electronics

MDPI

Article

Comparison of FPGA and Microcontroller Implementations of an Innovative Method for Error Magnitude Evaluation in Reed–Solomon Codes

Valentina Bianchi, Marco Bassoli * and Ilaria De Munari

Department of Engineering and Architecture, University of Parma, Parco Area delle Scienze, 181/A, 43124 Parma, Italy; valentina.bianchi@unipr.it (V.B.); ilaria.demunari@unipr.it (I.D.M.)

* Correspondence: marco.bassoli@unipr.it; Tel.: +39-0521-906043

Received: 23 October 2019; Accepted: 12 December 2019; Published: 1 January 2020

Abstract: Reed–Solomon (RS) codes are one of the most used solutions for error correction logic in data communications. RS decoders are composed of several blocks: among them, many efforts have been made to optimize the error magnitude evaluation module. This paper aims to assess the performance of an innovative algorithm introduced in the literature by Lu et al. under different systems configurations and hardware platforms. Several configurations of the encoded message chosen between those typically used in different applications have been designed to be run on an FPGA (field programmable gate array) device and an MCU (microcontroller unit). The performances have been evaluated in terms of resource usage and output delay for the FPGA and in terms of code execution time for the MCU. As a benchmark in the analysis, the well-established Forney's method is exploited: it has been implemented in the same configurations and on the same hardware platforms for a proper comparison. The results show that the theoretical finding are fully confirmed only in the MCU implementation, while on FPGA, the choice of one method with respect to the other depends on the optimization feature (i.e., time or area) that has been decided as a preference in the specific application.

Keywords: error correction codes; Reed–Solomon; embedded devices; FPGA; microcontroller

1. Introduction

The information exchange represents a significant aspect pervading all modern systems, from miniaturized wireless earphones to heavyweight space satellites. Considerable research work has been done to improve the efficiency of communication processes [1,2] and, among all the introduced techniques, errors management is of utmost importance.

The basic errors management implementation involves the error detection and message resend [3]: if the message is corrupted by random and burst errors, the receiver is configured to require subsequent retransmissions of the same message until the error is no longer present. A clear downside of this approach is the increase of the number of messages in the communication channel, which may bring band saturation and limitations in high-speed applications.

One solution is the introduction of error correction features [4]. With this approach, the receiver is able to detect the error and, under certain conditions, to correct the message. At the price of an increase in the receiver complexity and in the messages' length, the channel traffic is reduced, with a positive effect for the communication speed [5].

Considerable research work has been done on this topic [6,7], and a variety of systems based on different algorithms have been proposed. Among them, Hamming error correction codes (ECCs) were one of the first introduced, in 1950, to correct errors in punched card readers [8]. Hamming

ECCs can correct up to one error in the message and are characterized by low parity information, which makes them suitable for applications requiring high data rates, but low error occurrence. To improve Hamming codes, Bose–Chaudhuri–Hocquenghem (BCH) codes were presented in the works of [9–11]. They solve the limitation of the maximum correctable errors by introducing Galois Fields [12]. By exploiting the multiplication, the addition, and the inversion defined in the field, the system can be configured to correct the desired amount of errors in the message by increasing the parity information appended to the message itself. A subset of BCH codes are Reed–Solomon (RS) codes [13], which ease the implementation in binary encoded messages. RS codes are among the most implemented solutions in a wide set of areas, such as data storage, bar and QR (quick response) codes, space and TV transmission, and so on [14–17].

