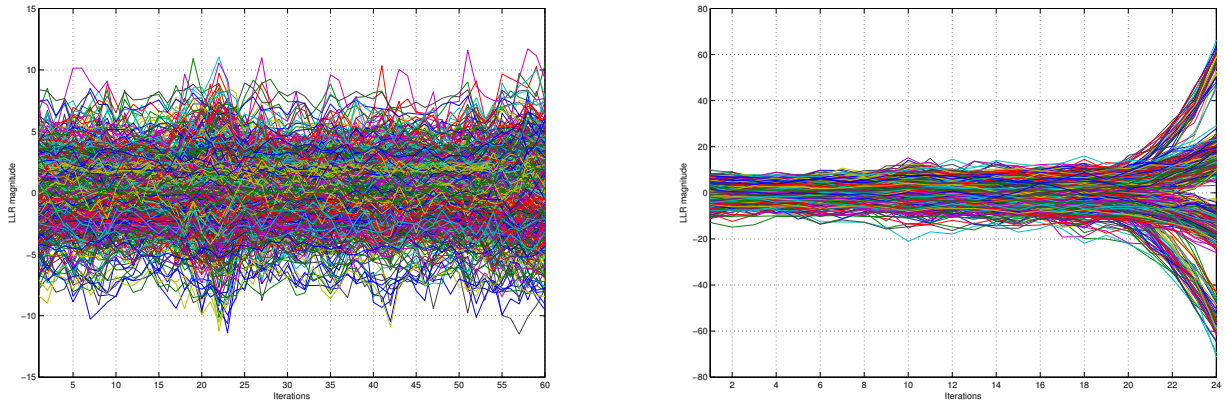
Based upon the behavior of the LLRs illustrated in Figure 4, we propose to perform the syndrome check *on-the-fly* in the following way: each parity-check constraint is verified right after each row is processed. Algorithm 1 outlines the proposed syndrome check within one decoding iteration for a parity-check matrix with M rows.

Algorithm 1 *On-the-fly* syndrome check

1. Decode each row i (or a plurality thereof for parallel architectures)
2. Evaluate each parity-check constraint PC_i by performing the $\oplus$ operation on the hard-decision values
3. Verification:

if $(PC_i = 1)$ **then**
  Disable further parity-checks verification
else
  if $(i = M)$ **then**
    Halt decoding: valid codeblock found
  end if
end if

For the proposed syndrome check there are two extreme cases regarding the latency between iterations. The worst-case scenario corresponds to the case when all individual parity-checks are satisfied but at least one from the last batch to process fails, in which case a new decoding iteration is triggered. The best-case scenario is when at least one of the first rows' parity-check fails, this disables further rows' parity-check verification and the next decoding iteration starts right after the end of the current one. The difference with the typical syndrome check is that it is always performed and it necessarily consumes more time as it involves the check of



(a) LDPC non-converging codeblock

(b) LDPC converging codeblock

Fig. 4. Posterior messages LLRs magnitude evolution.

the entire $\boldsymbol{H}$. Figure 5 shows the timing visualization of these scenarios and the evident source for latency reduction of the decoding task.

The notion of *typical* syndrome check that we use might appear rather naive at first glance, but notice that among all the published works on decoder architectures the way the syndrome is verified is consistently neglected. It could be argued that the syndrome of an iteration can be verified concurrently with the decoding of the following iteration. This indeed would belittle our claim on task speedup (refer to Section IV) but nevertheless the *on-the-fly* syndrome check hardware would still be of considerable lower complexity than said alternative mainly due to the lack of memories to save the hard decisions of the previous iteration.

B. Performance Analysis

A closer examination of the proposed syndrome check reveals the possibility for special scenarios. Indeed, the proposed syndrome check does not correspond to equation (1) since the parity-check constraints are evaluated sequentially and their arguments (LLR sign) could change during the processing of the rows. Consequently, there is a possibility that the decision taken by the *on-the-fly* strategy might not be the correct one at the end of the decoding process. Table I shows the possible outcomes of the decision taken by the proposed strategy in contrast to the typical syndrome check. A *Pass* event is synonymous to the condition $\boldsymbol{S} = \mathbf{0}$. A *false alarm* outcome corresponds to the case when all parity-check constraints were satisfied, indeed halting the decoding task during any iteration as a valid codeblock has been identified (when in fact a final typical syndrome check would fail). On the other hand, a *miss* outcome takes place when during the last iteration (maximum iteration limit) a single parity-check constraint fails rendering the codeblock as invalid (when in fact the typical syndrome check would pass). Both outcomes are the result of at least one LLR sign change right before the last row processing.

From this set of possible outcomes the probability P_H for the proposed syndrome check to be correct can be expressed by:

TABLE I
DECISION OUTCOMES OF THE PROPOSED SYNDROME CHECK

On-the-fly syndrome check	Typical syndrome check	Outcome decision
Pass	Pass	Hit
Pass	Fail	False Alarm
Fail	Pass	Miss
Fail	Fail	Hit

$$\begin{aligned} P_H &= 1-(P_{FA}+P_M) \\ &= 1-(P_P P_{CBE}+(1-P_P)(1-P_{CBE}))\,, \end{aligned} \quad (4)$$

where P_{FA} is the probability of a *false alarm*, P_M is the probability of a *miss* , P_{CBE} is the probability of a codeblock error and P_P is the probability of the proposed syndrome check to pass.

Based upon the analysis and observations in [25][26] the LLRs monotonic behavior is guaranteed for the high SNR regime, in this regime the outcome decision would be a *hit* with probability 1. Nevertheless, as the SNR degrades the inherent fluctuations of the LLRs at the beginning of the decoding process may cause the decision to be a *miss* or a *false alarm* with non-zero probability. In Figure 6, we show the outcome of the decoding of 10^5 codeblocks using the same simulation scenario as in Figure 4 with code length 1944 and two code rates in order to observe the rate at which a *miss* and a *false alarm* may occur on the low SNR regime.

