A Test Vector Generation Method Based on Symbol Error Probabilities for Low-complexity Chase Soft-Decision Reed-Solomon Decoding

Javier Valls, Vicente Torres, Maria Jose Canet, and Francisco M. Garcia-Herrero[*][†][‡]

November 19, 2018

Abstract

This work presents a Low-Complexity Chase (LCC) Decoder for Reed-Solomon (RS) codes, which uses a novel method for the selection of test vectors that is based on the analysis of the symbol error probabilities derived from simulations. Our results show that the same performance as the classical LCC is achieved with a lower number of test vectors. For example, the amount of test vectors is reduced by half and by 1/16 for the RS(255,239) and RS(255,129) codes, respectively. We provide evidence that the proposed method is suitable for RS codes with different rates and Galois Fields. In order to demonstrate that the proposed method results in a reduction of the complexity of the decoder, we also present a hardware architecture for an RS(255,239) decoder that uses 16 test vectors. This decoder achieves a coding gain of 0.56 dB at FER=10^{-6} compared to hard-decision decoding, which is higher than that of an η=5 LCC. The implementation results in ASIC show that a throughput of 3.6 Gbps can be reached in a 90 nm process and 29.1 KXORs are required. The implementation results in Virtex-7 FPGA devices show that the decoder reaches 2.5 Gbps and requires 5085 LUTs.

Reed-Solomon, algebraic soft-decision, Low-complexity Chase, error correction

1 Introduction

Low Complexity Chase (LCC) [1] achieves the same error correction performance with lower complexity, when compared to other Algebraic Soft-Decision Decoding algorithms (ASD) [2] for Reed-Solomon codes [3], *e.g.* bit-level generalized minimum distance (BGMD) decoding [4] or Kötter-Vardy (KV) [2]. The main benefit of the LCC is the use of just one level of multiplicity, which means that only the relationship between the hard-decision reliability value of the received symbols and the second best decision is required to perform the decoding and to exploit the soft information from the channel. This fact has a great impact on the number of iterations and on the global complexity of the interpolation and factorization steps compared to KV and BGMD [5]. Another benefit derived from having just one level of multiplicity is that the interpolation and factorization stages can be replaced by Berlekamp-Massey decoders [6], resulting in a considerable reduction in the total number of operations [7].

Currently, the bottleneck of the LCC decoders is located in the multiplicity assignment and the test vector selection. For the former topic, Peng *et al.* [8] showed that the computation of the symbol reliability values can be performed using bit-level magnitudes. For the latter, efforts on sharing intermediate results between test vectors have been made, but, to the best knowledge of the authors, no alternative proposal to the set of test vectors proposed by Bellorado at [1] has been published. On the other hand, Lin *et al.* [9] proposed a decoder that reduces the complexity by relaxing the criteria for the selection of the best test vector. The resulting decoder requires less area than decoders with worse performance.

[*]J. Valls, V. Torres and M. J. Canet are with the Universitat Politècnica de València, Valencia, Spain. E-mail: {jvalls,vtorres,macasu}@eln.upv.es

[†]F. M. Garcia-Herrero is with the ARIES Research Center, Universidad Antonio de Nebrija, Madrid, Spain. E-mail: fgarciahe@nebrija.es

[‡]This work is funded by the Spanish Ministerio de Economía y Competitividad and FEDER under the grant TEC2015-70858-C2-2-R.

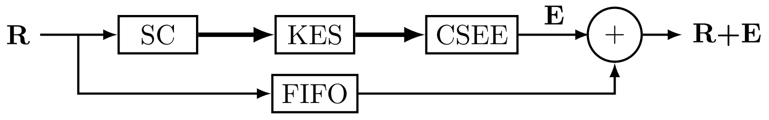


Figure 1: Basic blocks of a conventional RS decoder.

The main contributions of this paper are summarized as follows. First, we analyze the performance of each of the test vectors of the LCC. Then, we propose a method to select a novel set of test vectors, which is based on the symbol error probabilities of the frames that the hard-decision decoder (HDD) does not decode. Although the results we show are mainly obtained for the RS(255,239) and RS(255,129) codes, we provide evidence that the proposed set of test vectors has good performance for different RS codes. Overall, the proposed method achieves the same frame error rate (FER) as the classic LCC and reduces the number of test vectors by a factor between 2 and 16. In order to demonstrate that the proposed method implies a reduction of the overall complexity of the RS decoders, we present a decoder architecture for the implementation of the proposed decoding algorithm for 16 test vectors. We provide implementation results for 90nm ASIC and FPGA that demonstrate that, when compared with conventional LCC decoders, the proposed decoder improves the coding gain/area ratio.

2 RS decoders

In an RS(N,K) code over Galois Field GF(2^m), where N=2^m−1, $2t$ redundant symbols are added to the original K-symbol message to obtain the N-symbol codeword $C(x)$. This codeword is created as $C(x)$=$M(x)\cdot G(x)$, where $M(x)$ is the message polynomial and $G(x)$ is the generator polynomial of the code. After the codeword is transmitted over a noisy channel, the RS decoder receives $R(x)$=$C(x)$+$E(x)$, where $E(x)$ is the polynomial that represents the noise.

As shown in Fig. 1, the RS decoding process begins with the Syndrome Computer (SC) block. This block computes the $2t$ syndromes S_i that are the coefficients of the syndrome polynomial $S(x)$. This is achieved by evaluating the received polynomial in the $2t$ roots of $G(x)$, specifically S_i=$R(\alpha^{i+1})$ for $i \in \{0, 1, \ldots, 2t-1\}$, where α is the primitive element of GF(2^m). The Key Equation Solver (KES) block obtains the error-locator $\Lambda(x)$ and the error magnitude $\Omega(x)$ polynomials by solving the key-equation $\Lambda(x)\cdot S(x)$=$\Omega(x)$ *mod* x^{N-K}. The third and last block is the Chien Search and Error Evaluation (CSEE). The Chien Search finds the error locations, evaluating $\Lambda(x)$ in all the possible positions (*i.e.* $\Lambda(\alpha^{-n})$, for $n\in\{0, 1, \ldots, N-1\}$) and an error evaluation method (*e.g.* Forney's formula) is used to calculate the error magnitude (*e.g.* E_n=$\Omega(\alpha^{-n})/\Lambda'(\alpha^{-n})$) when the Chien search finds an error location, which is whenever $\Lambda(\alpha^{-n})$=0. If the total amount of errors in $R(x)$ does not exceed the error correcting capability t, all the errors in $R(x)$ are corrected by subtracting the error magnitudes from the received symbols.

3 Low-Complexity Chase Decoder

In this work, it is assumed that the codeword $C(x)$ is modulated using binary phase-shift keying (BPSK), transmitted over a Gaussian Noise (AWGN) channel. The RS decoder receives $R(x)$=$C(x)$+$E(x)$, where $E(x)$ represents the noise vector.