As all communication systems, RS ECCs are composed of an encoder at the sender side and a decoder at the receiver side. Most of the works in the literature [18–20] focus on the decoder part, as it holds the most complex operations. Several approaches have been proposed to improve the performance of the decoder [21,22]. Among them, many research works focus on the optimization of error magnitude evaluation [23–25]. A traditional implementation exploits the Forney method [23]. Komo and Joiner proposed an iterative algorithm in the work of [24] based on a Vandermonde matrix, which results in an improvement of the processing speed of about 2.8 times compared with the Forney method. Then, an innovative method has been proposed by Lu et al. [25], allowing the same result to be obtained with less system complexity and computational effort. However, to the best of our knowledge, this algorithm has not yet been implemented and only a theoretical description is provided in the literature. The aim of this paper is to complete Lu's theoretical work applying two different implementation strategies to the method and estimating the performance. For this purpose, two different platforms were considered, an FPGA (field programmable gate array) and an MCU (microcontroller unit), to evaluate the behavior over different possible solutions: the first fully parallel and the latter fully sequential. The performance is evaluated in terms of processing time (in both MCU and FPGA implementations) and area (for the FPGA only). To properly compare the results, the Forney method was deployed on the same platforms and its behavior was assessed as well. This method was adopted as a benchmark as it is, to date, the most implemented in RS decoders [26–28], and is considered the most efficient method. The results of this work can then be exploited to identify when Lu's method is preferable with respect to Forney's method, depending on the platform chosen to implement the whole decoder. The purely algorithmic and theoretical comparison already reported in the literature [25] does not take into account the implementation peculiarities of the two platforms considered (i.e., functions that can/cannot be parallelized or simplified) that may affect the final results. The paper is organized as follows. In Section 2, after a summary on error magnitude evaluation in RS codes, Forney's and Lu's methods are briefly reported. Then, the MCU and FPGA system setups used for the implementations and evaluation are illustrated. In Section 3, the results are reported and discussed. Finally, in Section 4, conclusions are drawn.

2. Error Magnitude Evaluation in Reed–Solomon Codes

An RS encoded message is composed of a sequence of n symbols (each one of m bits), of which k symbols represent the original uncoded message and the remaining are the added parity [29]. An example of an encoded message is shown in Figure 1.

In an RS system, the maximum number of detectable and correctable errors in the message is defined as follows:

$$t = \frac{n-k}{2}. \tag{1}$$

The architecture of a traditional RS decoder is shown in Figure 2.

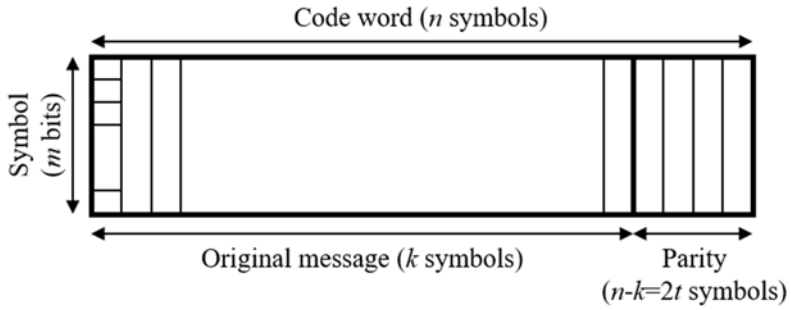


Figure 1. A Reed–Solomon encoded message.

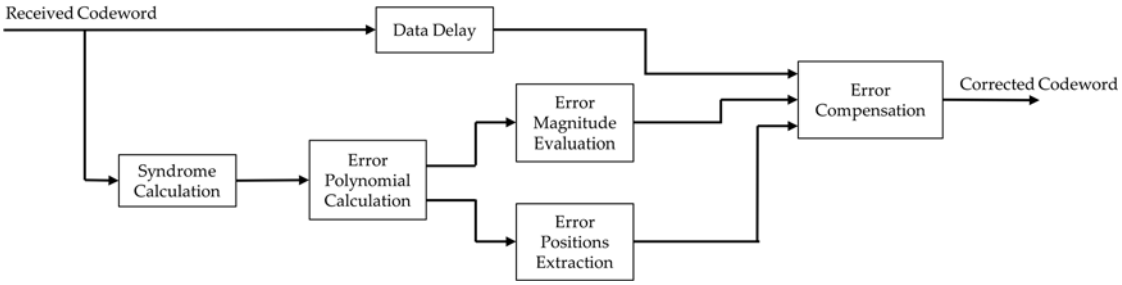


Figure 2. Traditional RS decoder scheme.

