Soft-Decision Low-Complexity Chase Decoders for the RS(255,239) Code

Vicente Torres [1], Javier Valls [1],* [ID]**, Maria Jose Canet [1]** [ID] **and Francisco García-Herrero [2]** [ID]

[1]   Instituto de Telecomunicaciones y Aplicaciones Multimedia, Universitat Politècnica de València, 46022 Valencia, Spain; vtorres@eln.upv.es (V.T.); macasu@eln.upv.es (M.J.C.)
[2]   ARIES Research Center, Universidad Antonio de Nebrija, 28040 Madrid, Spain; fgarciahe@nebrija.es
*   Correspondence: jvalls@eln.upv.es; Tel.: +34-96-284-9418

check for updates

Abstract: In this work, we present a new architecture for soft-decision Reed–Solomon (RS) Low-Complexity Chase (LCC) decoding. The proposed architecture is scalable and can be used for a high number of test vectors. We propose a novel Multiplicity Assignment stage that sorts and stores only the location of the errors inside the symbols and the powers of α that identify the positions of the symbols in the frame. Novel schematics for the Syndrome Update and Symbol Modification blocks that are adapted to the proposed sorting stage are also presented. We also propose novel solutions for the problems that arise when a high number of test vectors is processed. We implemented three decoders: a $\eta = 4$ LCC decoder and two decoders that only decode 31 and 60 test vectors of true $\eta = 5$ and $\eta = 6$ LCC decoders, respectively. For example, our $\eta = 4$ decoder requires 29% less look-up tables in Virtex-V Field Programmable Gate Array (FPGA) devices than the best soft-decision RS decoder published to date, while has a 0.07 dB coding gain over that decoder.

Keywords: FEC; Low-Complexity Chase; Reed–Solomon; Soft-Decision Decoding

1. Introduction

Reed–Solomon (RS) error-correction codes are widely used in communication and storage systems due to their capacity to correct both burst errors and random errors. These codes are being incorporated in recent 100 Gbps Ethernet standards over a four-lane backplane channel, as well as over a four-lane copper cable [1,2], and for optical fiber cables. Generally, the main decoding methods for RS codes are divided into hard-decision decoding (HDD) and algebraic soft-decision (ASD) decoding. The hard-decision RS decoder architecture consists, commonly, of three main computation blocks: syndrome computation, key equation solver and error location and evaluation. In a different way, ASD algorithms require three main steps: multiplicity assignment (MA), interpolation and factorization. ASD algorithms can achieve significant coding gain at a cost of a small increase in complexity when compared with HDD. Low-Complexity Chase (LCC) [3,4] achieves the same error correction performance with lower complexity, when compared to other Algebraic Soft-Decision Decoding algorithms [5–7] for Reed–Solomon codes [8], like bit-level generalized minimum distance (BGMD) decoding [9] or Kötter–Vardy (KV) [5]. The main benefit of LCC is the use of just one level of multiplicity, which means that only the relationship between the hard-decision (HD) reliability value of the received symbols and the second best decision is required 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 [10–16] compared to KV and BGMD [17]. Another benefit derived from having just one level of multiplicity is that the interpolation and factorization stages can be replaced by Berlekamp–Massey decoders [18], and this results in a considerable reduction in the total number of operations [19].

Recently, Peng et al. [20] showed that the computation of the symbol reliability values can be performed using bit-level magnitudes. They also presented implementation results for a soft-decision decoder that includes the Multiplicity Assignment stage (MAS). On the other hand, Lin et al. [21] 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.

The main contributions of the present paper are as follows. We present a novel MAS based on the one proposed in [20], which sorts and stores less data than in that proposal. We propose also novel implementation schematics for the Syndrome Update (SUS) and Symbol Modification (SMS) stages that are adapted to the proposed MAS. We also propose a scalable architecture for the computation of a high number of test vectors, therefore, it reaches high coding gain. We detail two architectures that use two or four Key Equation Solver (KES) blocks, and follow a Gray code sequence to process the test vectors, so the complexity of the decoder is not increased. Specifically, we present three decoders for soft-decision RS decoding. The first is a η = 4 LCC decoder. The other two, which we call Q5 and Q6, do not decode the complete set of 2^η test vectors of η = 5 and η = 6 LCC decoders, but only a subset of them. The proposed decoders give a solution for the problems created by using a high number of test vectors, since, in that case, the design of the decoder is not as simple as parallelizating resources. We present implementation results for FPGA and CMOS ASIC devices that confirm that our proposals have lower area while they achieve higher coding gain than state-of-the-art decoders.

The organization of this paper is as follows. In Sections 2 and 3 we summarize the background concepts about RS and LCC decoding, respectively. In Section 4 we detail the architecture for the proposed decoders. The implementation results and comparisons are given in Section 5. Finally, in Section 6 we present the conclusions.

2. RS Decoders

In an RS(N,K) code over GF(2^m), where N = 2^m-1, $2t$ redundant symbols are added to the K-symbol message to obtain the N-symbol codeword $C(x)$. After the codeword is transmitted over a noisy channel, the decoder receives $R(x)$ = $C(x)$ + $E(x)$, where $E(x)$ is the noise polynomial.

The RS decoding process begins with the Syndrome Computation (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 the generator polynomial, 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 KES block obtains the error-locator $\Lambda(x)$ and the error magnitude $\Omega(x)$ polynomials by solving the key-equation $\Lambda(x) \cdot S(x)$ = $\Omega(x) \bmod x^{N-K}$. The third 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 subtracting the error magnitudes from the received symbols.