The LCC algorithm uses a reliability measure of the received symbols in order to generate a set of test vectors that will be decoded with a HDD. The reliability of a symbol is derived from the *a posteriori* probabilities $p(C|R)$, but applying the Bayes' Law the likelihood function $p(R|C)$ can be used instead. The probability matrix $\mathbf{M}$ is filled with these probabilities for all the received symbols.

$$\mathbf{M}=\begin{bmatrix} p(r_0|0) & p(r_1|0) & \ldots & p(r_{n-1}|0) \\ p(r_0|\alpha^0) & p(r_1|\alpha^0) & \ldots & p(r_{n-1}|\alpha^0) \\ \vdots & \vdots & \ddots & \vdots \\ p(r_0|\alpha^{n-1}) & p(r_1|\alpha^{n-1}) & \ldots & p(r_{n-1}|\alpha^{n-1}) \end{bmatrix} \tag{1}$$

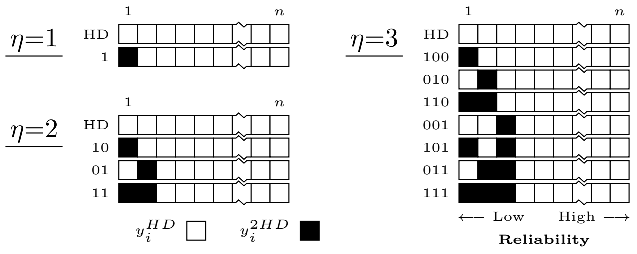


Figure 2: Flipping patterns for the LCC test vectors (η={1, 2, 3}).

Once the reliability matrix is created, the values in column i-th are sorted in order to select the symbols with the largest, y_i^{HD}, and second largest, y_i^{2HD}, probabilities. The reliability measure of symbol r_i is defined as

$$\Gamma_i = log\left[p\left(r_i|y_i^{HD}\right)/p\left(r_i|y_i^{2HD}\right)\right]. \tag{2}$$

The closer Γ_i is to zero, the less reliable r_i is, since the probabilities of being y_i^{HD} or y_i^{2HD} the transmitted symbol are more similar.

Once Γ_i is computed for all the received symbols, those η positions with the smallest values of Γ_i are selected, where η is a positive integer. The LCC decoding process creates 2^η different test vectors by generating all the possible combinations of choosing or not y_i^{2HD} instead of y_i^{HD} for those η symbols (as shown in Fig. 2 for η={1, 2, 3}).

As proposed in [8], y_i^{2HD} is obtained by flipping the least reliable bit of y_i^{HD}.

4 Proposed set of test vectors

In this section, we justify the set of test vectors we propose and we demonstrate its performance for the well-known RS(255,239) code and also for the RS(255,129), which has a rate quite different from the one of the RS(255,239) (0.93 and 0.5, respectively). Moreover we also provide results for other codes that have different rates and GF.

4.1 LCC Analysis

In this section, we analyze the performance of a conventional η=3 LCC decoder. The results we describe below were obtained for the decoding process of the RS(255,239) code, with an AWGN channel, BPSK mapping and Eb/No=7.4 dB. The HDD achieves a FER of $1.125{\cdot}10^{-5}$. In the following analysis we focus on the 210 000 frames that in our simulations were not successfully decoded by the HDD.

Fig. 3(a) shows the percentage of the 210 000 frames that were independently decoded by each one of the 7 test vectors. The test vector with pattern "111" (*i.e.* the one created by flipping one bit in the 3 symbols with the lowest reliability) decodes the highest percentage of frames, 89% of them. The second best test vector corresponds to pattern "100", which decodes only 74% of the frames. On the other hand, if we assume that the test vector with pattern "111" is applied first, Fig. 3(b) depicts the incremental percentage achieved by applying an additional test vector. These results show that the benefit-cost ratio is high for pattern "111" as a test vector, but the use of additional test vectors for η=3 offers a modest benefit.

Since the best test vector for η=3 is the one created by flipping one bit in those 3 symbols with the lowest reliability, we hypothesize that test vectors created by flipping a greater number of symbols, those with the lowest reliability, could achieve higher decoding rates than using just 3 flips. This possibility is analyzed in the following section.

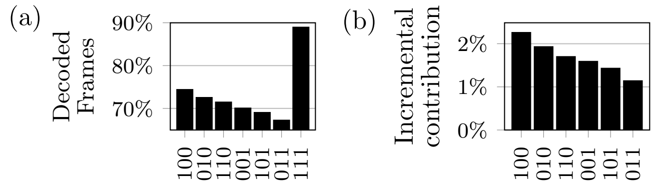


Figure 3: Classical LCC (η=3) for the RS(255,239) code at Eb/No=7.4 dB. (a) Percentage of frames decoded per test vector. (b) Incremental contribution of the indicated test vector to the frames decoded by pattern "111".

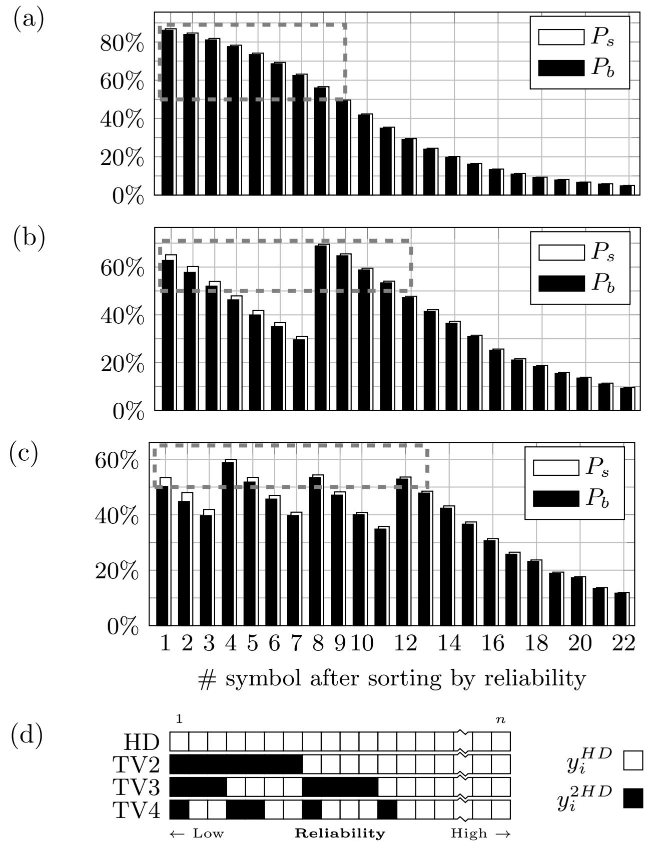


Figure 4: P_b and P_s, as defined in the text, versus symbol position after sorting symbols by their reliability, for the frames that were not decoded at Eb/No=7.4 dB for the RS(255,239). (a) After HDD. (b) After HDD+TV2. (c) After HDD+TV2+TV3. (d) Illustration of the proposed set of test vectors.

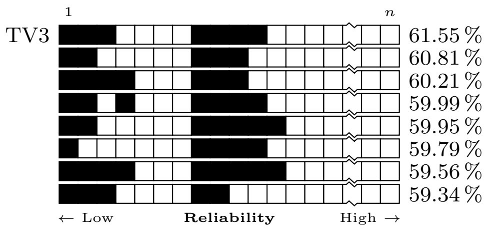


Figure 5: Comparison of TV3 in Fig. 4(d) with its best 7 alternatives. Each test vector succesfully decodes the percentage of the remaining wrong frames (after HDD+TV2) that is shown on its right side.