The decoder is typically composed of five subsystems [30,31]: syndrome calculation, error polynomial calculation, error positions extraction, error magnitude evaluation, and error compensation. Once a message is received by a Reed–Solomon decoder, the first step should be to divide the received polynomial by the generator polynomial chosen for encoding: the remainders of this division are known as syndromes, and they do not depend on the transmitted code word, but rather only on errors. The syndrome calculation block computes the $2t$ syndromes contained into a Reed–Solomon code word, usually exploiting Horner's method [32]. The next step is to introduce the error locator polynomial that contains the information about the location of the errors and their magnitudes. Two methods are widely used in error polynomial calculation, the Euclidean algorithm [33] and Berlekamp's algorithm [34]. Once the coefficients of error location polynomial are carried out, the error position block identifies the corrupted symbols, by means of the Chien search algorithm [35], while the error magnitude block computes the error values. Finally, the error compensation block uses this information to fix the errors. In traditional RS decoders, as depicted in Figure 2, the whole process typically takes several clock cycles: each block contributes to this delay and the optimization of a single block can improve the performance of the entire system.

In classical RS decoders, the search for error position is carried out in parallel with the error position estimation and, within the blocks, a serial strategy is adopted. This implies a number of clock cycles equal to n for the error magnitude and position evaluation. Recently, many efforts have been made to speed up the whole process, exploiting parallel Chien search strategies [36,37]. Combining these methods with parallel algorithms for error magnitude extraction can potentially reduce the required number of clock cycles to n/p, where p is the Chien search parallelism, plus one clock cycle for the subsequent error magnitude evaluation. In this context, the analysis of the parallelism capability and the analog in–out time of the available error magnitude strategies can be very useful to identify the preferred method to optimize the whole process. For this reason, in the following, we will focus only on the error magnitude subsystem.

There are many approaches to compute error magnitudes in the literature [23–25] and, among these, the one with the most promising performance is the one introduced by Lu et al. [25]. The error magnitude subsystem can be schematically represented as in Figure 3.

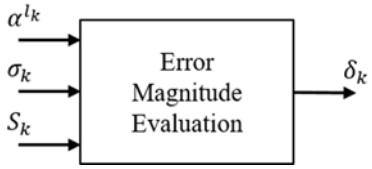


Figure 3. Error magnitude evaluation block.

To compute the error magnitudes δ_k, the subsystem takes as inputs the positions of the errors α^{l_k} in the code word; the error-location polynomial coefficients σ_k, computed in the previous error polynomial calculation phase; and the syndrome polynomial coefficients S_k, computed in the syndrome calculation phase.

The RS error magnitude evaluation performed with a standard technique (the Forney method) will be compared with the alternative method introduced by Lu et al. Table 1 shows the theoretical number of required Galois additions ($N_{\bigoplus}$), multiplications ($N_{\bigotimes}$), and inversions (N^{-1}) when the message includes a number ν of errors.

Table 1. Number of Galois field operations required by Forney's and Lu's methods.

Galois Operation	Forney's Method	Lu's Method
Addition	$N_{\oplus} = \nu\left(\frac{\nu-1}{2} + \frac{3(\nu-1)}{2}\right)$	$N_{\bigoplus} = \frac{3\nu(\nu-1)}{2}$
Multiplication	$N_{\bigotimes} = \nu\left(\frac{\nu+1}{2} + \frac{3\nu-1}{2}\right)$	$N_{\bigotimes} = \frac{\nu(3\nu-1)}{2}$
Inversion	$N^{-1} = \nu$	$N^{-1} = \nu - 1$

In Table 2, a numerical example of the equations of Table 1 with $\nu = 3$ is reported.

Table 2. Numerical example of equations of Table 1.

Number of Errors	Forney's Method	Lu's Method
$\nu = 3$	$N_{\bigoplus} = 12$ $N_{\bigotimes} = 18$ $N^{-1} = 3$	$N_{\bigoplus} = 9$ $N_{\bigotimes} = 12$ $N^{-1} = 2$

In the following, for the sake of clarity, the two methods are briefly summarized and, as an example, implementation in terms of addition, multiplication, and inversion blocks in the case of an error magnitude $\nu = 3$ is presented.