3. Low-Complexity Chase Decoder

We assume that the codeword C is modulated in binary phase-shift keying (BPSK) and transmitted over a Gaussian Noise (AWGN) channel. The LCC algorithm uses the reliability of the received symbols in order to generate a set of test vectors that will be decoded with a HD decoder (HDD). The reliability of a symbol is derived from the *a posteriori* probabilities $p(C|R)$, but instead, the likelihood function, $p(R|C)$, can be used by applying Bayes' Law. The reliability of the received symbol r_i is defined as Γ_i = $log\left[p\left(r_i|y_i^{HD}\right)/p\left(r_i|y_i^{2HD}\right)\right]$, where y_i^{HD} is the symbol with the highest probability of being the transmitted symbol for the i-th received symbol and y_i^{2HD} is the symbol with the second highest probability.

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 symbols 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: all the possible combinations of choosing or not y_i^{2HD} instead of y_i^{HD} for those η symbols. As proposed in [20], y_i^{2HD} is obtained by flipping the least reliable bit of y_i^{HD}.

The Frame Error Rate performance of the RS(255,239) LCC decoder is shown in Figure 1 for $\eta = \{1, 2, 3, 4, 5, 6\}$.
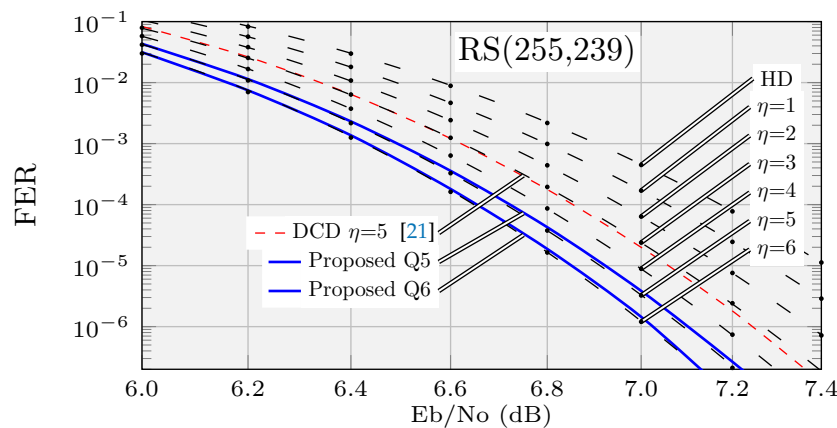

Figure 1. Frame Error Rate (FER) versus Eb/No for RS(255,239) decoders.

4. Decoder Architecture

In this section we present the architecture for three soft-decision RS(255,239) decoders. The first decoder we present is a $\eta = 4$ LCC. The second one, which we refer to as Q5, is a quasi-$\eta = 5$ LCC: it uses all the test vectors of a true $\eta = 5$ LCC, but one. The third one, which we refer to as Q6, is a quasi-$\eta = 6$ LCC: it uses all the test vectors of a true $\eta = 6$ LCC, but four. They are based on a systolic KES, the enhanced parallel inversionless Berlekamp–Massey algorithm (ePiBMA) [22], that requires $2t = 16$ cycles for the computation of each frame, with low critical path: one adder (T_+), one multiplexer (T_x) and one multiplier (T_*). Moreover, the selected KES requires fewer resources than other popular options. If the computation times of the three main pipeline stages are equalized, one KES can be used to compute 16 test vectors, for example for a $\eta = 4$ LCC decoder. For the Q5/Q6 decoders we propose the use of 2/4 KES working in parallel, which increases the decoding capability to 32/64 test vectors.

Figure 2 shows the block diagram for the proposed Q5 decoder. 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 additional test vectors. First, the test vectors have to be created and their relevant characteristics are stored so the rest of the blocks can process those test vectors. A tree of comparators and multiplexers finds the least reliable bit of each symbol. The Sorting Array block, as described below, selects the η least reliable symbols of the received frame, which are sorted and stored for later use. The SC block computes the syndromes for the HD test vector and that information is used to create the syndromes for the additional test vectors. Each KES is fed with the syndromes of a new test vector each 16 cycles. The 16 Parallel Chien Polynomial Evaluation (PCPE) blocks are used to anticipate if those test vectors will be successfully decoded in a full CSEE block. After all those computations, the Vector Selection stage (VSS) feeds the CSEE block with the best test vector available. The vector test selection criteria are as follows: the first vector that accomplishes that the number of errors is equal to the order of the error-locator polynomial is the one to be decoded; otherwise, the HD test vector is selected.

In the case of η = 4, one KES is enough to process all the test vectors and, therefore, the block diagram is the same as in Figure 2 but without KES2, SUS2 and PCPE2. In the case of the Q6 decoder, two more copies of each of those three blocks are required.
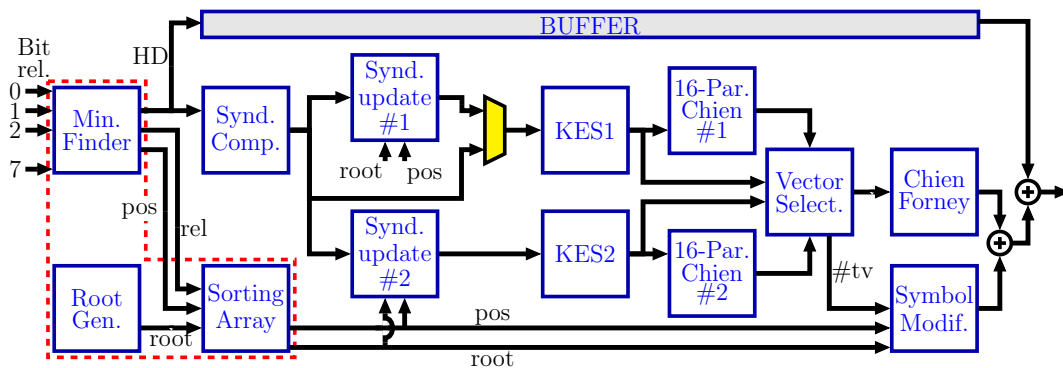

Figure 2. Block diagram for the Q5 decoder.