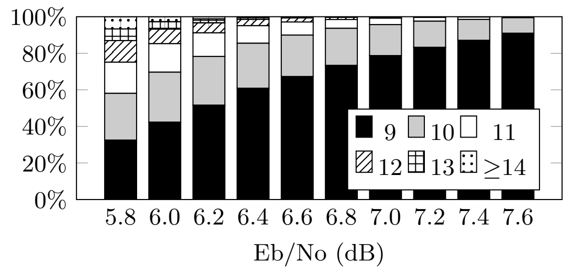


Figure 6: Percentage of frames with the indicated number of wrong symbols, as a function of Eb/No, for frames that were not decoded by the HDD.

4.2 Selection of the second test vector

As in the previous section, we analyze the 210 000 frames that were not decoded by the HDD, in a simulation of the RS(255,239) code at Eb/No=7.4 dB. First, we sort the symbols of the frames according to their reliability and we compute P_s, the probability of having at least one error bit in that symbol, and P_b, the probability of having just one error in that symbol and that this error is specifically in the least reliable bit of the symbol. P_b is, therefore, the probability of fixing a wrong symbol by flipping its least reliable bit.

Fig. 4(a) shows P_b and P_s for the sorted symbols: #1 corresponds to the least reliable symbols of the frames. For the sake of clarity, we only represent values for the 22 least reliable symbols. As shown in the figure, since P_b and P_s are similar, it can be concluded that in most of the cases there is only one error in each symbol and that this error can be fixed just by flipping its least reliable bit. Another interesting conclusion is that the probability of fixing a symbol just by flipping its least reliable bit is higher than 50% for the 8 least reliable symbols of the frames (dashed box in Fig. 4(a)). This is consistent with the fact that the 210 000 frames that the HDD does not decode must have at least t+1 errors, which means at least 9 errors in the target code. Moreover, as shown in Fig. 6, for high Eb/No values, including Eb/No=7.4 dB, most of the frames have only t+1 wrong symbols. This result suggests that, in those cases, a test vector that flips more than t+1 symbols is expected to be counterproductive.

In a situation where we mostly deal with frames that have t+1 wrong symbols, if several of the least reliable symbols are flipped, the test vector will be successfully decoded if the resulting flipping pattern fixes more wrong symbols than correct symbols are spoiled. Since, as we have already seen, the probability of fixing a symbol by flipping its least reliable bit is higher than 50% for the 8 least reliable symbols, a test vector created by flipping around 8 bits (in the least reliable symbols) may result in a high rate of decoded frames. This hypothesis is confirmed in Fig. 7, which shows the percentage of frames successfully decoded for different test vectors created by flipping C_f consecutive symbols, being C_f an integer value between 1 and 16, always those bits with the lowest reliability. It can be seen that in our case the best test vector is the one that performs 7 flips (test vector named TV2 in

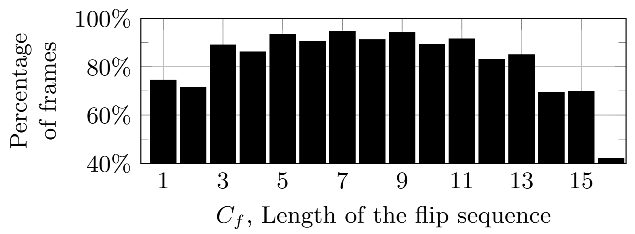


Figure 7: Percentage of decoded frames for test vectors where C_f consecutive least reliable symbols are flipped, for frames that were not decoded by the HDD at Eb/No =7.4 dB.

Table 1: Comparison of the conventional LCC with the proposed LCC with only 2 test vectors that achieve a similar FER

RS code-GF(2^m)	Rate	Eb/No	Proposed		Classical		Reduct.
(n,k)-m		(dB)	C_f	FER	η	FER	factor
(255,129)-8	0.5	6.3	35	$4.5{\cdot}10^{-7}$	5	$4.7{\cdot}10^{-7}$	2/32
(255,167)-8	0.65	6.0	27	$5.1{\cdot}10^{-7}$	4	$5.6{\cdot}10^{-7}$	2/16
(255,205)-8	0.8	6.2	15	$7.3{\cdot}10^{-7}$	3	$8.3{\cdot}10^{-7}$	2/8
(255,239)-8	0.93	7.3	7	$6.3{\cdot}10^{-7}$	2	$6.4{\cdot}10^{-7}$	2/4
(1024,921)-10	0.9	6.4	27	$3.7{\cdot}10^{-7}$	4	$3.5{\cdot}10^{-7}$	2/16
(4095,3687)-12	0.9	6.2	52	$1.5{\cdot}10^{-6}$	5	$1.7{\cdot}10^{-6}$	2/32

Fig. 4(d)): it decodes 94.6% of the frames that were not decoded by the HDD. As we explain above, using 3 consecutive flips is the best pattern in a classical LCC with η=3: it decodes 89.0% of the frames.

It should be noted that in Fig. 7 a pattern with an odd number j of consecutive flips always performs better than j+1 flips. This is to be expected in the situations where most of the frames have t+1 errors, as is the case for high Eb/No (see Fig. 6), and, therefore, the goal of the test vector is to, globally, fix at least one symbol (*i.e.* to correct more wrong symbols than correct symbols are spoiled). In this case, the only way that an additional symbol flip can help to decode a frame that would be otherwise undecoded is when an odd number of flips corrected as much wrong symbols as correct symbols were spoiled. That case can only happen if at least one of the flipped symbols is wrong but there are errors in bits different from its least reliable bit. As we show in Fig. 4(a), for high Eb/No values that is an unlikely situation, since in that case P_b and P_s are quite similar. On the contrary, it is likely that an additional flip will prevent the decodification of the frame by spoiling a correct symbol.

Although Fig. 7 was obtained for a specific RS code and Eb/No, we confirmed a good performance of the proposed method for different RS codes and Eb/No values. This is shown in Table 1, which compares the FER achieved with a) a test vector composed of C_f consecutive flips in addition to the HDD, with b) a conventional LCC dimensioned with an η that achieves a similar FER. For example, for an RS(255,129) code, with a rate of 0.5, the HDD plus a test vector that consists of 35 consecutive flips achieves a similar FER as a conventional LCC with η=5, *i.e.* 32 test vectors. In that case the reduction factor is 2/32.

In the specific case we have utilized to introduce the consecutive-flip pattern, RS(255,239) code at Eb/No=7.4 dB, the 8 least reliable symbols have a P_b higher than 50% of fixing a wrong symbol with a flip. In this case, the best consecutive-flip pattern is the one with 7 flips. We simulated the performance of consecutive-flip patterns for other codes at different Eb/No values, and it can be concluded that the optimal number of consecutive flips is not always exactly the amount of symbols with a P_b higher than 50%, but the result obtained flipping all those symbols whose P_b are above 50% is a good choice that reduces the FER almost as much as the optimal consecutive-flip pattern. This is shown in Fig. 8 for different RS codes and Eb/No values, where we represent the best FER achievable with a consecutive-flip pattern versus FER$_p$, the FER obtained with a consecutive-flip pattern that flips

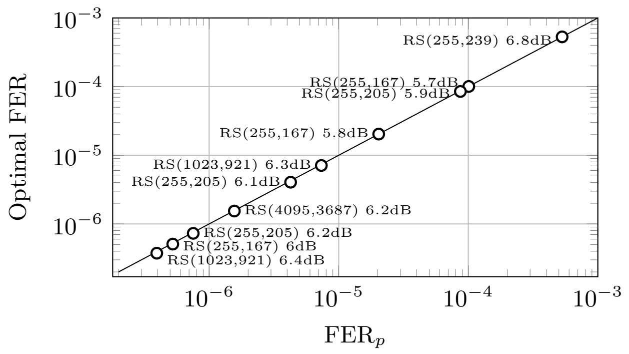


Figure 8: Comparison of the lowest FER achievable with a test vector in addition to the HDD, with FER$_p$, the FER achieved with a flip pattern created by flipping as much symbols as P_b probabilities are above 50% (although restricted to odd lengths).