2.1. Forney Method

The Forney method relies on the Forney equation (2):

$$\delta_k = \frac{Z_0\left(\alpha^{-l_k}\right)}{\sigma\prime(\alpha^{-l_k})}\text{, for } k = 1, 2, \ldots, \nu\text{,} \tag{2}$$

where ν is the number of error actually present in the code word ($\nu \leq t$), and

$$\alpha^{-l_k} = 1/\alpha^{l_k}\text{,} \tag{3}$$

$$\sigma'\left(\alpha^{-l_k}\right) = \sigma_1 + \sigma_3\alpha^{-2l_k} + \ldots + \sigma_{odd}\alpha^{-(odd-1)l_k}\text{,} \tag{4}$$

$$Z_0\left(\alpha^{-l_k}\right) = S_1 + (S_2 + \sigma_1 S_1)\alpha^{-l_k} + \ldots + (S_\nu + \sigma_1 S_{\nu-1} + \ldots + \sigma_{\nu-1}S_1)\alpha^{(\nu-1)l_k}\text{,} \tag{5}$$

with *odd* as the largest odd number less than or equal to ν.

In Figure 4, an example of an implementation of Equation (3) is presented. For simplicity, the $Z_0\left(\alpha^{-l_k}\right)$ and $\sigma\prime\left(\alpha^{-l_k}\right)$ notations are replaced by Z_{0k} and σ'_k, respectively.

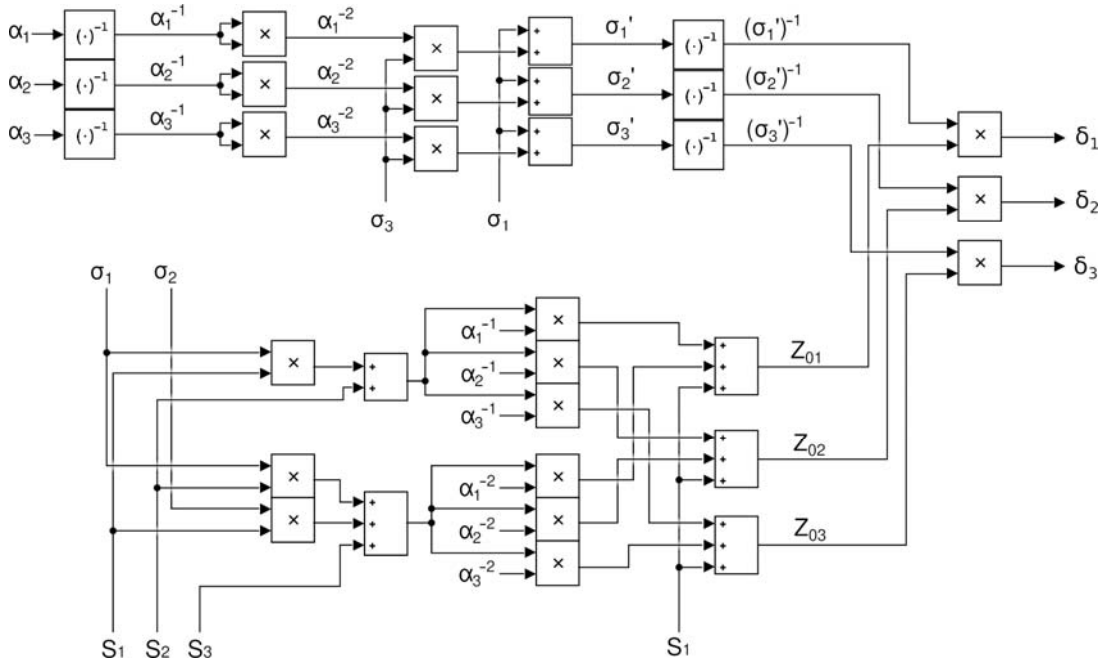


Figure 4. Error magnitude evaluation with $\nu = 3$ implemented with the Forney algorithm. The blocks shown are additions, multiplications, and inversions ($(\cdot)^{-1}$) over the Galois field.

In this case, the system is configured to perform the error magnitude evaluation of up to three errors ($\nu = 3$). Inside the system, the two operands of Equation (3) (Z_{0k} and σ'_k) can be identified and the number of required operations to perform the result is found to follow the equation of Table 1: 3 inversions (plus 3 required to generate the inverses of α coefficients), 12 additions, and 18 multiplications.