Even though the *hit* rate is shown to be empirically greater than a *miss* or a *false alarm* it is important to address the occurrence of such anomalies. A *miss* result would trigger an unnecessary retransmission in the presence of an ARQ protocol, while a *false alarm* result would introduce undetected codeblock errors. This indeed represents some concerns that must be analyzed on an application-specific context, as for

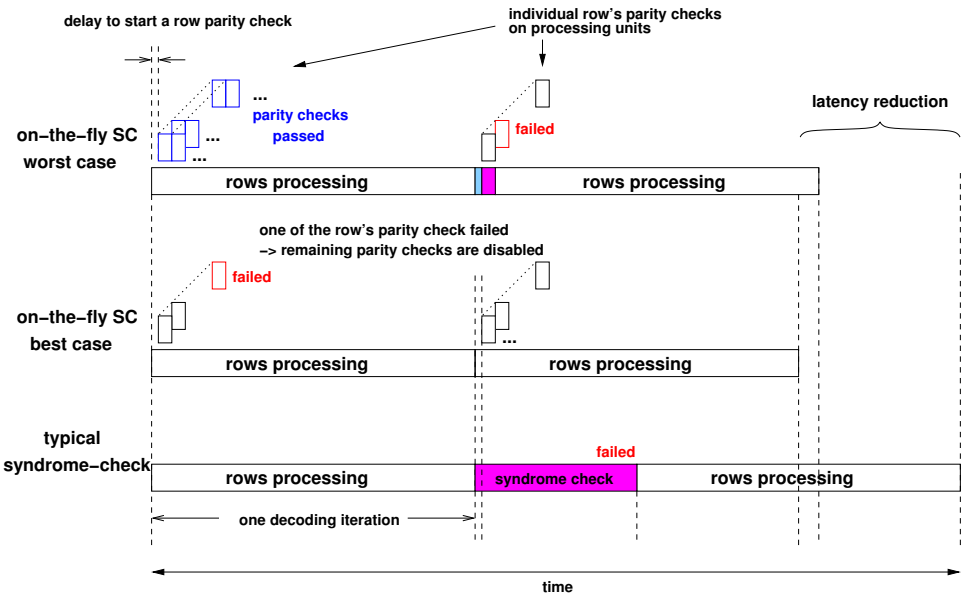
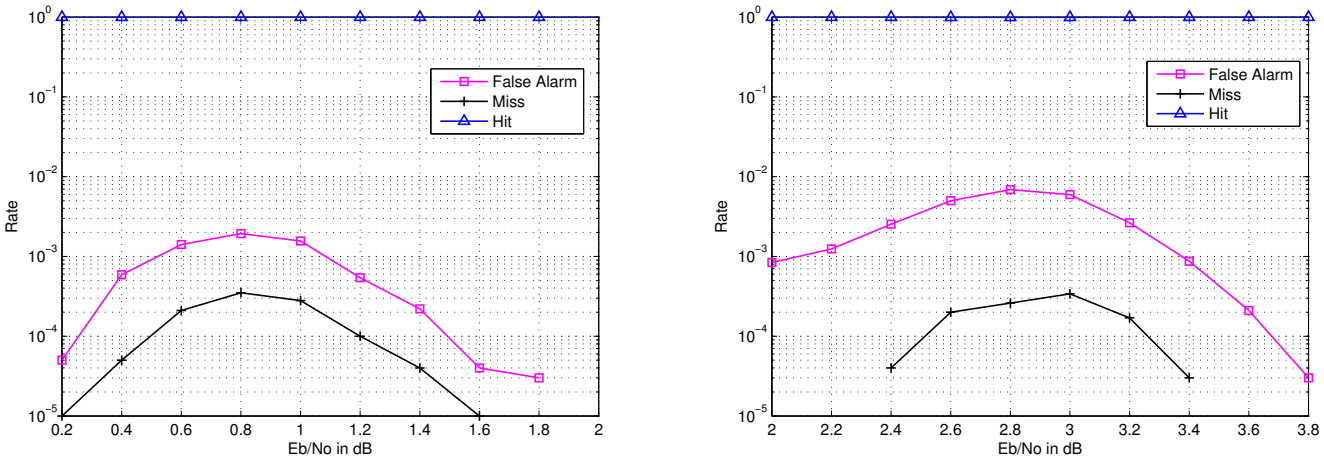


Fig. 5. Timing visualization for two consecutive decoding iterations.



(a) Rate 1/2

(b) Rate 5/6

Fig. 6. Decision outcome rates from the proposed syndrome check for N=1944.

example a wireless modem for [5] is not likely to operate at such low SNR because of the required minimum packet-error rate performance.

The error-correction performance is affected by the *false alarm* outcomes. In Figure 7, we compare the simulated bit-error rate (BER, in solid lines) and frame-error rate (FER, in dashed lines) of the typical syndrome check and the proposed method, this corresponds to the same simulation scenario from Figure 6a. The performance loss is such that an error-floor becomes evident, therefore we address the ways in which this situation can be circumvented.

Detection of the *miss* and *false alarm* outcomes can be performed in two ways:

1) Validating the result provided by *on-the-fly* syndrome check by calculating the typical syndrome check.
2) Allowing an outer coding scheme to detect such conditions: e.g., a cyclic redundancy check (CRC) that typically follows a codeblock decoding.

We propose to detect both *miss* and *false alarm* outcomes by validating the final calculated syndrome (in *on-the-fly* fashion) while executing the first iteration of the following codeblock (CB). Figure 8 depicts both situations. In this way an ARQ protocol can react to a *false alarm* outcome and also avoid an unnecessary retransmission under the presence of a *miss* outcome. The performance is fully recovered, shown in Figure 7 as *validated on-the-fly* syndrome check.

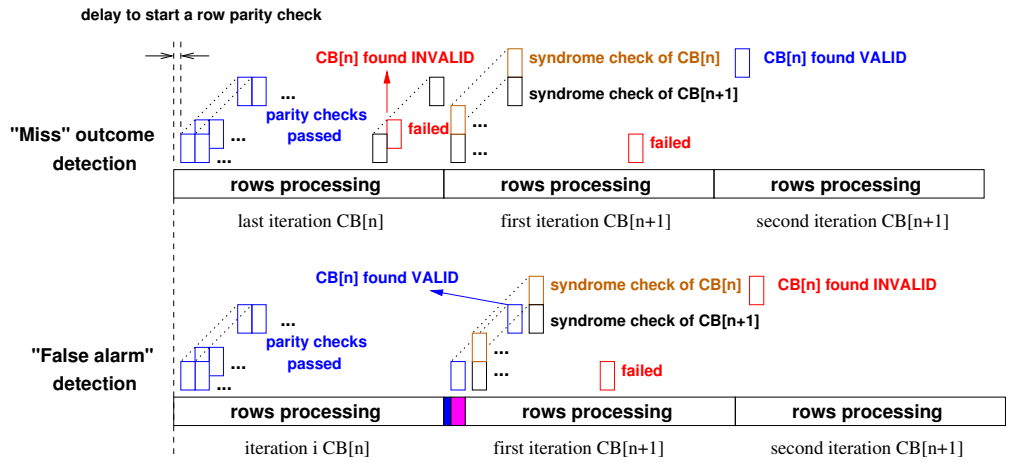


Fig. 8. *False alarm* and *miss* outcomes detection.

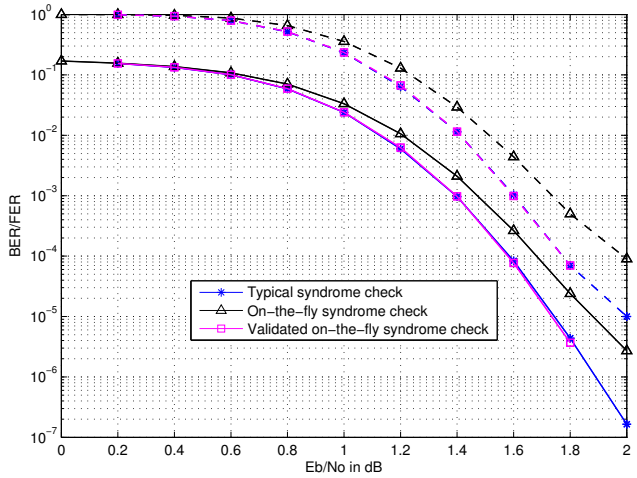


Fig. 7. Error-correction performance comparison.