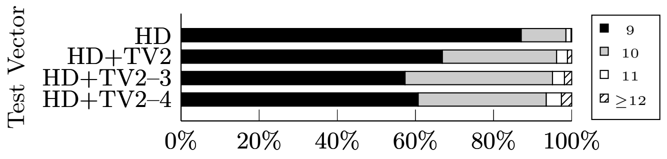


Figure 9: Percentage of frames with 9, 10, 11 or more than 11 errors from those not decoded by the indicated set of test vectors. RS(255,239) code at Eb/No=7.4 dB.

as much symbols as P_b probabilities are higher than 50%.

4.3 Selection of additional test vectors

In this section, we expand the proposed method to select additional test vectors, based on the computed P_b probabilities for the RS(255,239) code at Eb/No=7.4 dB. The purpose of this section is, therefore, to show a way to derive a set of test vectors that can decode a high percentage of the 11 418 frames (out of 210 000) that were not decoded neither by the HDD nor by the TV2 (the 7 consecutive-flip pattern). Fig. 4(b) shows the P_b and P_s probabilities, as defined above, of those 11 418 frames for all the symbols in each frame once sorted by their reliability. In this case, our exhaustive simulations concluded that the best pattern is the one that flips those symbols that have P_b probabilities higher than 50% in Fig. 4(b) (we tested all the combinations of flipping/not flipping the 14 least reliable symbols). The selected flip pattern is named TV3 (see Fig. 4(d)). As shown in Fig. 5, other test vectors achieve a similar decoding performance as the selected TV3. Fig. 4(c) shows P_b and P_s, as defined above, for the remaining 4 390 undecoded frames after applying patterns HD, TV2 and TV3 in the decoder. From those probabilities, pattern TV4 (see Fig. 4(d)) is proposed. Our simulations confirm that no other test vector performs significantly better than TV4 for those frames.

It should be noted that the total number of errors in the frames changes as the patterns are applied, as shown in Fig. 9

4.4 Error correction performance

In this section, we present results for sets of test vectors created following the procedure described above, for the RS(255,239) and RS(255,129) codes. Fig. 10 shows the achieved FER for both codes, evaluated at a range of representative Eb/No values. The proposed set of test vectors for each code is also represented in this figure.

Fig. 10(a) shows that for the RS(255,239) code, the proposed set of vectors is as good as a classical LCC with η=2 with just 2 test vectors (HD+TV2), η=3 with 4 test vectors (HD+TV2–4) and η=4 with 8 test vectors (HD+TV2–8). This results mean

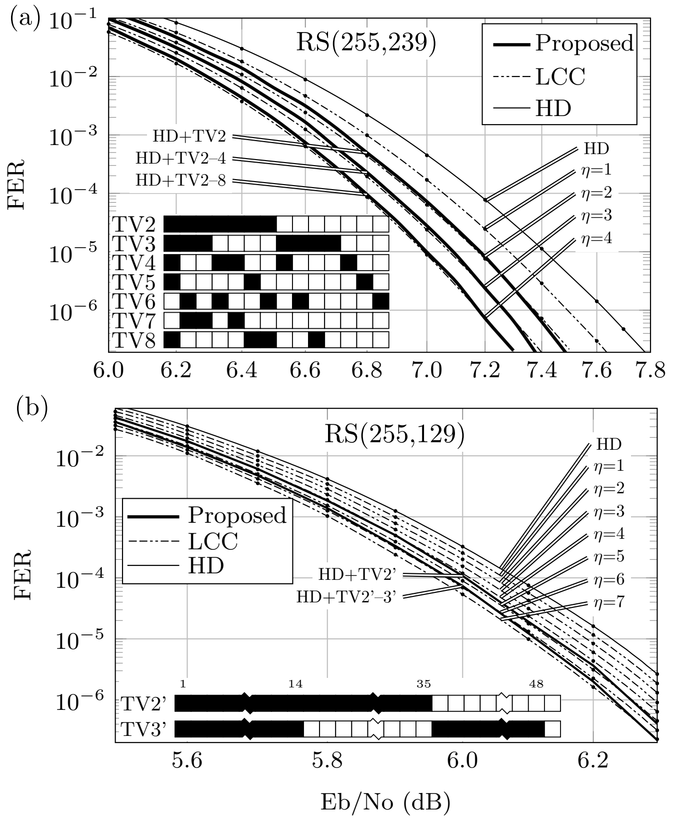


Figure 10: Frame Error Rate (FER) versus Eb/No for (a) RS(255,239) and (b) RS(255,129) codes. The proposed set of test vectors is compared with the results of a classical LCC with different values of η.

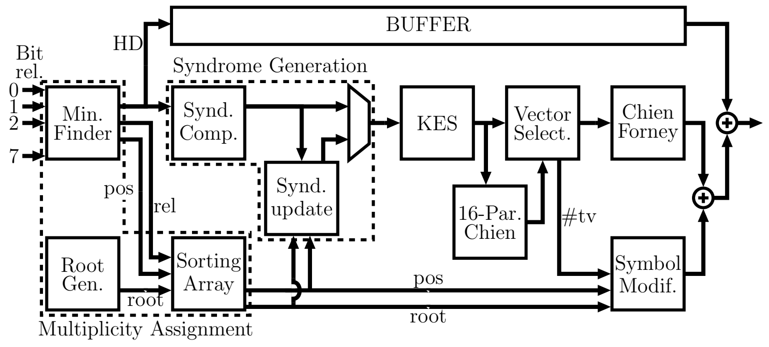


Figure 11: Block diagram for the proposed decoder for 16 test vectors.

around a 50% reduction in the complexity of the Key-Equation Solver (KES) and Chien Search (CS) stages of the decoder. In Section 6 we present detailed data of the impact of this reduction on the overall implementation cost of a decoder.

The same method to select test vectors was applied to the RS(255,129) code at Eb/No=6.2 dB, where the HDD achieves a FER of $1.5{\cdot}10^{-5}$. Fig. 10(b) shows that in this case the set of test vectors is as good as a classical LCC with η=5 with just 2 test vectors (HD+TV2', being TV2' a test vector with 35 consecutive flips, as shown in Fig. 10(b)), and as good as a classical LCC with η=6 with just 3 test vectors (HD+TV2'+TV3', where TV3', shown in Fig. 10(b), has a pattern similar to TV3 from Fig. 4). For example, using 2 test vectors instead of 32 means around a 94% reduction in the complexity of the KES and CS stages of the decoder.