2.2. Lu Method

The method presented by Lu et al. [25] performs error magnitude computation with less computational effort in respect to the Forney method, as shown in Table 1. It is composed by three main phases, each one performing different operations: preprocessing phase, syndrome refining phase, and error-magnitude extraction phase. In the preprocessing phase, partial results, $P_{i,j}$ and $Q_{i,j}$, are introduced for convenience, and are defined as follows:

$$P_{i,0} = Q_{i,0} = \alpha^{l_i}\ , \text{ for } i = 1,\ 2,\ \ldots, \nu\ , \tag{6}$$

$$P_{i,j} = \alpha^{l_i} + \alpha^{l_j}\ , \text{ for } 1 \le j < i \le \nu\ , \tag{7}$$

$$Q_{i,j} = Q_{i,j-1}P_{i,j}\ , \text{ for } 1 \le j < i \le \nu\ . \tag{8}$$

In the subsequent syndrome refining phase, following the idea that, to evaluate ν error magnitudes, knowing all error location numbers, only the first ν syndromes are needed, the original syndromes S_k are re-elaborated to compute $S_w^{(k)}$ as

$$S_w^{(1)} = S_w\ , \text{ for } w = 1,\ 2,\ \ldots, \nu\ , \tag{9}$$

$$S_w^{(k)} = S_{w+1}^{(k-1)} + S_w^{(k-1)}P_{k-1,0}\ , \text{ for } k = 3,\ 4,\ \ldots, \nu \text{ and } w = 3,\ 4,\ \ldots, \nu - k + 1\ . \tag{10}$$

Finally, in the error magnitude extraction phase, δ_k can be recursively computed as

$$\delta_\nu = \frac{S_1^{(\nu)}}{Q_{\nu,\nu-1}}\,, \tag{11}$$

$$\delta_k = \frac{S_1^{(k)} + \sum_{i=k+1}^{\nu} \delta_i Q_{i,k-1}}{Q_{k,k-1}}\,, \text{ for } = \nu-1,\ \nu-2,\ \ldots, 1\,. \tag{12}$$

An implementation for $\nu = 3$ is reported in Figure 5.

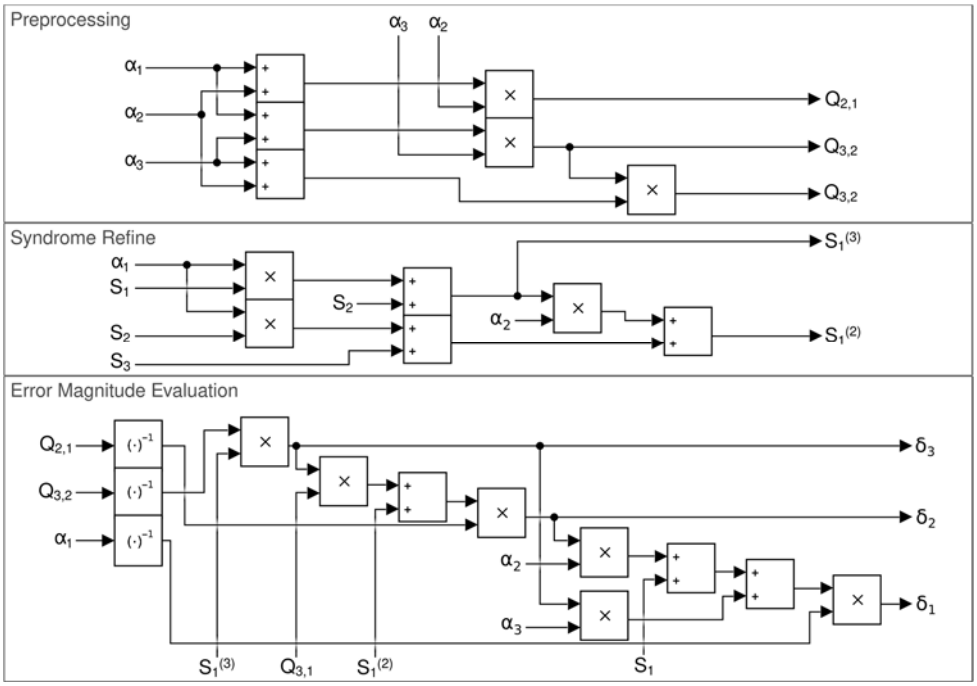


Figure 5. Error magnitude evaluation with $\nu = 3$ implemented with the Lu algorithm. The blocks shown are additions, multiplications, and inversions $((\cdot)^{-1})$ over the Galois field.