Figure 3 shows the decoding chronogram for the Q5 decoder. As can be seen, while a KES computes a specific test vector, the corresponding SUS calculates the syndromes for the next one. At the same time, the corresponding PCPE processes the previous test vector. The decoding of a new frame can start every 256 cycles. In this decoder, KES2 must wait 16 cycles until the syndromes for its first test vector (i.e., #31) are available. KES1, on the other hand, works with test vector #0 (HD) during those cycles, since its syndromes are available. If KES2 were to compute 16 test vectors, the latency of the decoder would increase, since the decisions in VSS would be delayed. Moreover, the complexity of the decoder would also increase because the control logic would have to consider decisions for two consecutive frames at the same time. Therefore, Q5 computes 31 out of the 32 possible test vectors of a η = 5 LCC (see Figure 4a). The test vectors that are evaluated by each KES follow a Gray code sequence. This allows the syndromes for a test vector to be easily created from the syndromes of the previous one [19]. The total amount of required operations is reduced, since only one symbol changes from one test vector to the next one. It should be noted that the first test vector evaluated by each KES and the HD frame are different in just one symbol. Note that SUS2 follows the Gray sequence in reverse order, starting with test vector #31. In Q6, for the same reasons explained above, only 16 + 15 + 15 + 15 = 61 test vectors could be decoded. Nevertheless, in order to start the computation in all four KES with a test vector that has only one symbol difference with respect to the HD frame, we compute test vector #31 simultaneously in two KES, as shown in Figure 4b. Therefore, Q6 computes 60 out of the 64 possible test vectors in η = 6. For the η = 4 LCC, the full 4-bit Gray sequence is decoded. As can be observed in Figure 1, the coding gain of decoders Q5 and Q6 is close to that of true η = 5 and η = 6 decoders, respectively.

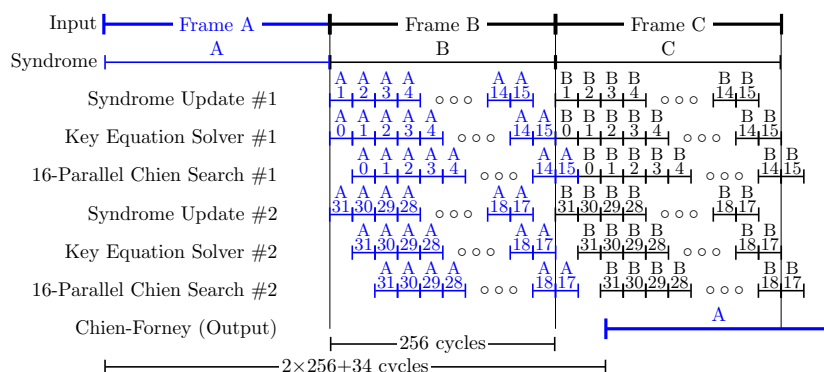In the following subsections, we describe the blocks that are different from the ones in other LCC decoders.

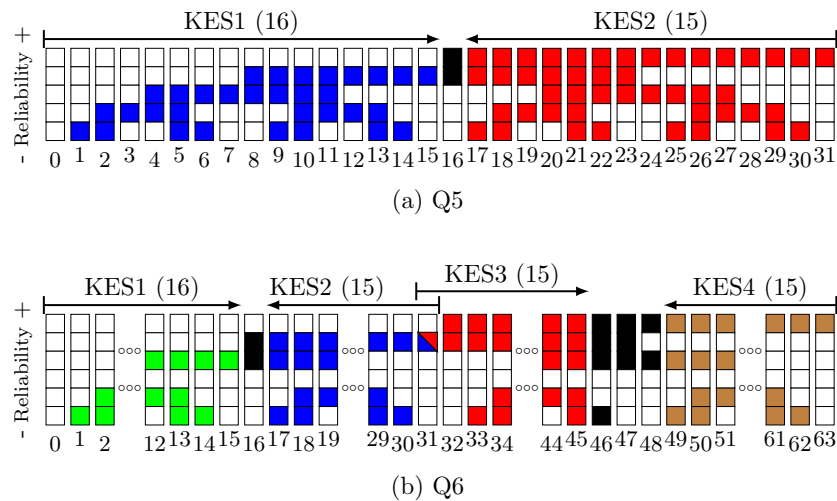**Figure 3.** Decoding chronogram for the Q5 decoder.

(a) Q5

(b) Q6

Figure 4. Test vectors used by the Q5 and Q6 decoders. The arrows show the processing order followed by each KES. ■ = 1, □ = 0.

4.1. Multiplicity Assignment Block

The Minimum Finder block receives the soft magnitudes of the m = 8 bits of a symbol and sorts them according to their absolute value [20]. For each symbol of the received frame 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 additional test vectors. The information we need to create the test vectors is the position of each one of these η symbols in the frame and the location of their least reliable bit inside those symbols. In [20] both y_i^{HD} and y_i^{2HD} are sorted and stored for the η least reliable symbols. Nevertheless, in our proposal, instead of sorting/storing 2η 8-bit values, we only sort/store η 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, since $y_i^{HD} + y_i^{2HD} = 2^{pos_i}$. Assuming that the reliability values are stored with g bits, a total of $(g+22)\cdot\eta$ bit registers are required in our proposal, whereas in [20] $(g+48)\cdot\eta$ bit registers are required.