For both RS codes the change in the slope seen for high Eb/No values suggests that the performance of the proposed set of test vectors would improve for Eb/No above those we simulated.

5 Decoder architecture

In this section, we present the architecture for the proposed decoder. This architecture is a generalization of the decoder in [10], in which the test vectors follow a Gray code sequence and the syndromes are computed from the syndromes of the previous test vector. In the decoder we present in this section, the test vectors do not follow a Gray code sequence and the syndromes are computed from the hard-decision syndromes, not from the syndromes of the previous test vector.

5.1 Decoding strategy

In this section, we present the architecture for a soft-decision RS(255,239) decoder that is based on a systolic KES, the enhanced parallel inversionless Berlekamp-Massey algorithm (ePiBMA) [11], that requires $2t$=16 cycles for the computation of each codeword, with low critical path: one adder (T_{add}), one multiplexer (T_{mux}) and one multiplier (T_{mult}). Moreover, the selected KES requires fewer resources than other popular options. If the computation times of the three main pipeline stages are equalized, the KES can be used to compute a set of 16 test vectors, which in our decoder are built following the strategy described in previous sections.

Fig. 11 shows the block diagram of the proposed decoder for 16 test vectors. The decoder is based on the three classical blocks of a HDD: SC, KES and CSEE. Furthermore, more functional blocks are required to manage the 15 additional test vectors. First, in the Multiplicity Assignment stage, the information required to process the test vectors is computed and stored in the output registers of the Sorting Array Block. Specifically, in the Minimum Finder block, a tree of comparators and multiplexers finds the least reliable bit of each symbol. Concurrently, the Sorting Array block, as described below, selects the 14 least reliable symbols of the received codeword, which are sorted and stored for later use. In the Syndrome Generation stage, the Syndrome Computer

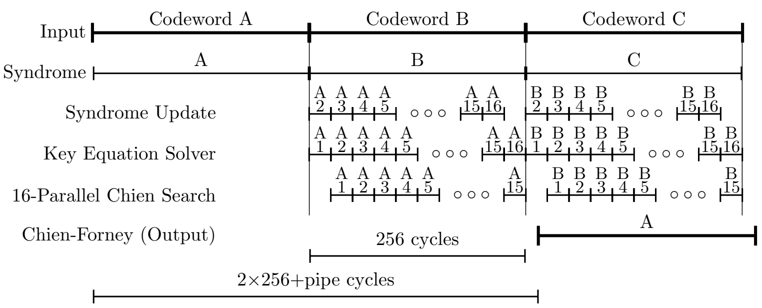


Figure 12: Decoding chronogram of the decoder for 16 test vectors.

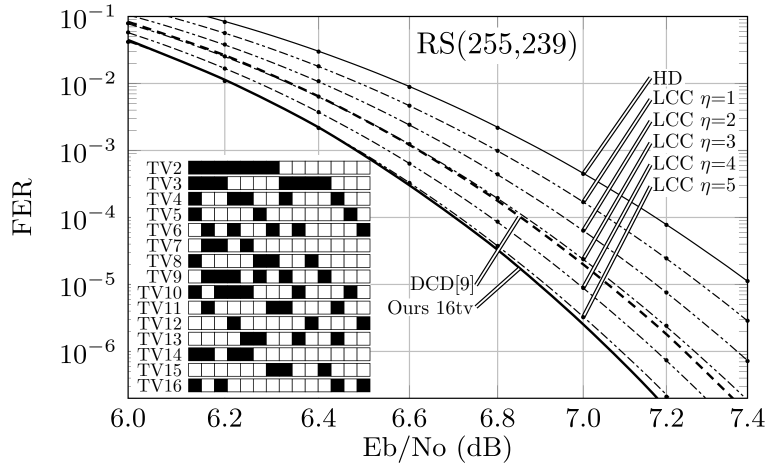


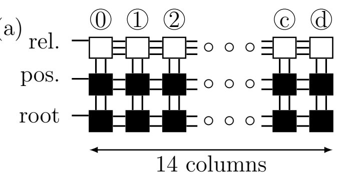
Figure 13: Frame Error Rate (FER) versus Eb/No of the proposed decoder ("Ours 16tv") compared with conventional LCC decoding and the DCD from [9].

block computes the syndromes for the hard-decision test vector and that information is used to obtain the syndromes for the 15 additional test vectors. The KES is fed with the syndromes of a new test vector each 16 cycles. A 16-parallel Chien evaluation block is used to anticipate if those test vectors will be successfully decoded in a full CSEE block. As explained below, this block only processes 15 out of 16 test vectors. After all those computations, the Vector Selection block feeds the CSEE block with the best test vector available.

Fig. 12 shows the decoding chronogram for the decoder. As can be seen, while the KES computes a specific test vector, the Syndrome Update block calculates the syndromes for the next one. At the same time, the 16-parallel Chien polynomial evaluator processes the previous test vector. It should be noted that if test vector #16 were processed by the mentioned block, the output of the decoder would have to be delayed 16 cycles. For that reason, in our decoder that last evaluation is not performed and this test vector is only considered to be the one to be processed by the CSEE block if the order of the error locator polynomial is lower than t (note that this relaxed decoding criteria was proposed in [9] for all of its 32 test vectors).

The test vector selection is performed according to the following rules: 1$^{\text{st}}$) the first vector among the vectors obtained as explained in Section 4 (excluding #16) that accomplishes that the number of errors is equal to the order of the error locator polynomial is the one to be decoded; 2$^{\text{nd}}$) Otherwise, test vector #16 is selected if the order of the error locator polynomial is lower than t. 3$^{\text{rd}}$) Otherwise, the hard-decision test vector is selected.

(a)



(b)

	shift			shift
c	in	s1	s2	out
0	0	X	0	0
0	1	0	1	1
1	0	1	1	1
1	1	0	1	1

(c) (d)

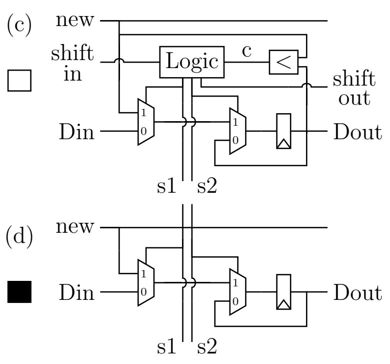


Figure 14: (a) Organization of the Sorting Array block of the proposed decoder. There are 14 columns labeled using hexadecimal digits. Column 0 is the least reliable symbol; (b) Truth table for the Logic subblock in figure (c); (c) Schematic of the blocks in the first row of figure (a); (d) Schematic of the blocks in the second and third rows of figure (a).

The decoding performance of the proposed decoder is shown as “Ours 16tv” in Fig. 13. As expected, the achieved FER is similar to that of a conventional LCC with η=5 using half the amount of test vectors. The coding gain compared to the hard-decision decoding (hereinafter, coding gain) at FER=10^{-6} is 0.56 dB, which is 0.18 dB better than [9]. It should be noted that since the last test vector is supposed to be the one with the lowest correction capability of all the test vectors, the penalty for reducing its full correction capability is negligible.

In the following subsections we describe the blocks that are different from the ones in a conventional LCC decoder.