As for the Forney example in Figure 4, the number of used operations is found to follow the equations of Table 1: 2 inversions (plus 1 required to generate the inverse of the α_1 coefficient), 9 additions, and 12 multiplications. Compared with the Forney method, the new method requires less operations to obtain the result. Moreover, as can be seen in the diagram, the new method does not need the values of the error locator polynomial (σ_k) as input.

3. Hardware Configuration

The aim of this work is to give an evaluation of Lu's method for estimating error magnitudes in a received code word in Reed–Solomon code, considering practical implementations. As a benchmark for the evaluation, Forney's method was considered. The two algorithms were designed with different configurations of the n, k, t, and m values, as shown in Table 3. Two different m values were chosen: code-words with symbols composed by 4 bits are typically used in image transmission [38,39], while code-words with 8-bit symbols are practically implemented in Quick Response (QR) codes [14], Digital Video Broadcasting—Terrestrial (DVB-T) [40], and the Consultative Committee for Space Data Systems (CCSDS) standard for space applications [41].

Table 3. Configurations designed for the implementation comparison.

Configuration Name	m	(n,k) Notation	Number of Correctable Errors (t)
A	4	(15,11)	2
B	4	(15,9)	3
C	4	(15,7)	4
D	8	(255,249)	3
E	8	(255,239)	8
F	8	(255,223)	16

All configurations were designed to correct any number of errors $v \leq t$ in the code-word. However, as the worst performance is obtained when the maximum number of errors occurs, the number of errors introduced in the code-word was set to be equal to the maximum detectable error (i.e., $v = t$). The configurations were designed for implementations both on an FPGA device and on an MCU, as different behaviors are to be expected on the two hardware devices because of the different nature of the two platforms.

The FPGA platform was used to evaluate the parallelization capability of the two algorithms. For this purpose, the configurations presented in Table 3 for both the Forney and Lu algorithms were designed in full concurrent VHDL code, and results were evaluated in terms of resources' usage and delay to obtain a valid output. The platform environment is Xilinx Vivado 2019.1 and the target device was set to Artix-7 XC7A100T-CSG324 FPGA. As an example, in Figures 6 and 7, the register transfer level (RTL) results of the RS (15,11) configuration are reported.

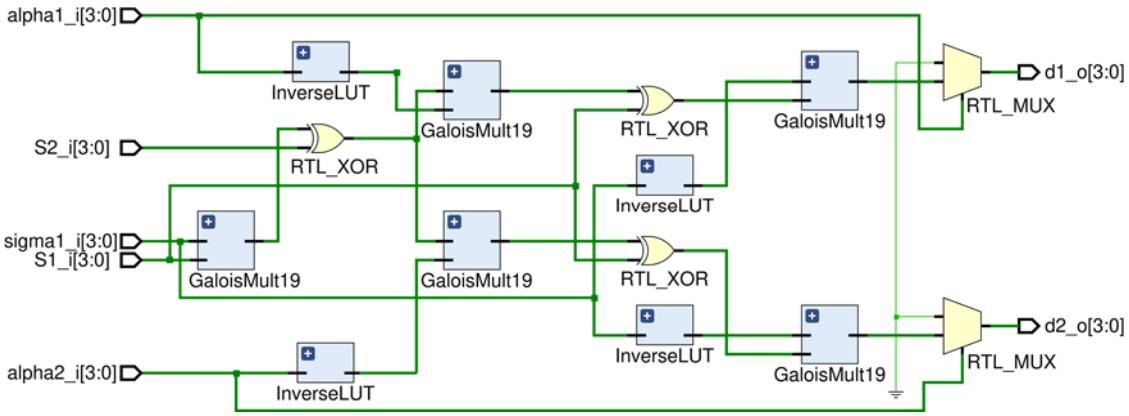


Figure 6. Register transfer level (RTL) result of the Forney method for the RS(15,11) configuration.