Moreover, instead of sorting/storing the positions of the symbols in the frames, for convenience reasons that are explained below, we sort/store the corresponding powers of α created by the Root Generator block. Figure 5a shows the architecture of the Sorting Array block. The first row uses the output from the Minimum Finder block to sort the symbols according to their reliability. The other 2 rows of the schematic apply the decisions adopted in the first block of their column and, therefore, store the position of the least reliable bits inside their symbol and the location of the symbols inside the frame. Figure 5b,d show the implementation schematic of the basic blocks in Figure 5a. The pseudocode of the Sorting Array block is shown in Algorithm 1.
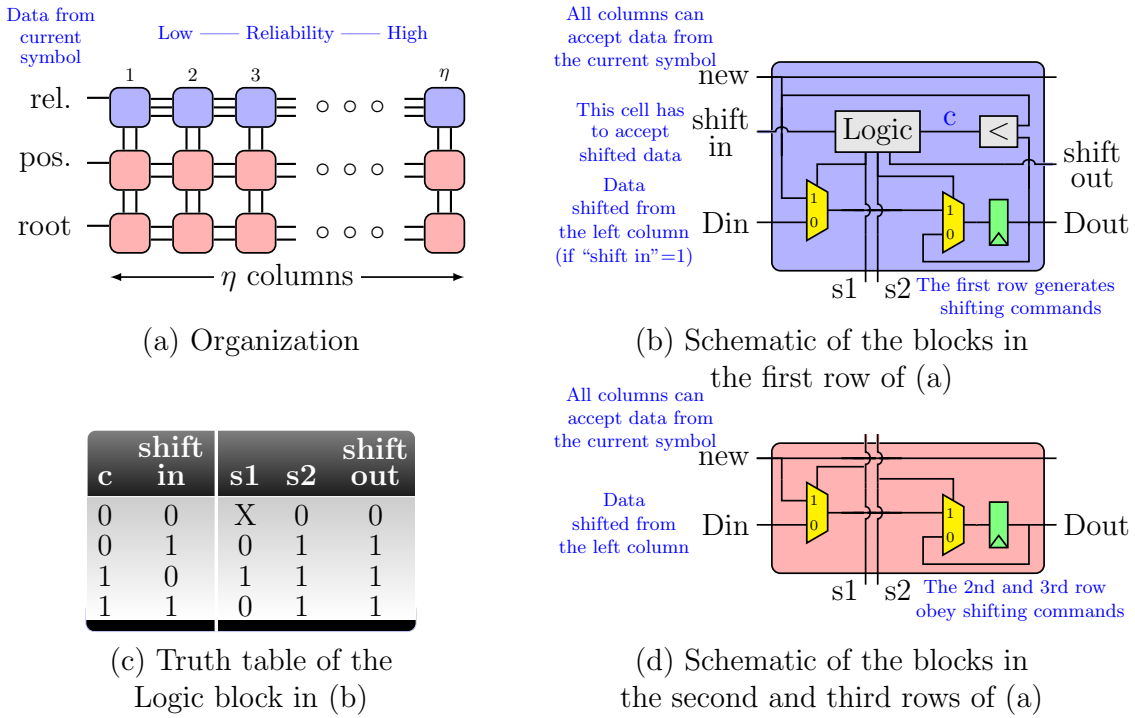
(a) Organization

(b) Schematic of the blocks in the first row of (a)

(c) Truth table of the Logic block in (b)

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

(d) Schematic of the blocks in the second and third rows of (a)

Figure 5. Schematics of the Sorting Array block. Note that the first column is different than the rest, since "Din" and "shift in" inputs do not exist.

Algorithm 1 Pseudocode for the Sorting Array block

Input: For every symbol location $i \epsilon [0, N-1]$ in the frame and its bit $k \epsilon [0, m-1]$, the Min. Finder block provides $\gamma_i = \min\{|r_{i,k}|\}$ and $\delta_i = \underset{k \epsilon [0,m-1]}{arg \min}\{|r_{i,k}|\}$ (where $r_{i,k}$ are the bit-level received voltages)

begin
    Step 1 Sort symbols of the frame according to γ_i.
    Step 2 $\mathbf{s}[1{:}\eta]$ = symbol locations of the η symbols with lowest γ_i
    Step 3
        for $h = 1{:}\eta$　　　　　　　　　　　　　// for all symbols selected in Step 2
            root$[h] = \alpha^{-\mathbf{s}[h]}$　　　　　// root associated with its position in the frame
            pos$[h] = \delta_{\mathbf{s}[h]}$　　　　　　// location of the least reliable bit in the symbol
        end
end
Output: root,pos

4.2. Syndrome Update Block

The value to be added to the previous i-th syndrome, S_i^{prev}, to obtain the new i-th syndrome, S_i^{new}, is:

$$\Delta S_i = S_i^{new} - S_i^{prev} = y_j^{new} \cdot \alpha^{-j \cdot i} - y_j^{prev} \cdot \alpha^{-j \cdot i}$$
$$= (y_j^{new} - y_j^{prev}) \cdot \alpha^{-j \cdot i} = 2^{pos_j} \cdot \alpha^{-j \cdot i}, \tag{1}$$

where j is the position of the symbol in the frame and pos_j is the position of its least reliable bit.

In this work, we propose a novel architecture for SUS that takes advantage of Equation (1) and of the fact of storing powers of α to indicate the positions of the least reliable symbols in the frame instead of their positions itself. Figure 6a shows the schematic of this block. The root multiplexer outputs α^{-j} (selected from root$_1$–root$_\eta$). The pos multiplexer outputs pos_j (selected from pos$_1$–pos$_\eta$). Both values are changed each 16 clock cycles. The shift block scales by 2^{pos_j} and the Reduction Modulo