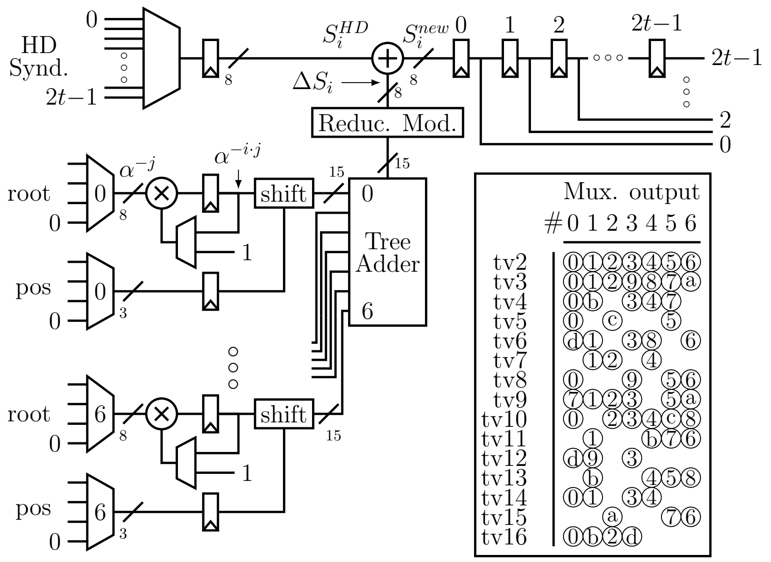
5.2 Multiplicity Assignment

The Minimum Finder block receives the soft magnitudes of the m=8 bits of a symbol and sorts them according to their absolute value [8]. For each symbol of the received codeword this block outputs the hard decision value, the absolute value of the least reliable bit of the symbol and its position in the symbol (a 3-bit value). The goal of the Sorting Array block is to provide all the information required to create the 15 additional test vectors. As can be seen in Fig. 13, the flip patterns that define the set of test vectors were selected with the restriction that only the 14 least reliable symbols can be involved. For this reason, the Sorting Array block is designed to output information about only the 14 least reliable symbols. The information we need to create the test vectors is the position of each one of these 14 symbols in the codeword and the location of their least reliable bit inside of those symbols. In [8] both y_i^{HD} and y_i^{2HD} are sorted and stored for the least reliable symbols. Nevertheless, instead of sorting/storing 28 8-bit values we only sort/store 14 3-bit values that are the positions of the least reliable bits in the symbols. It is unnecessary to store y_i^{HD} and y_i^{2HD} since y_i^{HD} is already stored in the buffer and y_i^{2HD} can be obtained from y_i^{HD} if the position of its least reliable bit, pos_i, is known, according to:

$$y_i^{HD} + y_i^{2HD} = 2^{pos_i}. \tag{3}$$

Moreover, instead of sorting/storing the positions of the symbols in the codewords, for convenience reasons that are explained below, we sort/store the corresponding powers of α created by the Root Generator block.

Fig. 14(a) shows the architecture of the Sorting Array block. In that figure the 14 columns are numbered with hexadecimal digits, from 0 to d. The first row uses the output of the Minimum Finder block to sort the symbols according to their reliability. As explained above, information is stored only about the 14 least reliable symbols of the codeword. The other two rows apply the decisions adopted in the first block of their column and, therefore, store the position of the least reliable bits inside of their symbol and the location of the symbols inside of the codeword. Fig. 14(c) and Fig. 14(d) show the implementation schematic of the basic blocks in Fig. 14(a).



Mux. output #	0	1	2	3	4	5	6
tv2	⓪	①	②	③	④	⑤	⑥
tv3	⓪	①	②	⑨	⑧	⑦	ⓐ
tv4	⓪	ⓑ		③	④	⑦	
tv5	⓪		ⓒ			⑤	
tv6	ⓓ	①		③	⑧		⑥
tv7		①	②		④		
tv8	⓪			⑨		⑤	⑥
tv9	⑦	①	②	③		⑤	ⓐ
tv10	⓪		②	③	④	ⓒ	⑧
tv11		①			ⓑ	⑦	⑥
tv12	ⓓ	⑨		③			
tv13		ⓑ			④	⑤	⑧
tv14	⓪	①		③	④		
tv15			ⓐ			⑦	⑥
tv16	⓪	ⓑ	②	ⓓ			

Figure 15: Schematic of the Syndrome Update block. The table details which symbol positions, identified by the labels introduced in Fig. 14(a), are selected by each multiplexer for each test vector.

It should be noted that once all the symbols are processed, the outcome of this block is registered so the block can be used for the next codeword.

5.3 Syndrome Update

In a classical LCC the test vectors are evaluated following a Gray code sequence, and, therefore, the syndromes for a test vector can be easily created from the syndromes of the previous one [7]. In our case the new syndromes S_i are always created from the HD syndromes S_i^{HD}.

To obtain the new i-th syndrome S_i for test vector x, the value that has to be added to the HD syndrome is:

$$\Delta S_i = S_i^{new} - S_i^{HD} = \sum_{j \in \mathrm{tv}_x} \left(y_j^{new} \cdot \alpha^{-j \cdot i} - y_j^{HD} \cdot \alpha^{-j \cdot i} \right) = $$
$$= \sum_{j \in \mathrm{tv}_x} \left((y_j^{new} - y_j^{HD}) \cdot \alpha^{-j \cdot i} \right) = \sum_{j \in \mathrm{tv}_x} \left(2^{pos_j} \cdot \alpha^{-j \cdot i} \right), \tag{4}$$

where tv$_x$ is the set of all the symbol positions j that belong to the flip pattern of test vector x, and pos_j are the positions of the least reliable bits inside the symbols. For each test vector the sum extends over all the symbols that are flipped with respect to the HD, which are those represented in Fig. 13 with black squares.

In this work we propose a novel architecture for the Syndrome Update block that takes advantage of Eq. 4 and of the fact of storing powers of α to indicate the positions. Fig. 15 shows the schematic of this block. The multiplexers labeled as "root" output α^{-j} for the symbol positions determined by the test vector that is evaluated. The multiplexers labeled as "pos" output pos_j for those symbol positions. Seven multiplexers are required, since in the target set of test vectors the maximum number of flips in a test vector is seven (see Fig. 13). As the inset table in Fig. 15 shows, the data inputs of the multiplexers can be arranged so only 4-input multiplexers are used. The control inputs from the multiplexers are changed each 16 clock cycles, *i.e* when the evaluation

Table 2: Implementation results (XORs)

Block	Proposed	η=4 LCC [10]
Root Generator	55	55
Minimum Finder	599	596
Sorting Array	1568	407
Syndrome Update	1923	830
Vector Selection	1312	1626
Symbol Modif.	430	219
Syndrome Computer	1538	1538
KES	3922	3937
Polynomial Eval.	2255	2242
CSEE	1665	1665
Others	1773	1528
Decoder	17040	14643

Table 3: Implementation results of RS decoders in 90nm CMOS ASICs.