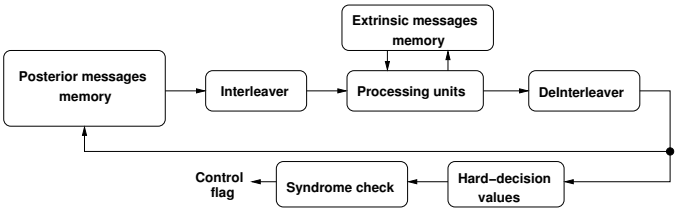


Fig. 9. Canonical decoder architecture.

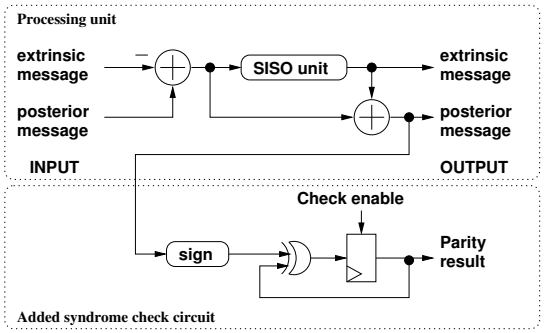


Fig. 10. Processing unit with syndrome check option.

IV. Decoding Task Speedup

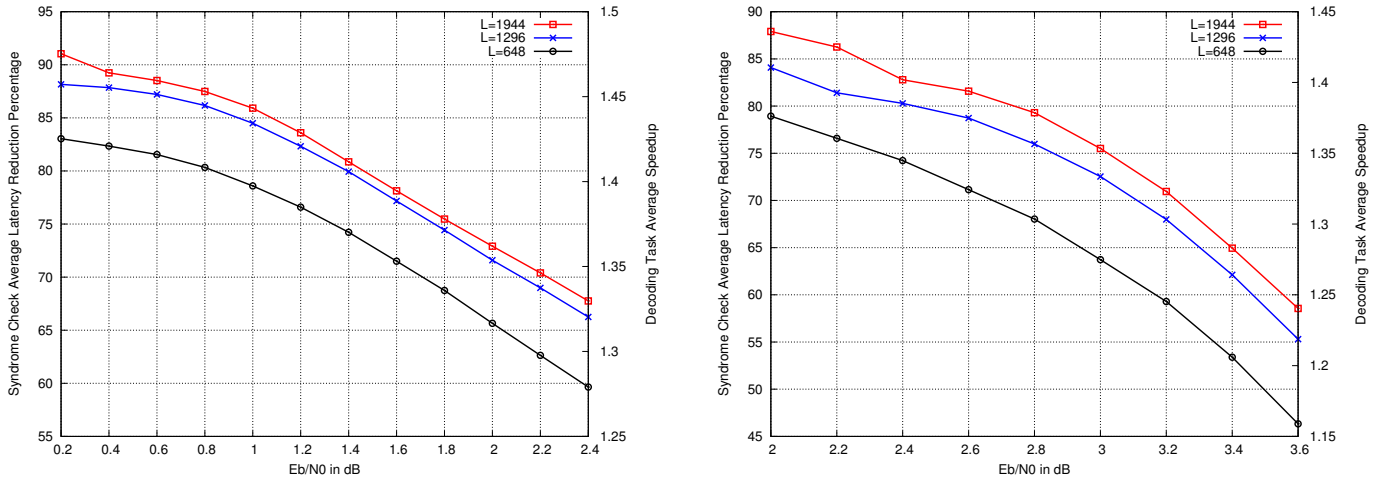
In Figure 9, we show an LDPC decoder with the modules used to perform the typical syndrome check. From this it is evident that the proposed strategy does not require dedicated elements for the syndrome verification. In fact, in order to implement the syndrome check in *on-the-fly* fashion each processing unit is augmented by a marginal set of components. Figure 10 shows a serial processing unit driven by a SISO kernel and the added syndrome check capability. Synthesis results on CMOS $65nm$ technology showed that the area overhead due to the syndrome check capability is only 0.65% for a BCJR-based processing unit (the SISO kernel is the modified BCJR algorithm described in [27]).

If the *false alarm* and *miss* outcomes are to be detected by the proposed method in Section III-B, then the syndrome check circuitry must be replicated and the hard-decision memory in Figure 9 must be kept. Notice that the implementation of *on-the-fly* syndrome check must be evaluated on an application-specific context: decoder operating range, outer multilevel coding schemes and ARQ protocols, logic and memory overheads.

The main benefit of the proposed syndrome check is the speedup of the overall decoding task. The processing latency per decoding iteration for P processing units is given in number of cycles by:

$$\tau_c = m_b \times \frac{Z}{P} \times L_c \ , \qquad (5)$$

where a QC-LDPC code defines $\boldsymbol{H}$ as an array of m_b block-rows of Z rows. In this case P rows are processed concurrently. L_c is the number of cycles consumed during the decoding task where decoding and syndrome verification take place. This value depends upon the number of arguments to process per row, memory access latencies and syndrome verification duration. It is in the latter time duration where our



(a) Rate 1/2

(b) Rate 5/6

Fig. 11. Average latency reduction for the syndrome check process and overall decoding task speedup.

proposal exhibits advantages in terms of speedup. A reduction in the overall task latency improves as well the decoder throughput assuming the arrival time of frames is high enough to provide a 100% decoder utilization:

$$\Gamma = \frac{N \times R \times f_{clk}}{I \times \tau_c} , \qquad (6)$$

where I is the total number of iterations, R the coding rate of the N code symbols and f_{clk} the operating frequency.

The main benefit from the proposed strategy is the reduction in the time consumed during the syndrome check when the decoding process is far from reaching convergence. It could be argued that the syndrome check may very well be disabled during a preset number of initial iterations, but still this tuning must be done offline or shall depend upon extraneous variables like the SNR. Estimating these variables provides sensible overheads. Figure 11 shows the obtained average latency reduction of the syndrome check process compared to the typical one as a function of operating SNR. A total of three use cases with different code lengths L are shown, for a code rate of 1/2 in Figure 11a and code rate of 5/6 in Figure 11b. The low SNR region provides the best opportunities for syndrome check latency reduction since LLRs fluctuate quite often in this region, i.e., a higher decoding effort renders useless the initial syndrome verification.