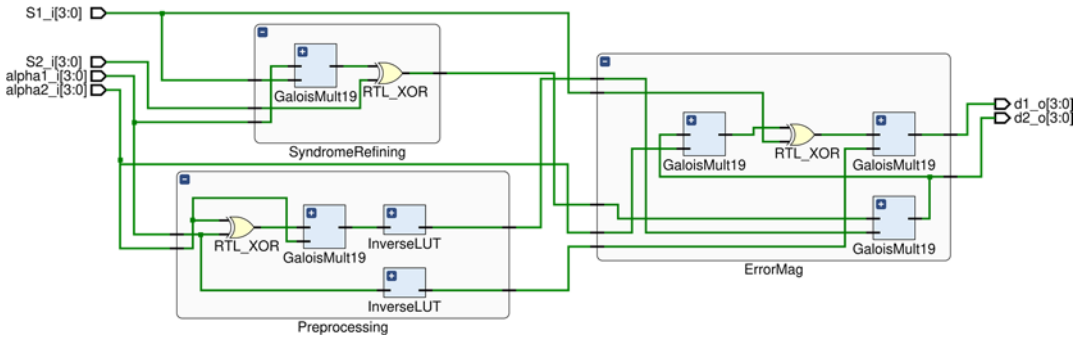


Figure 7. RTL result of the Lu method for the RS(15,11) configuration.

To evaluate the performance of the Forney and Lu methods when implemented on an FPGA platform, a Nexys DDR 4 board hosting the target FPGA was exploited. The measurement set-up is shown in Figure 8.

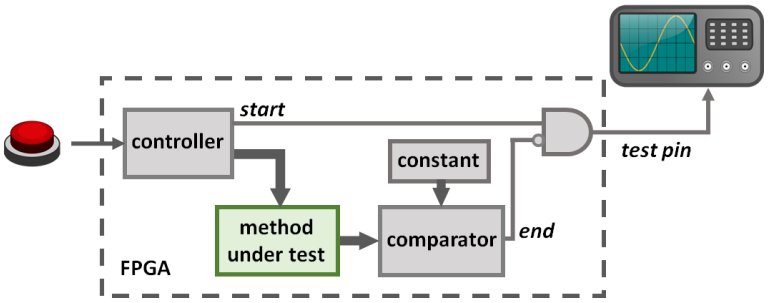
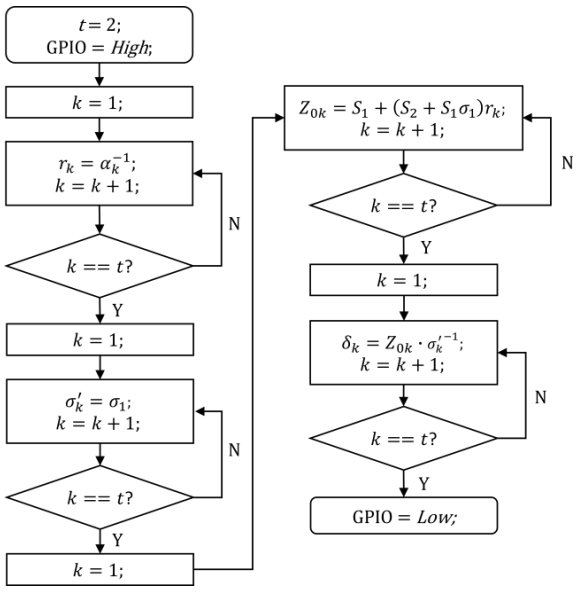


Figure 8. Measurement set-up for the field programmable gate array (FPGA) platform.