RS(255,239)	Proposed A	Proposed B	η=4 LCC [10]	η=5 DCD [9]
Gate count (K XORs) without/with buffer	17.0/29.1	17.0/29.1	14.6/26.9	22.5/45.3
Chip area (mm^2)/# Metal layers	0.272/8	0.225/8	-	0.216/9
Frequency (MHz)	450^a	320^a	450^a	320^b
Throughput (Gb/s)	3.6	2.6	3.6	2.6
Power consumption (mW)	32.5@450 MHz^c	21.5@320 MHz^c	-	19.6@320MHz^b
Latency (clock cycles)	256×2+18	256×2+18	256×2+34	259×3
Coding gain (dBs@FER)	0.56@10^{-6}	0.56@10^{-6}	0.45@10^{-6}	0.38@10^{-6}
Critical path	$T_{mult}+T_{add}+T_{mux}$	$T_{mult}+T_{add}+T_{mux}$	$T_{mult}+T_{add}+T_{mux}$	$2T_{mult}+2T_{add}+T_{mux}$

^{a}Post-layout result ^{b}Measurement ^{c}Estimated with Encounter

of a new test vector starts. The shift block performs the scaling by 2^{pos_j}. Its output is 15-bit wide. The modulo reduction is performed only once, by the Reduction Modulo block, at the output of the tree adder. Each clock cycle a new syndrome value is updated.

5.4 Vector Selection block

This block selects the test vector whose KES output feeds the CSEE block. For the first 15 test vectors, including the HD, the decision depends on whether the number of errors found by the 16-parallel Chien Search block matches the order of the error locator polynomial. Test vector #16 is selected only if the order of the error locator polynomial is lower than t. Since the latency of the 16-parallel Chien Search block is 21 (which is greater than 16 cycles, the latency of the KES) the Vector Selection block requires that the KES output from the previous test vector is already stored. Therefore, two sets of registers are required to store the current and the previous test vectors. The implementation schematic of this block is shown in Fig. 16.

The Vector Selection block also outputs the identification number of the test vector that is selected.

Table 4: Implementation results of RS decoders in FPGA devices.

RS(255,239)	Proposed		η=4 LCC [10]		η=3 LCC [8]	η=3 LCC [7]
FPGA device	Virtex-7	Virtex-V	Virtex-7	Virtex-V	Virtex-V	Virtex-V
LUTs	5085	5861	4227	5246	7377	5470[a]
Registers	3498	3428	3083	2729	3380	2230[a]
Frequency (MHz)	312.5	166.7	312.5	166.7	134	149.5
Throughput (Gb/s)	2.5	1.3	2.5	1.3	1.0	1.1
Latency (clock cycles)	256×2+18	256×2+18	256×2+34	256×2+34	275×2	275×2
Coding gain (dBs@FER)	$0.56@10^{-6}$	$0.56@10^{-6}$	$0.45@10^{-6}$	$0.45@10^{-6}$	$0.37@10^{-6}$	$0.37@10^{-6}$

[a]Does not include the Multiplicity Assignment stage.

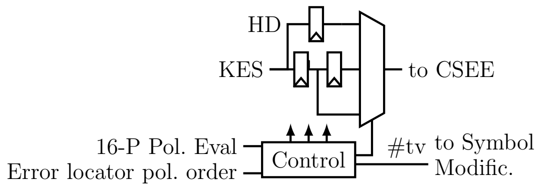


Figure 16: Schematic of the Vector Selection Block for the decoder.

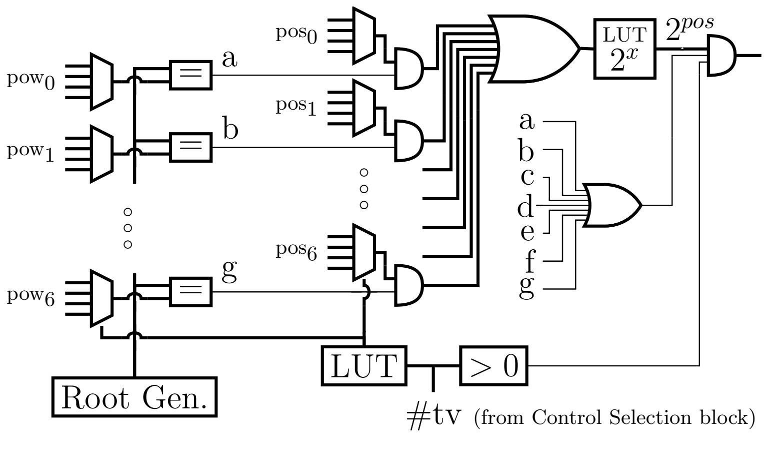


Figure 17: Schematic of the Symbol Modification block.

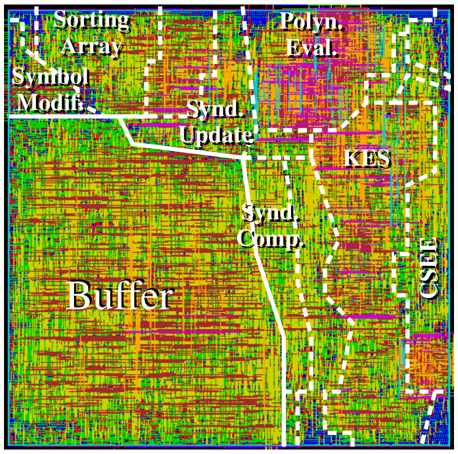


Figure 18: Chip layout of decoder A.

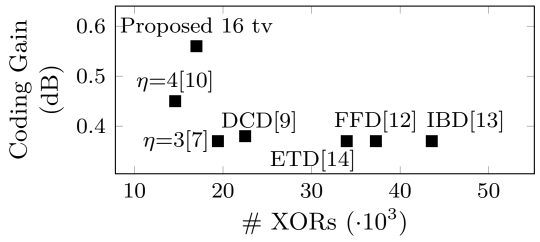


Figure 19: Coding gain at FER$=10^{-6}$ versus implementation cost (#XORs) for ASIC devices. Note: data from the decoders labeled as η=3 [7], FFD [12], IBD [13] and ETD [14] are estimations and do not include the multiplicity assignment stage.

5.5 Symbol Modification block

In a HDD the corrected codeword is created from the received codeword and the error information. But in a LCC decoder the error information is not related to the received codeword, but to the selected test vector. Therefore, in order to create the corrected codeword, first it is necessary to create the test vectors from the received codeword (the hard-decision symbols, which are stored in the buffer). According to the selected set of test vectors, seven is the maximum number of flips required for any test vector (see Fig. 13). The architecture we propose for this block is shown in Fig. 17. The seven multiplexers select the symbols that have to be changed according to the test vector being processed. When the output of the Root Generator matches one of the outputs of a multiplexer, the pattern required to change that symbol is obtained from the position of the bit to be changed inside the symbol. The inset table in Fig. 15 shows the assignment of the inputs of each 4-input multiplexer.

The obtained pattern is added to the hard-decision symbol (from the buffer) and the error magnitude (from the CSEE) to correct the corresponding symbol.

6 Implementation results

A decoder that uses 16 test vectors (see flip patterns in Fig. 13), created with the method explained in Section 4 and that follows the architecture detailed in Section 5, was coded with VHDL and implemented on a 8-metal layer 90 nm CMOS standard-cell technology with Cadence software and also in a Xilinx Virtex-V FPGA device (xc5vlx50t-3) with ISE Foundation software and in a Xilinx Virtex-7 ultrascale FPGA device (xcvu065-3e) with Vivado software. The chip layout is shown in Fig. 18.