block computes the modular reduction to the primitive polynomial of the Galois Field. One syndrome is updated each clock cycle. For the first 16 clock cycles, the HD syndromes are used to compute the new syndromes. After that, the syndromes are computed from the syndromes of the previous test vector. The pseudocode of the Syndrome Update block is shown in Algorithm 2.
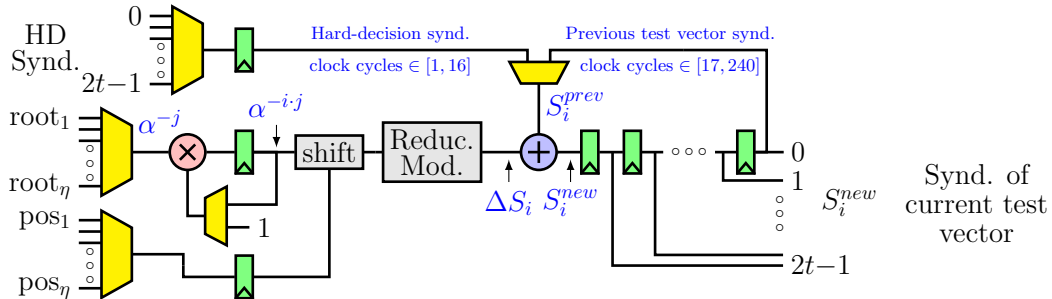
(a) Schematic of the Syndrome Update block.
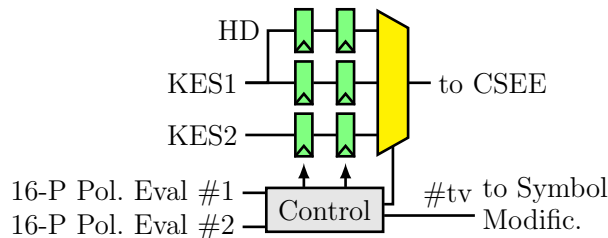
(b) Schematic of the Vector Selection Block for the Q5 decoder.
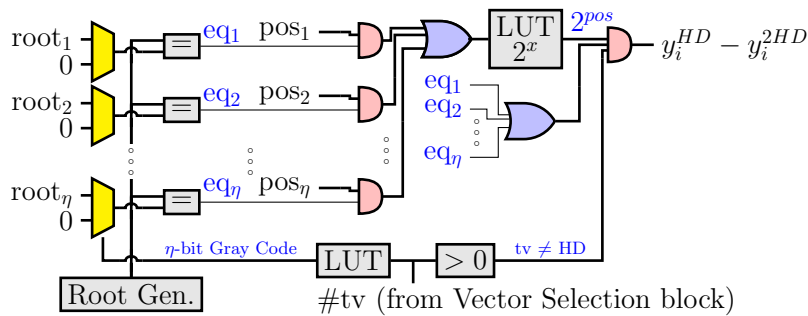
(c) Schematic of the Symbol Modification block

Figure 6. Schematics for different blocks of the proposed decoders.

Algorithm 2 Pseudocode for the Syndrome Update block

Input: root and **pos** from the Sorting Array block, and hard-decision syndromes $\mathbf{S}^{HD}$ from the
　　　　Syndrome Computer
begin
　　for all the test vectors (tv) assigned to this block
　　　　h=index in **root** of the symbol that is different from previous tv
　　　　factor = **root**$[h]$　　　　　　　　　　　　　　　　　　　　// **root**$[h]{=}\alpha^{-j}$
　　　　pos = **pos**$[h]$
　　　　for $i = 1 : 2t$　　　　　　　　　　　　　　　　　　　// for all the syndromes
　　　　　　if tv = first test vector (one with 1 symbol change from HD)
　　　　　　　　$\mathbf{S}^{prev}[i] = \mathbf{S}^{HD}[i]$
　　　　　　end
　　　　　　$\mathbf{S}^{new}[i] = \mathbf{S}^{prev}[i] + 2^{\text{pos}} \cdot \text{factor}$　　　　// factor=$\alpha^{-j \cdot i}$
　　　　　　$\mathbf{S}^{prev}[i] = \mathbf{S}^{new}[i]$
　　　　　　factor = factor $\cdot$ **root**$[h]$
　　　　end
　　　　Output: S^{new}　　　　　　　　　　　　// $2t$ syndromes for the current test vector
　　end
end

4.3. Vector Selection Block

This block selects the test vector whose KES output feeds the CSEE block. The decision depends on whether the number of errors found by the PCPE block matches the order of the error-locator polynomial. Since the latency of the PCPE block is 21 (which is greater than the latency of the KES), VSS requires that the KES output from the previous test vector is still available. Therefore, for each KES in the decoder, two sets of registers are required to store the current and the previous test vectors. On the other hand, since the decision to feed the CSEE block with HD may be delayed beyond the moment a new frame is being received (see Figure 3), the KES output for HD requires also two sets of registers to save the current and the previous frames. VSS also outputs the identification number of the test vector that is selected. The schematic for the VSS of the Q5 decoder is shown in Figure 6b. For the η = 4 LCC the schematic is similar, but there is neither KES2 nor a second polynomial evaluation block. In the case of the Q6 decoder, more registers should be added for the storage of the KES3 and KES4 outputs, just as shown in Figure 6b for KES2.

4.4. Symbol Modification Block

In an HDD, the corrected frame is created from the received frame and the error information. but in a LCC decoder, the error information is not related to the received frame, but to the selected test vector. Therefore, in order to create the corrected frame, first it is necessary to create the test vectors from the received frame (the HD symbols, which are stored in the buffer). The architecture we propose for this block is shown in Figure 6c. The multiplexers select the symbols that have to be changed according to the Gray code. 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 pattern obtained in the Symbol Modification block is added to the HD symbol (from the buffer) and to the error magnitude (from CSEE) to correct the corresponding symbol (see Figure 2). The pseudocode of the Symbol Modification block is shown in Algorithm 3.