Indeed, what this strategy is doing is speeding up a portion of the decoding task. With the use of Amdahl's law [28] it is possible to observe the overall speedup of the decoding task based upon the obtained latency reduction of the syndrome check. The overall speedup is a function of the fraction $P_{enhanced}$ of the task that is enhanced and the speedup $S_{enhanced}$ of such fraction of the task:

$$S_{overall} = \frac{1}{(1 - P_{enhanced}) + \frac{P_{enhanced}}{S_{enhanced}}} . \qquad (7)$$

Figure 11 shows as well the average speedup obtained as a function of operating SNR for the same test cases, these results consider that the syndrome check process corresponds to 35% of the overall decoding task per iteration. Amdahl's law provides an upper bound for the achievable overall speedup, 1.53 for this setup. The average speedup is higher for the code rate 1/2 case since the parity-check matrix contains more rows than the code rate 5/6. For the former case the achieved speedup ranged from 84% to 96% of the maximum achievable bound, this corresponds to enhancing the decoder throughput by a factor of 1.28 and 1.48, respectively.

V. Iteration Control

Iterative decoding algorithms are inherently dynamic since the number of iterations depends upon several factors. Proper iteration control policies should identify decodable and undecodable blocks in order to improve on energy expenditure and overall task latency. Convergence of a codeword is detected by verifying equation (1) while non-convergence is usually detected by completing a preset maximum number of iterations.

A. Prior Art

Iteration control techniques (also known as *stopping criteria*) attempt to detect or predict the convergence or not of a codeblock[1] and decide whether to halt the decoding task. This decision is aided by so-called *hard* or *soft* decisions. Hard-decision aided (HDA) criteria are obtained as a function of binary-decision values from the decoding process; on the other hand, the soft-decision aided (SDA) criteria use a non-binary-decision parameter from the decoding process that is compared against threshold values.

The authors in [29] proposed a termination criterion that detects so-called *soft-word cycles*, where the decoder is trapped in a continuous repetition without concluding in a codeblock.

[1] We use the general term *codeblock* instead of *codeword* in order to address decodable and undecodable instances.

This is achieved by storing and comparing the soft-words generated after each decoding iteration. This is carried out by means of content-addressable memories. This criterion saves on average iterations but clearly introduces storage elements.

In [14] a stopping criterion was proposed based upon the monitoring of the variable node reliability (VNR), defined as the sum of the magnitudes of the variable node messages. This decision rule stops the decoding process if the VNR does not change or decreases within two successive iterations. This comes from the observation that a monotonic increasing behavior is expected from the VNR of a block achieving convergence. The criterion is switched off once the VNR passes a threshold value that is channel dependent.

The criterion proposed in [30] is similar to the one in [14], it monitors the convergence of the mean magnitude of the variable node reliabilities. The decision rule uses two parameters tuned by simulations that are claimed to be channel independent.

The authors in [15] proposed a criterion that uses the number of satisfied parity-check constraints as the decision metric. Given the syndrome $\boldsymbol{S} = [s_1, s_2, \ldots, s_M]^T$, the number of satisfied constraints at iteration l is:

$$N_{spc}^{l} = M - \sum_{m=1}^{M} s_m \ . \tag{8}$$

The decision rule monitors the behavior of this metric tracking the increments and their magnitudes as well as the persistence of such behavior. In this rule three threshold values are used, all claimed to be channel independent.

A similar scheme was presented in [16]. This criterion monitors the summation of the checksums of all parity-checks given by:

$$S_p = \sum_{m=1}^{M} P_m \ , \tag{9}$$

where P_m is the checksum of row m as follows:

$$P_m = \bigoplus_{n \in \mathcal{I}_m} c(n) \ , \text{ with } c(n) = \begin{cases} 0 & \text{if } sign(n) > 0 \\ 1 & \text{otherwise} \end{cases} \tag{10}$$

where $c(n)$ is the hard-decision mapping of a soft-input of a row and $\mathcal{I}_m$ is the set of non-zero elements in the mth row. This is indeed the complement of the decision metric used in [15]. The decision rule monitors this metric and uses two threshold values that are dependent upon signal-to-noise ratio to make a decision.

In [17] a channel-adaptive criterion was proposed by monitoring the *sign-changing rate* of the LLRs per iteration. The control rule uses two threshold values that are claimed to be channel independent.

The above control policies have been derived based upon the observation of the characteristic behavior shown by a particular decision metric within the decoding task. The decision metrics used by these control policies are characterized by their dependence or not upon extraneous variables. Estimating these variables (e.g., SNR) raises the implementation effort. In Section VI, we show empirically how the tuning of the parameters used for a decision rule essentially trades off the false alarm rate and missed detection rate of undecodable blocks.

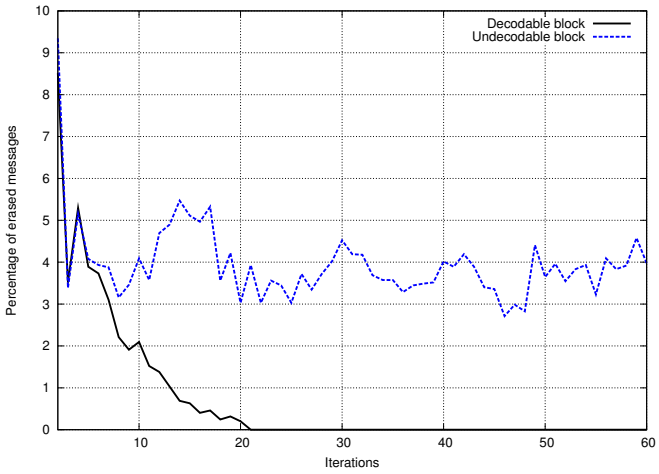


Fig. 12. Percentage of erased messages.

B. Proposed Control Policy

SCMS decoding introduces the concept of *erased messages*, messages which are deemed useless and are discarded after each decoding iteration. A formal treatment behind the concept of *erased messages* can be found in [18], but intuitively the number of messages erased per iteration provides some measure of the reliability (convergence) of the decoding task. For example, the fewer messages erased, the more reliable the decoding task is. Through simulations we observed the total number of erased messages per iteration to identify the possibility to detect earlier an unsuccesful decoding task and also convergence. In the case of an undecodable block the number of erased messages fluctuates around a mean value (dependent upon the SNR), whereas for a decodable block this metric approaches zero relatively fast. In Figure 12, we show how the percentage of erased messages evolves with each decoding iteration for an instance of a decodable and an undecodable block. This corresponds to the decoding of the code defined in [5] with block length 1944 and coding rate 1/2 over the AWGN channel with QPSK modulation, with a maximum of 60 decoding iterations at $E_b/N_0 = 1dB$.

By detecting the characteristic monotonic decreasing behavior of the total number of erased messages when the decoder enters a convergence state, it is possible to save energy on potential undecodable blocks. The *erased messages* metric follows the cumulative quality of the arguments for the parity-check constraints, allowing in fact to observe the dynamics and evolution of the decoding process with fine granularity.