In Fig. 19, we compare the gate count (#XORs) and coding gain of the proposed decoders with the results from state-of-the-art soft-decision RS(255,239) decoders, specifically η=3 LCC based on HDD [7], Factorization-Free decoder (FFD) [12], Interpolation-Based decoder (IBD) [13], Decision-Confined Decoder (DCD) [9], Early-Termination LCC Decoder (ETD) [14] and our η=4 LCC [10]. As can be seen, the proposed decoder has a coding gain of 0.56 at FER$=10^{-6}$ with respect to a HDD while the best alternative, the η=4 LCC from [10], has a coding gain of 0.45 (see Fig. 13) but requires 16% less XOR gates. Table 2 shows the detailed gate count in ASICs (given in number of XORs) of the different blocks of the RS(255,239) decoder. In this table we also include data for our η=4 LCC decoder [10], which is the only complete η=4 LCC published to date. As can be seen, compared with this decoder, which also uses 16 test vectors, our new proposal increases the cost of the Sorting Array and Syndrome Update blocks, but the overall cost of the decoder is only increased in 16%. It should be noted that an η=5 LCC based on the architecture from [10] would reach a similar coding gain as the proposed decoder, but it would require additional blocks (1 KES, 1 Syndrome Update and 1 Polynomial Evaluation). According to the data from Table 2, these new blocks would increase at least 48% the area of the decoder. Overall, it can be concluded that the proposed decoder is expected to require at least 31% less area than an η=5 LCC decoder that also uses the decoding architecture proposed in [10] and has slightly worse coding gain (see Fig. 13).

Table 3 shows the implementation results in ASIC devices, for the same RS code, of our decoder and two other decoders: η=4 LCC [10], as in Table 2, and the Decision-Confined decoder (DCD) [9], which processes 32 test vectors with less complexity than a conventional LCC because it relaxes the criteria for the selection of the best test vector. We include in Table 3 results for two different implementation strategies of the proposed decoder: A) optimized for throughput, and B) designed to work at 320 MHz, the frequency of the decoder in [9]. As can be seen, although the proposed decoder A (at 450 MHz) and η=4 LCC from [10] achieve the same throughput (both have the same critical path), the former has higher coding gain. On the other hand, the decoder in [9] requires more XOR gates, achieves lower throughput and has lower coding gain than the proposed decoder A when implemented using the same CMOS process. Note that [8] only gives the area for a 130 nm process but not the number of equivalent XOR gates, while [7], which does not include the Multiplicity Assignment stage, only gives an estimation of the number of XORs, but not the synthesis nor place&route implementations results.

Table 3 also shows chip area and consumption details for the proposed decoder. Our consumption data is obtained with the Static Power Analysis tool of the Encounter software from Cadence. As can be seen, for our decoder B (at 320 MHz), the chip area and power consumption are similar to those of the decoder in [9]. It should be noted that these two decoders are implemented with different Standard-Cell libraries and number of metal layers. Our decoder A (at 450 MHz) requires more area and has higher power consumption than decoder B.

Table 4 shows the implementation results for several decoders in FPGA devices for the same RS code. Compared with the η=4 LCC from [10], when using the same Virtex-V FPGA device, the proposed decoder requires only 12% more LUTs, but has a coding gain of 0.56 dB, which is higher than the 0.45 dB from [10]. On the other hand, the proposals from [8] and [7] require more LUTs, and have lower coding gain than the proposed decoder when implemented in the same Virtex-V device.

7 Conclusion

We present a modified Low-Complexity Chase decoding algorithm that uses a novel set of test vectors, which are derived from the analysis of the symbol error probabilities. Our simulations demonstrate that the proposed set of vectors outperforms the classical LCC set of test vectors for different RS codes and a wide range of Eb/No. In order to show that the proposed method reduces the complexity of the decoder we present an architecture for a RS(255,239) decoder with 16 test vectors. This architecture was implemented in ASIC and FPGA devices and the results show that the ratio between coding gain and area (in XORs or LUTs) is improved when compared with state-of-the-art LCC decoders.

References

[1] J. Bellorado, "Low-complexity soft decoding algorithms for reed-solomon codes," Ph.D. dissertation, Cambridge, MA, USA, 2006.

[2] R. Koetter and A. Vardy, "Algebraic soft-decision decoding of Reed-Solomon codes," *IEEE Transactions on Information Theory*, vol. 49, no. 11, pp. 2809–2825, Nov 2003.

[3] R. E. Blahut, *Theory and practice of error control codes*. Reading, MA: Addison-Wesley, 1983.

[4] J. Jiang and K. R. Narayanan, "Algebraic soft-decision decoding of Reed-Solomon codes using bit-level soft information," *IEEE Transactions on Information Theory*, vol. 54, no. 9, pp. 3907–3928, Sept 2008.

[5] J. Bellorado and A. Kavcic, "Low-complexity soft-decoding algorithms for Reed-Solomon codes –part i: An algebraic soft-in hard-out Chase decoder," *IEEE Transactions on Information Theory*, vol. 56, no. 3, pp. 945–959, March 2010.

[6] F. García-Herrero, J. Valls, and P. K. Meher, "High-speed RS(255, 239) decoder based on LCC decoding," *Circuits, Systems, and Signal Processing*, vol. 30, no. 6, pp. 1643–1669, Dec 2011.

[7] W. Zhang, H. Wang, and B. Pan, "Reduced-complexity LCC Reed-Solomon decoder based on unified syndrome computation," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 21, no. 5, pp. 974–978, May 2013.

[8] X. Peng, W. Zhang, W. Ji, Z. Liang, and Y. Liu, "Reduced-complexity multiplicity assignment algorithm and architecture for low-complexity Chase decoder of Reed-Solomon codes," *IEEE Communications Letters*, vol. 19, no. 11, pp. 1865–1868, Nov 2015.

[9] Y. M. Lin, C. H. Hsu, H. C. Chang, and C. Y. Lee, "A 2.56 Gb/s soft RS (255, 239) decoder chip for optical communication systems," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 61, no. 7, pp. 2110–2118, July 2014.

[10] V. Torres, J. Valls, M. Canet, and F. Garcia-Herrero, "Soft-decision LCC decoder architecture with η=4 for RS(255,239)," in *2018 16th IEEE International New Circuits and Systems Conference (NEWCAS)*, June 2018, pp. 305–308.

[11] Y. Wu, "New scalable decoder architectures for Reed-Solomon codes," *IEEE Transactions on Communications*, vol. 63, no. 8, pp. 2741–2761, Aug 2015.

[12] J. Zhu and X. Zhang, "Factorization-free Low-complexity Chase soft-decision decoding of Reed-Solomon codes," in *2009 IEEE International Symposium on Circuits and Systems*, May 2009, pp. 2677–2680.

[13] F. Garcia-Herrero, M. J. Canet, J. Valls, and P. K. Meher, "High-throughput interpolator architecture for Low-Complexity Chase decoding of RS codes," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 20, no. 3, pp. 568–573, March 2012.

[14] H. Luo, W. Zhang, Y. Wang, Y. Hu, and Y. Liu, "An algorithm for improving the throughput of serial low-complexity Chase soft-decision Reed-Solomon decoder," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 25, no. 12, pp. 3539–3542, Dec 2017.