Algorithm 3 Pseudocode for the Symbol Modification block

Input: root and **pos** from the Sorting Array block, and
    ntv = number of the selected test vector (tv) from Vector Selection block
begin
    for h=1:η
        root[h]=0 if the symbol associated with that root is not changed
    end
    for i=1:N // for all the symbols in the frame
        d[i] = 0
        for h=1:η // compare with the η roots
            if (ntv<>0)&(α^{-i}=**root**[h])   // no changes if HD is selected
                d[i] = $2^{\textbf{pos}[h]}$
            end
        end
    end
end
Output: d   // differences between y_i^{HD} and y_i^{2HD} for the selected tv

---

5. Implementation Results

The proposed architectures for the η = 4 LCC, Q5 and Q6 decoders were implemented on an eight-metal layer 90 nm CMOS standard-cell technology with Cadence software and also in a Xilinx Virtex-V and Virtex-7 ultrascale FPGA devices with ISE and Vivado software, respectively. The chip layout of the proposed η = 4 LCC decoder in ASIC is shown in Figure 7.
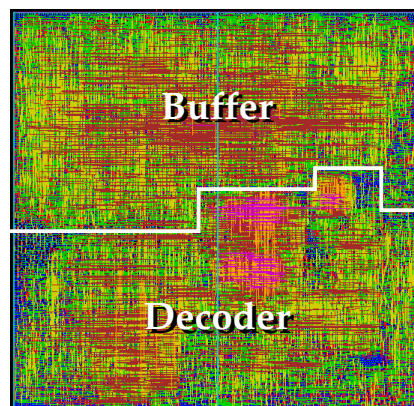

Figure 7. Chip layout of the proposed η = 4 LCC decoder.

In Figure 8, 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, 4, 5} LCC based on HDD (ZHA) [19], Factorization-Free decoder (FFD) [23] and Interpolation-Based decoder (IBD) [24]. As can be seen, our decoders improve the ratio coding gain versus area, when compared with other decoders.

Table 1 shows the detailed gate count in ASIC (given in number of XORs) of the different blocks of the three proposed decoders.
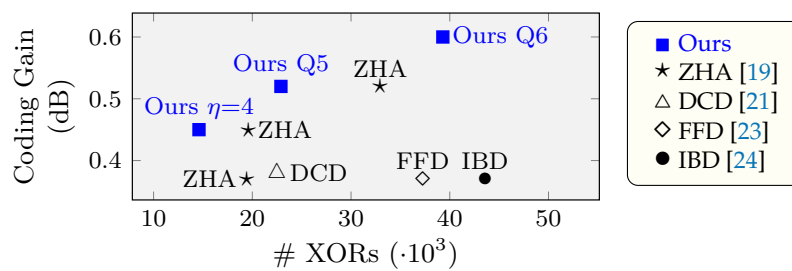
Figure 8. Coding gain at FER = 10^{-6} versus implementation cost (#XORs) for ASIC devices. Note: data from the decoders labeled as ZHA [19], FFD [23] and IBD [24] are estimations and do not include the multiplicity assignment stage.

Table 1. Gate count (#XORs).

Block	Q6	Q5	$\eta = 4$
Root Generator	55	54	55
Minimum Finder	597	596	596
Sorting Array	619	506	407
Syndrome Update	3373	1681	830
Vector Selection	3911	2387	1626
Symbol Modification	304	269	219
Syndrome Computer	1538	1538	1538
KES	15,717	7906	3937
Polynomial Evaluation	9054	4493	2242
CSEE	1664	1665	1665
Others	2480	1852	1528
Decoder (without BUFFER)	39,312	22,947	14,643
BUFFER	12,289	12,281	12,282
Total gate count (#XOR)	51,601	35,228	26,925

Tables 2 and 3 compare, for the same RS code, our proposals and state-of-the-art published decoders, for ASIC and FPGA devices, respectively. On the one hand, Table 2 compares our decoders with [21], the only decoder, to the best of our knowledge, that provides complete implementation results in ASIC, and also with [20]. On the other hand, Table 3 compares our decoders with [20], the only decoder, to the best of our knowledge, that provides complete implementation results in a Virtex-V FPGA device. As can be seen in Table 2, our $\eta = 4$ LCC decoder requires 41% fewer gates in ASIC than [21], whereas it has a 0.07 dB improvement in coding gain at FER = 10^{-6} compared to this decoder. Furthermore, our Q5 decoder has a gate count similar to [21], but has 0.14 dB advantage in coding gain at FER = 10^{-6}. On the other hand, the comparison with [20] is not that straightforward, due to differences in the technology used and the lack of gate count. Nevertheless, the reduction in area is clear when using the same Virtex-V FPGA device. As shown in Table 3, in this case, our $\eta = 4$ LCC decoder reduces the LUT count in [20] about 29 %. Moreover, Ref. [20] has lower coding gain, since this is a $\eta = 3$ LCC decoder. Our decoder $\eta = 4$ LCC has similar area to the $\eta = 3$ LCC decoder in [19], but it should be noted that [19] does not include the multiplicity assignment stage.

In regard to latency and throughput results, as can be seen in Tables 2 and 3, our decoders reach 450 MHz thanks to the low critical path, which is $T_* + T_+ + T_x$. The throughput of our $\eta = 4$ LCC and Q5 decoders in ASICs is $255 \times 8 \times 450 \times 10^6 / 256 = 3.58$ Gb/s. Since the decoder from [21] has longer critical path and higher computational latency than our decoders (259 versus 256 cycles), the potential throughput that it can achieve is slightly lower than ours. On the other hand, since the decoders from [19,20] have the same critical path as ours but slightly higher computational latency than ours (275 versus 256 cycles), it is expected that our throughput is slightly higher: 1.3 Gb/s versus 1.0 Gb/s [20], for the same FPGA device, as can be seen in Table 3. Additionally, the proposed decoders

have a latency of 546 cycles, whereas the DCD [21] requires 777 cycles and the decoders from [19,20] require 550 clock cycles plus the pipeline.

Table 2 also shows chip area and consumption details for the proposed decoders. Our consumption data are obtained with the Static Power Analysis tool of the Encounter software from Cadence. It should be noted that the proposed decoders are implemented with different Standard-Cell libraries, number of metal layers and supply voltage compared to [21]. For comparison purposes, we also implemented an η = 4 LCC decoder optimized for area and working at 320 MHz (the same clock frequency as the decoder in [21]). For that implementation, the chip area is 0.268 mm^2 and the estimated power consumption is 21.3 mW, which are similar to those of the decoder in [21].