In Figure 13, we show the average number of decoding iterations as a function of SNR for the same simulation scenario of Figure 12 for several stopping rules:

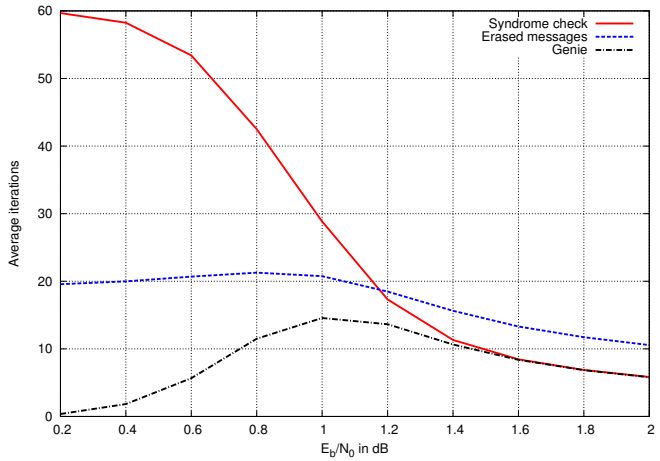


Fig. 13. Average iterations for stopping rules.

1) Syndrome check verification, this corresponds to equation (1).
2) Erased messages metric. Decoding is halted when either the number of erased messages equals zero or a non-convergence condition is satisfied. For non-convergence detection we allow only a fixed number of increments of this metric.
3) Genie. An ideal stopping rule with foreknowledge of the transmitted block, in this case decoding would not even start on an undecodable block.

The syndrome check and the genie criteria correspond to the empirical bounds of any valid stopping rule. From Figure 13 it is clear that the number of erased messages may be used as a decision metric to detect earlier undecodable blocks, but indeed it is not suitable for detecting early convergence since the absence of erased messages within an iteration is not a necessary condition for convergence.

From these observations we use the erased messages metric to detect an undecodable block and the syndrome check for decodable blocks. We devise a stopping rule that follows the evolution of the total number of erased messages by counting the increments of this metric and halting the decoding task once the number of increments exceeds a given threshold T. This threshold is a static parameter that essentially trades error-correction performance and the average number of iterations. Algorithm 2 outlines the proposed decision rule. After the decoding of a row m the number of erased messages ϵ_m is accumulated per iteration in S_ϵ^l. This sum is compared with the one from the previous iteration in order to detect the behavior of the metric as illustrated in Figure 12.

The objective of a stopping criterion can be formulated as the detection of an undecodable block. Thus the possible outcomes of such criterion may be a hit, a false alarm and a missed detection. A false alarm corresponds to the halting of the decoding task that would have been successful in the absence of such stopping rule. This indeed generates unnecessary retransmissions in ARQ protocols. On the other hand, a missed detection represents useless energy expenditure

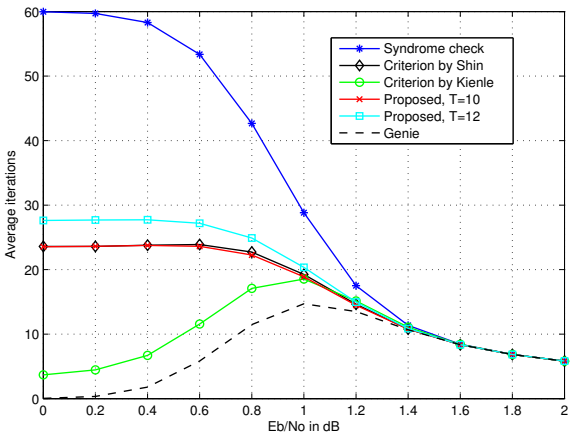
Algorithm 2 Stopping Criterion - SCMS

ϵ_m: number of erased messages in row m
$\mathcal{M}$: set of check nodes
f_s: boolean function for syndrome check, equation (1)
$count \leftarrow 0$; $S_\epsilon^l \leftarrow 0$
for all $iterations$ $1 < l \leq iterations_{max}$ **do**
  for all rows $m \in \mathcal{M}$ **do**
    Decode row m
    $S_\epsilon^l \leftarrow S_\epsilon^l + \epsilon_m$
  end for
  if (f_s) **then**
    Halt decoding (convergence)
  end if
  if $(S_\epsilon^l > S_\epsilon^{l-1})$ **then**
    $count \leftarrow count + 1$
  end if
  if $(count > T)$ **then**
    Halt decoding (non-convergence)
  end if
end for

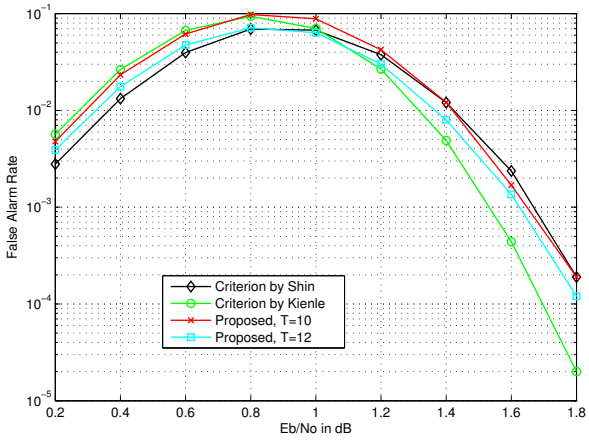
and an unnecessary delay to request a retransmission. Even though any stopping criteria can be tuned to make arbitrarily small the average number of iterations this has an impact on the false alarm rate. In [15] the authors showed empirically how the average number of iterations and the false alarm rate are complementary. We investigated further by looking at the missed detection rate since this indeed can provide hints into a criterion's efficiency. We compared the proposed criterion in Algorithm 2 to the works in [15] (Shin) and [14] (Kienle) along with the syndrome check and the genie rules. In Figure 14, we show the performance comparison in terms of average iterations, false alarm and missed detection rates. We observed that when tuning the stopping criteria to have a similar false alarm rate, as shown in Figure 14b, the missed detection rates exhibit different behaviors. In fact, the proposed criterion showed missed detection rates of several magnitudes of order smaller than the other criteria. The curves for $T = 10$ and $T = 12$ of the proposed criterion are below the value of 10^{-6}, not shown in the figure.

Since it is possible to monitor several decision metrics we investigated how a particular combination may impact the tuning and performance of the resulting *hybrid* control rule. By assisting the decision process with several metrics it is possible to tune the control policy to reach better performance in terms of false alarms, missed detections and average number of iterations. Nevertheless, it was possible to reduce the missed detections only by adding the rule in Algorithm 2. For this reason we propose to enhance the performance of the previous art by adding the number of erased messages per iteration as a second decision metric on an SCMS-based LDPC decoder. Figure 15 shows the proposed hybrid iteration control system.

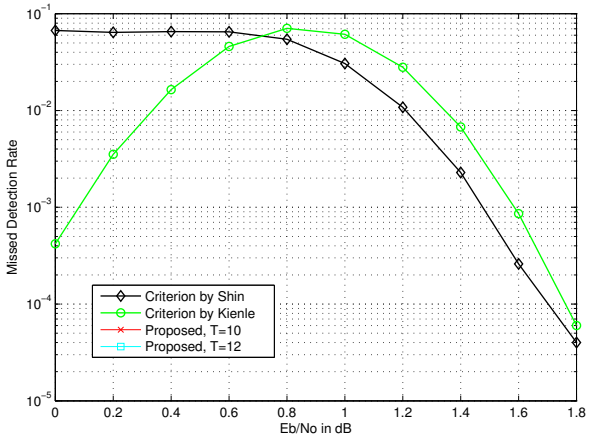
We selected the number of parity-check constraints metric [15] as it offers less computational complexity than the VNR metric [14]. In Table II, we compare the cited stopping rules



(a) Average iterations



(b) False alarm rates



(c) Missed detection rates

Fig. 14. Performance of stopping criteria.

and the one proposed in [17] (Chen) along with Algorithm 2. The number of operations is given as a function of the dimensions of the parity-check matrix. N is usually much

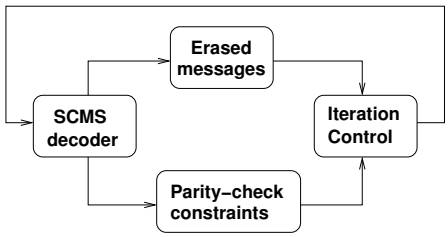


Fig. 15. Hybrid iteration control system.

TABLE II
COMPLEXITY OF DECISION RULES

Criterion	Operations		Tuning Parameters	Data Type
	Compare	Add		
Shin [15]	3	M+3	3	Integer
Kienle [14]	1	N	1	Real
Chen [17]	3	N	2	Integer
Algorithm 2	2	M+2	1	Integer

larger than M (e.g., twice for a rate 1/2 code), this means that on the number of calculations alone the criterion by Kienle is the most complex one. Furthermore, the type of data used by this criterion requires full resolution real quantities, this indeed imposes a more complex datapath (within a VLSI implementation) when compared to the other listed criteria.

Therefore, by observing the performance (error-correction, average iterations, false alarm and missed detection rates) of the mentioned stopping criteria we propose the hybrid iteration control policy for SCMS-based LDPC decoders such that two decision metrics are monitored in order to detect decodable and undecodable blocks. Even though it is possible to monitor all previously proposed decision metrics we found out that the erased messages metric provides the most effective detection for undecodable blocks (in the sense of exhibiting the lowest missed detection rate). In the following, we provide results when utilizing the hybrid technique by using both Algorithm 2 and the criterion in [15] embodied as shown in Figure 15.

VI. STOPPING CRITERIA COMPARISON

All stopping criteria can reduce the average number of iterations depending upon the tuning of the decision parameters used within their control policy. This has consequences of different aspects that are worth investigating. In the following, we tune the stopping criteria in [14][15][17] along with the proposed hybrid control to be used in the SCMS decoding within the simulation scenario described in the previous section.

Figure 16 shows the simulated BER performance for the tested criteria. The stopping criteria can be tuned to be close in performance, for the case of the criterion in [15] (Shin) the parameters used were $\theta_d = 6, \theta_{max} = 4$ and $\theta_{spc} = 825$; for the criterion in [14] (Kienle) $MB = 16$ was used; and for the criterion in [17] (Chen) $lte = 6, thr = 9\%$ were used. The proposed hybrid criterion uses $T = 22$ and the same setup just mentioned for [15].

Figure 17 shows the average number of iterations for the stopping criteria. The syndrome check and the genie are once

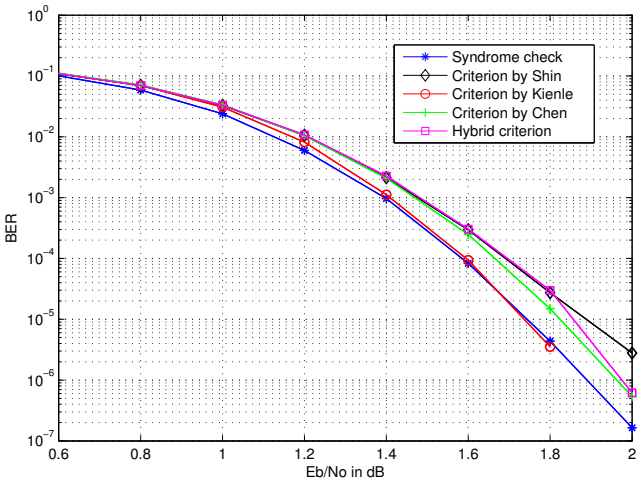


Fig. 16. Error-correction performance for stopping criteria.

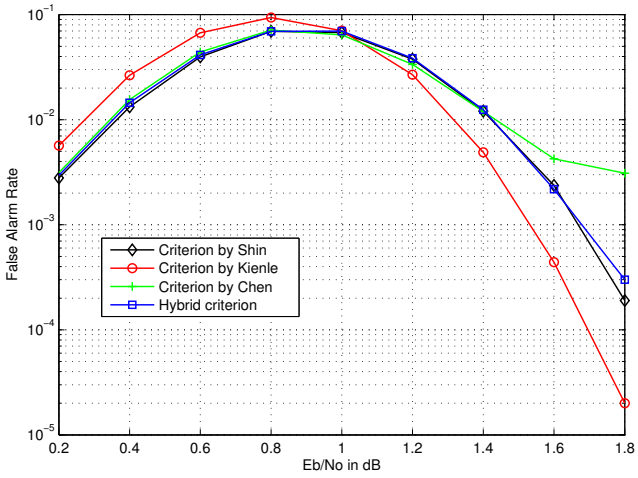


Fig. 18. False alarm rate of stopping criteria.

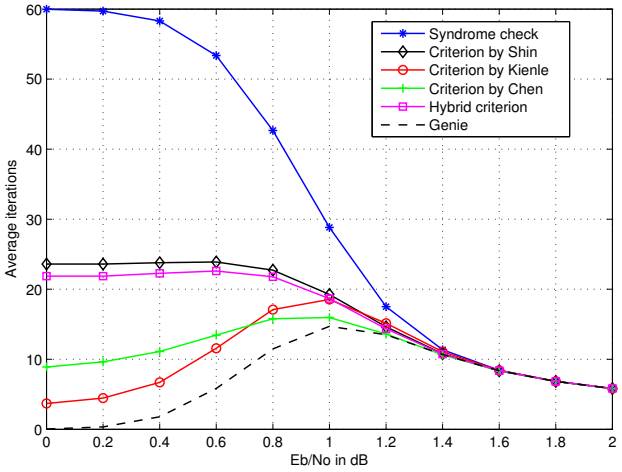


Fig. 17. Average iterations for stopping criteria.