Table 2. Implementation results of RS decoders in CMOS ASICs.

RS(255,239)	Ours Q6	Ours Q5	Ours η = 4 LCC	η = 5 DCD [21]	η = 3 LCC [20]
Process (nm) @ Supply Volt. (V)	90@1.2	90@1.2	90@1.2	90@0.98	130@-
Chip area (mm^2)/# Metal layers	0.632/8	0.435/8	0.336/8	0.216/9	0.332/-
Gate ct. (kXOR) no/with buffer	39.3/51.6	22.9/35.2	14.6/26.9	22.5/45.3	-/-
Frequency (MHz)	446 *	450 *	450 *	320 †	220
Throughput (Gb/s)	3.55	3.58	3.58	2.56	1.6
Latency (clock cycles)	256 × 2 + 34	256 × 2 + 34	256 × 2 + 34	259 × 3	275 × 2 [§]
Power consumpt. (mW@MHz)	62.2@446 [‡]	32.1@450 [‡]	28.8@450 [‡]	19.6@320 †	-
Coding gain (dBs@FER)	0.60@10^{-6}	0.52@10^{-6}	0.45@10^{-6}	0.38@10^{-6}	0.37@10^{-6}
Critical path	$T_* + T_+ + T_X$	$T_* + T_+ + T_X$	$T_* + T_+ + T_X$	$2T_* + 2T_+ + T_X$	$T_* + T_+ + T_X$

* Post-layout result. † Measurement. ‡ Estimated. § Does not include latency from MAS, SMS and Chien-Forney stages.

Table 3. Implementation results of RS decoders in a Virtex-V XC5vlx50t-3 FPGA device.

RS(255,239)	Ours Q6	Ours Q5	Ours η = 4 LCC	η = 3 LCC [20]	η = 3 LCC [19]
LUTs	16,049	8914	5246	7377	5470 *
Registers	6362	3966	2729	3380	2230 *
Frequency (MHz)	166.7	166.7	166.7	134	149.5
Throughput (Gb/s)	1.3	1.3	1.3	1.0	1.1
Latency (clock cycles)	256 × 2 + 34	256 × 2 + 34	256 × 2 + 34	275 × 2 [§]	275 × 2 [§]
Coding gain (dBs@FER)	0.60@10^{-6}	0.52@10^{-6}	0.45@10^{-6}	0.37@10^{-6}	0.37@10^{-6}
Critical path	$T_* + T_+ + T_X$	$T_* + T_+ + T_X$	$T_* + T_+ + T_X$	$T_* + T_+ + T_X$	$T_* + T_+ + T_X$

* Does not include the MAS. § Does not include latency from MAS, SMS and Chien-Forney stages.

More up-to-date FPGA device implementation results are shown in Table 4. As can be seen, in this technology our decoders reach 2.5 Gb/s.

Table 4. Implementation results of RS decoders in a Virtex-7 FPGA devices.

RS(255,239)	Ours Q6	Ours Q5	Ours η = 4 LCC
LUTs	13,343	7372	4227
Registers	7223	4480	3083
Frequency (MHz)	312.5	312.5	312.5
Throughput (Gb/s)	2.5	2.5	2.5

It should be noted that in the comparison with state-of-the-art decoders, the coding gain performance of other decoders [19–21] is that of an η = 3 LCC decoder. The decoders we propose are the first ones that use 16 or more test vectors with their full-decoding capabilities. Zhang et al. [19] give estimation results for η = 4 and η = 5 using their architecture: 19,594 and 32,950 XOR gates, respectively. The hardware requirement of the decoder is reduced to 14,643/19,594 = 75% and 22,947/32,950 = 70%, respectively. On the one hand, it should be noted that their estimation does not include the Multiplicity Assignment nor the Symbol Modificaction stages. On the other hand, these authors estimate the cost of the η = 5 LCC decoder assuming that the design of this decoder

only requires the parallelization of specific resources (i.e., syndrome update, Key Equation Solver and polynomial evaluation), but, in the case of $\eta = 5$ and $\eta = 6$, the use of a Gray code sequence in the decoding process is not straightforward. We propose a solution for this issue in the present work. Moreover, when a decoder has to process 32 or 64 test vectors by using 2 or 4 processing units in parallel, respectively, part of these units would have to process their last test vector while the processing of the next frame has already started (see Figure 3). Processing those last test vectors would imply that the latency of the decoder increases and that a considerable amount of registers are required to concurrently process data from two frames. In the present work we propose a solution for this problem that still profits from the use of a Gray code processing sequence.

6. Conclusions

In this work, we present three soft-decision Reed–Solomon LCC decoders for $\eta = 4$, quasi-$\eta = 5$ and quasi-$\eta = 6$ that are based on HD decoding. The Frame Error Rate coding gains of the proposed decoders are 0.45, 0.52 and 0.60 at FER = 10^{-6} compared to hard-decision decoding, which are higher than those of previously published LCC decoders. The proposed architecture is easily scalable and is based on a simplification of the Multiplicity Assignment stage. We present also detailed implementation schematics for those computational blocks that are different from conventional implementations. In the present work we propose novel solutions for decoders that use a high number of test vectors, problems that go beyond the simple parallelization of resources. We present implementation results in ASIC and FPGA devices for the three decoders. The results show, for example, that our $\eta = 4$ decoder, which has a 0.07 higher coding than the best $\eta = 3$ decoder published to date, requires 41% less area and 29% less LUTs, in ASIC and FPGA, respectively, than the $\eta = 3$ decoder. This results are achieved without spoiling the throughput nor the latency of the decoder.

Author Contributions: Conceptualization, J.V.; All authors have equally contributed to the methodology, implementation of the models, validation, and writing the original draft.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Cideciyan, R.D.; Gustlin, M.; Li, M.P.; Wang, J.; Wang, Z. Next Generation Backplane and Copper Cable Challenges. *IEEE Commun. Mag.* **2013**, *51*, 130–136. [CrossRef]
2. Perrone, G.; Valls, J.; Torres, V.; García-Herrero, F. Reed–Solomon Decoder Based on a Modified ePIBMA for Low-Latency 100 Gbps Communication Systems. *Circuits Syst. Signal Process.* **2018**. [CrossRef]
3. Bellorado, J. Low-Complexity Soft Decoding Algorithms for Reed–Solomon Codes. Ph.D. Thesis, Harvard University, Cambridge, MA, USA, 2006.
4. Chase, D. Class of Algorithms for Decoding Block Codes with Channel Measurement Information. *IEEE Trans. Inf. Theory* **1972**, *18*, 170–182. [CrossRef]
5. Koetter, R.; Vardy, A. Algebraic Soft-Decision Decoding of Reed–Solomon Codes. *IEEE Trans. Inf. Theory* **2003**, *49*, 2809–2825. [CrossRef]
6. Sudan, M. Decoding of Reed–Solomon Codes beyond the Error-Correction Bound. *J. Complex.* **1997**, *13*, 180–193. [CrossRef]
7. Guruswami, V.; Sudan, M. Improved Decoding of Reed–Solomon and Algebraic-Geometry Codes. *IEEE Trans. Inf. Theory* **1999**, *45*, 1757–1767. [CrossRef]
8. Blahut, R.E. *Theory and Practice of Error Control Codes*; Addison-Wesley: Reading, MA, USA, 1983.
9. Jiang, J.; Narayanan, K.R. Algebraic Soft-Decision Decoding of Reed–Solomon Codes Using Bit-Level Soft Information. *IEEE Trans. Inf. Theory* **2008**, *54*, 3907–3928. [CrossRef]
10. Gross, W.J.; Kschischang, F.R.; Koetter, R.; Gulak, R.G. A VLSI Architecture for Interpolation in Soft-Decision List Decoding of Reed–Solomon Codes. In Proceedings of the IEEE Workshop on Signal Processing Systems, San Diego, CA, USA, 16–18 October 2002; pp. 39–44.

11.	Zhu, J.; Zhang, X.; Wang, Z. Backward Interpolation Architecture for Algebraic Soft-Decision Reed–Solomon Decoding. *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.* **2009**, *17*, 1602–1615.

12.	Zhu, J.; Zhang, X. Efficient VLSI Architecture for Soft-Decision Decoding of Reed–Solomon Codes. *IEEE Trans. Circuits Syst. I Regul. Pap.* **2008**, *55*, 3050–3062.

13.	Wang, Z.; Ma, J. High-Speed Interpolation Architecture for Soft-Decision Decoding of Reed–Solomon Codes. *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.* **2006**, *14*, 937–950. [CrossRef]

14.	Zhang, X. Reduced Complexity Interpolation Architecture for Soft-Decision Reed–Solomon Decoding. *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.* **2006**, *14*, 1156–1161. [CrossRef]

15.	Zhang, X.; Parhi, K.K. Fast Factorization Architecture in Soft-Decision Reed–Solomon Decoding. *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.* **2005**, *13*, 413–426. [CrossRef]

16.	Zhang, X.; Zhu, J. Hardware Complexities of Algebraic Soft-Decision Reed–Solomon Decoders and Comparisons. In Proceedings of the 2010 Information Theory and Applications Workshop (ITA), San Diego, CA, USA, 31 January–5 February 2010; pp. 1–10.

17.	Bellorado, J.; Kavcic, A. Low-Complexity Soft-Decoding Algorithms for Reed–Solomon Codes—Part I: An Algebraic Soft-In Hard-Out Chase Decoder. *IEEE Trans. Inf. Theory* **2010**, *56*, 945–959. [CrossRef]

18.	García-Herrero, F.; Valls, J.; Meher, P.K. High-Speed RS(255,239) Decoder Based on LCC Decoding. *Circuits Syst. Signal Process.* **2011**, *30*, 1643–1669. [CrossRef]

19.	Zhang, W.; Wang, H.; Pan, B. Reduced-Complexity LCC Reed–Solomon Decoder Based on Unified Syndrome Computation. *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.* **2013**, *21*, 974–978. [CrossRef]

20.	Peng, X.; Zhang, W.; Ji, W.; Liang, Z.; Liu, Y. Reduced-Complexity Multiplicity Assignment Algorithm and Architecture for Low-Complexity Chase Decoder of Reed–Solomon Codes. *IEEE Commun. Lett.* **2015**, *19*, 1865–1868. [CrossRef]

21.	Lin, Y.M.; Hsu, C.H.; Chang, H.C.; Lee, C.Y. A 2.56 Gb/s Soft RS(255,239) Decoder Chip for Optical Communication Systems. *IEEE Trans. Circuits Syst. I Regul. Pap.* **2014**, *61*, 2110–2118. [CrossRef]

22.	Wu, Y. New Scalable Decoder Architectures for Reed–Solomon Codes. *IEEE Trans. Commun.* **2015**, *63*, 2741–2761. [CrossRef]

23.	Zhu, J.; Zhang, X. Factorization-Free Low-complexity Chase Soft-Decision Decoding of Reed–Solomon Codes. In Proceedings of the 2009 IEEE International Symposium on Circuits and Systems, Taipei, Taiwan, 24–27 May 2009; pp. 2677–2680.

24.	Garcia-Herrero, F.; Canet, M.J.; Valls, J.; Meher, P.K. High-Throughput Interpolator Architecture for Low-Complexity Chase Decoding of RS Codes. *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.* **2012**, *20*, 568–573. [CrossRef]