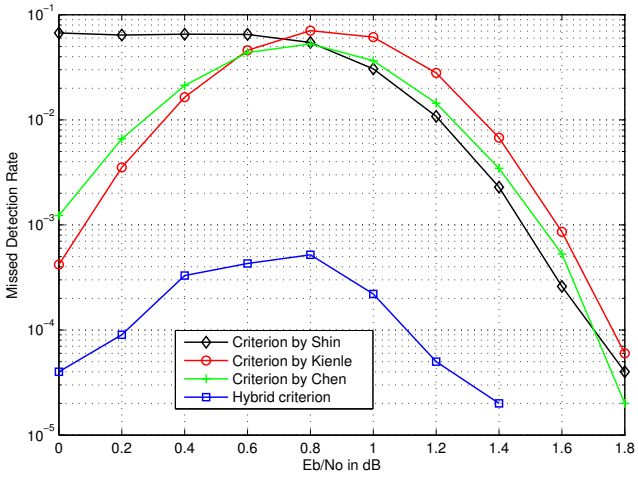


Fig. 19. Miss rate of stopping criteria.

again provided to observe the achievable empirical bounds. Here the tradeoff between average iterations and performance loss is evident. From these figures the criterion by Kienle shows an advantage for a fewer number of iterations in the low SNR region with the smallest performance loss, but this criterion shows the highest false alarm rate (FAR) on the same SNR region.

In Figure 18, we show the FAR of the simulated stopping criteria. This is a relevant figure of merit since the stopping mechanism on its own can be responsible for unnecessary retransmissions. We can observe how the criterion by Kienle shows a smaller number of false alarms on the high SNR region, this is due to the inherent threshold that is used within this criterion to disable the stopping rule, but on the other hand this criterion shows the highest false alarm rate for the low SNR region. The comparison between the proposed criterion and the one by Shin and Chen is much closer and indeed can be tuned to have a similar performance.

So far we can observe that the criterion by Kienle in the low SNR region exhibits the lowest average number of iterations but leads to the highest number of retransmissions. In general, the FAR of these criteria is relatively close, so we proceed to investigate their missed detection performance. Indeed, the missed detection rate (MDR) can provide further insights into which criterion is actually saving energy without incurring into any penalties. Figure 19 shows the MDR for the investigated criteria. The criterion by Kienle performs better than Shin for the low SNR region, but this no longer holds as the SNR increases. The criterion by Chen follows similarly the criterion by Shin. The most relevant result is that the proposed hybrid criterion achieved a MDR at least one order of magnitude below the best of the other ones.

Notice that on the high SNR regime all stopping criteria are irrelevant. A proper receiver design should guarantee the operation of the wireless modem to be within this regime so that a target BER/FER is provided. Nevertheless, in the

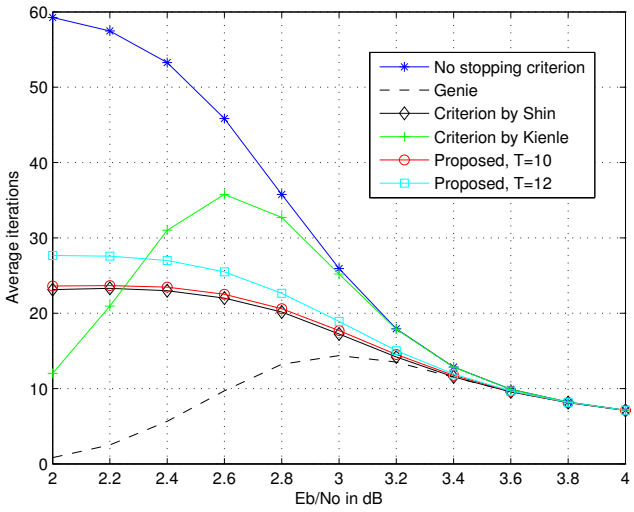


Fig. 20. Average iterations over a fading channel.

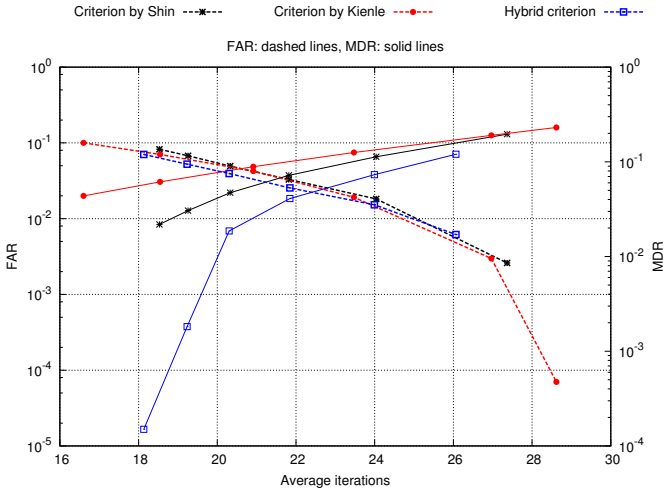


Fig. 21. FAR and MDR for stopping rules with different tuning of parameters.

eventuality that a particular application involves a highly fluctuating channel quality and battery-operated devices these stopping criteria would become relevant. Because of this reason we assert the importance of our study in order to assess the performance of such criteria.

We validated as well the independence of the proposed decision rule and the parameters in Algorithm 2 from the channel characteristics. We applied the criteria in [14][15] to the same LDPC code simulation scenario from Figure 14a on a fading channel where all symbols have independent channel gains that follow a Rayleigh distribution. Figure 20 shows the obtained average number of iterations. For the proposed hybrid policy two values of T were used. It can be observed how all the criteria follow the same behavior as in the AWGN case (refer to Figure 14a) but the criterion proposed by Kienle.

The performance for each stopping criterion depends upon the tuning of the decision-making parameters. In Figure 21, we show the FAR and MDR for different choices of tuning parameters that result in different average number of iterations. These results are from the same simulated scenario for $E_b/N_0 = 1dB$. From this we can observe the tradeoff involving FAR and the average number of iterations for all criteria. In general, the criteria can reduce the average number of iterations but this would result in a higher FAR, this tradeoff must be selected based upon the particular target application (required throughput and allowable retransmissions). Furthermore, we can observe the relationship between MDR and average number of iterations. In this respect the proposed criterion exhibits the best performance. From this figure we can see how a proper tuning of the parameters for a decision rule must consider the relationship between FAR and MDR. FAR refers to the penalty risk introduced by the stopping rule, whereas MDR refers to how effective the stopping rule is for detecting undecodable blocks.

VII. Conclusion

In this paper, we presented two valuable optimizations to enhance the decoding task from a speedup perspective along with low power considerations. The overall speedup in the task allows important throughput gains as well. As a first optimization we proposed an alternative method for performing the syndrome check. By partitioning the calculation among the rows of the parity-check matrix several advantages were identified. *On-the-fly* syndrome check reduces the number of hardware components on a VLSI architecture, offers a speedup in the overall decoding task and improves accordingly the decoding throughput. We analyzed the possible scenarios in which this technique may potentially provide erroneous outcomes regarding the validity of a codeblock and proposed how to handle these cases such that there is no error-correction performance loss. Results from a decoder for the codes defined in IEEE 802.11n provided a speedup of up to a factor of 1.48 at a cost of less than 1% in logic area overhead for a $65nm$ CMOS process.

The second proposed optimization considered the control of iterations. Even though iteration control is relevant only for the low SNR region of the performance curves it is an important technique studied for the purpose of avoiding useless decoder operation. We provided insights into the performance of several control rules in terms of the detection of undecodable blocks as false alarms and missed detections. We proposed a stopping criterion whose control law relies upon the combination of two decision metrics. Motivated by the quasi-optimal error-correction performance of the SCMS decoding kernel, we enhanced the performance of the previous art by adding the number of erased messages per iteration as a second decision metric for proper iteration control. We achieved a notorious decrease in the average number of missed detections for the iteration control policy, making it the best choice in terms of energy efficiency. Furthermore, we showed empirically how the proper tuning of stopping criteria should consider both FAR and MDR in order to accurately assess their performance.

REFERENCES

[1] E. Amador, R. Knopp, V. Rezard, and R. Pacalet, "Hybrid Iteration Control on LDPC Decoders," in *Proc. of 6th International Conference on Wireless and Mobile Communications*, September 2010, pp. 102–106.

[2] E. Amador, R. Knopp, R. Pacalet, and V. Rezard, "On-the-fly Syndrome Check for LDPC Decoders," in *Proc. of 6th International Conference on Wireless and Mobile Communications*, September 2010, pp. 33–37.

[3] R. Gallager, "Low-Density Parity-Check Codes," *IRE Trans. Inf. Theory*, vol. 7, pp. 21–28, January 1962.

[4] D. J. C. MacKay and R. M. Neal, "Good Codes Based on Very Sparse Matrices," in *Cryptography and Coding. 5th IMA Conf., Lecture Notes in Computer Science*, October 1995, vol. 1025, pp. 100–111.

[5] IEEE-802.11n, "Wireless LAN Medium Access Control and Physical Layer Specifications: Enhancements for Higher Throughput," *P802.11n-2009*, October 2009.

[6] IEEE-802.16e, "Air Interface for Fixed and Mobile Broadband Wireless Access Systems," *P802.16e-2005*, October 2005.

[7] IEEE-802.15.3c, "Amendment 2: Millimeter-wave-based Alternative Physical Layer Extension," *802.15.3c-2009*, 2009.

[8] ETSI DVB-S2, "Digital video broadcasting, second generation," *ETSI EN 302307*, vol. 1.1.1, 2005.

[9] DTMB-2006, "Framing Structure, Channel Coding and Modulation for Digital Television Terrestrial Broadcasting System," *Chinese National Standard GB 20600-2006*.

[10] IEEE-802.3an, "Amendment: Physical Layer and Management Parameters for 10Gb/s Operation," *P802.3an-2006*, 2006.

[11] IEEE-802m, "TGm System Requirements Documents (SRD)," *802.16m-10/0003r1*, 2010.

[12] 3GPP LTE-Advanced, "Requirements for Further Advancements for E-UTRA (LTE-Advanced)," *3GPP TSG RAN TR 36.913 v8.0.0*, 2009.

[13] C. Struder, N. Preyss, C. Roth, and A. Burg, "Configurable High-Throughput Decoder Architecture for Quasi-Cyclic LDPC Codes," in *Proceedings of the 42th Asilomar Conference on Signals, Systems and Computers*, October 2008.

[14] F. Kienle and N. Wehn, "Low Complexity Stopping Criterion for LDPC Code Decoders," in *Proc. of IEEE VTC 2005-Spring*, 2005, pp. 606–609.

[15] D. Shin, K. Heo, S. Oh, and J. Ha, "A Stopping Criterion for Low-Density Parity-Check Codes," in *Proc. of IEEE VTC 2007-Spring*, 2007, pp. 1529–1533.

[16] Z. Cui, L. Chen, and Z. Wang, "An Efficient Early Stopping Scheme for LDPC Decoding," in *13th NASA Symposium on VLSI Design*, 2007.

[17] Y.H Chen, Y.J. Chen, X.Y Shih, and A.Y Wu, "A Channel-Adaptive Early Termination Strategy for LDPC Decoders," in *IEEE Workshop on Signal Processing Systems*, 2009, pp. 226–231.

[18] V. Savin, "Self-Corrected Min-Sum Decoding of LDPC Codes," in *Proc. of IEEE International Symposium on Information Theory*, 2008, pp. 146–150.

[19] M. Fossorier, "Quasi-Cyclic Low-Density Parity-Check Codes from Circulant Permutation Matrices," in *IEEE Trans. Information Theory*, 2004, vol. 50, pp. 1788–1793.

[20] M. Mansour and N. Shanbhag, "High-Throughput LDPC Decoders," *IEEE Trans. on VLSI Systems*, vol. 11, no. 6, pp. 976–996, December 2003.

[21] F.R. Kschischang, B. Frey, and H.A. Loeliger, "Factor graphs and the sum-product algorithm," *IEEE Trans. Inf. Theory*, vol. 47, pp. 498–519, Febraury 2001.

[22] J. Chen and M. Fossorier, "Near Optimum Universal Belief Propagation based Decoding of LDPC Codes," in *IEEE Trans. on Comm.*, 2002, pp. 406–414.

[23] E. Amador, V. Rezard, and R. Pacalet, "Energy Efficiency of SISO Algorithms for Turbo-Decoding Message-Passing LDPC Decoders," in *Proc. of 17th IFIP/IEEE International Conference on Very Large Scale Integration*, October 2009.

[24] J. Hagenauer, E. Offer, and L. Papke, "Iterative Decoding of Binary Block and Convolutional Codes," *IEEE Trans. on Inf. Theory*, vol. 42, pp. 429–445, Mar. 1996.

[25] A. Heim and U. Sorger, "Turbo Decoding: Why Stopping-Criteria Do Work ," in *5th International Symposium on Turbo Codes and Related Topics*, 2008, pp. 255–259.

[26] G. Lechner and J. Sayir, "On the Convergence of Log-Likelihood Values in Iterative Decoding," in *Mini-Workshop on Topics in Information Theory, Essen, Germany*, September 2002.

[27] M. Mansour, "A Turbo-Decoding Message-Passing Algorithm for Sparse Parity-Check Matrix Codes," *IEEE Trans. on Signal Processing*, vol. 54, no. 11, pp. 4376–4392, November 2006.

[28] J. L. Hennessy and D. A. Patterson, *Computer Architecture: A Quantitative Approach*, Morgan Kaufmann Publishers, third edition, 2003.

[29] G. Glikiotis and V. Paliouras, "A Low-Power Termination Criterion for Iterative LDPC Code Decoders," *IEEE Workshop on Signal Processing Systems Design and Implementation*, pp. 122–127, November 2005.

[30] Jin Li, Xiao hu You, and Jing Li, "Early Stopping for LDPC Decoding: Convergence of Mean Magnitude (CMM)," in *IEEE Communications Letters*, 2006, number 9.