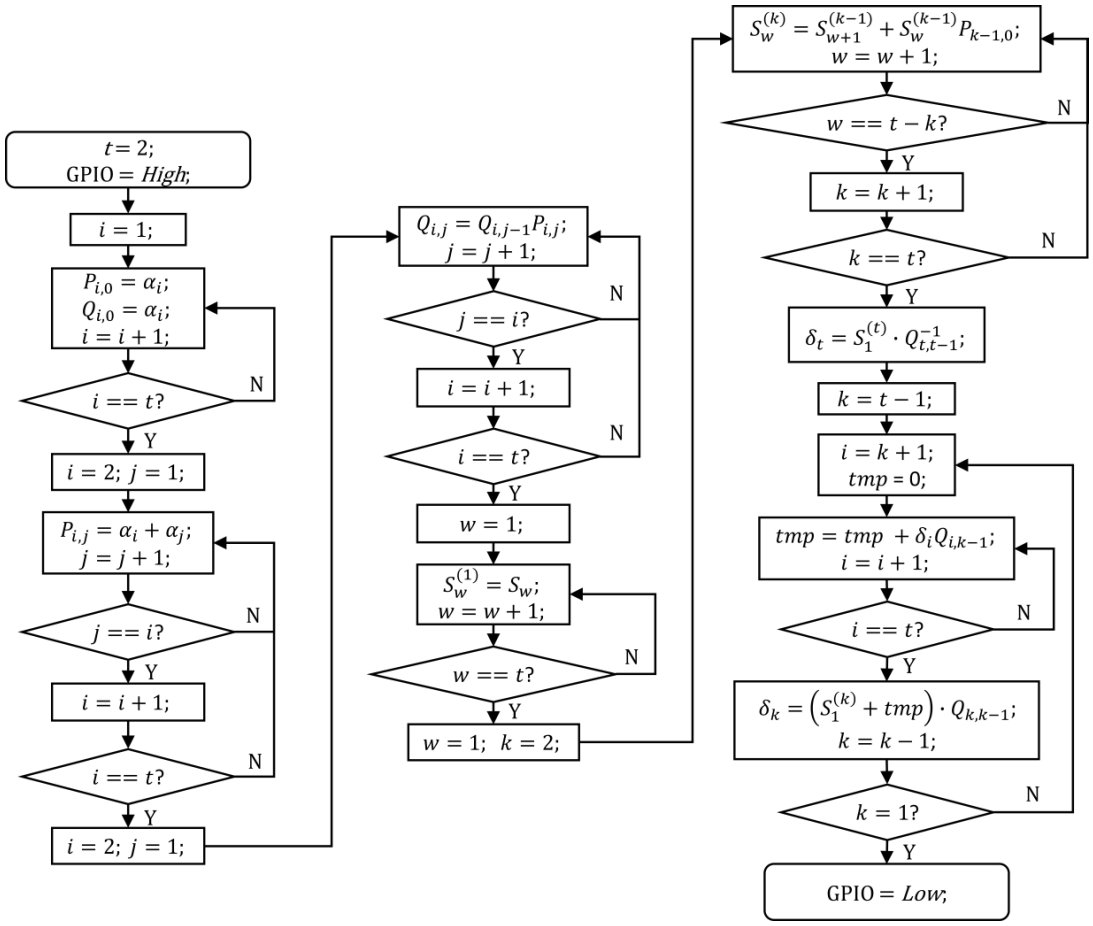
A button is pressed to begin the elaboration and a controller takes care of signaling the start by raising the *start* signal. As the outputs of the method under the test consist of several bits (up to 128 in the worst case), it would be impossible to monitor all of them with an oscilloscope. Thus, a comparator was introduced to assert the *end* signal when a valid data matches a constant that represents the expected value. Thus, the *test pin* rises with the *start* signal and falls when the *end* signal goes high. The time between the two transitions of the *test pin* gives the algorithm output delay. The measurements were carried out with a Tektronix DPO7254 oscilloscope, exploiting its 40 GSa/s, real-time sample rate.

Exploiting the full sequential architecture of an MCU platform, the speed of the two algorithms without parallelization was evaluated. The platform environment in this case is IAR Embedded Workbench, in which the different configurations of Table 3 were coded in C language and flashed on a Texas Instruments (TI) LaunchPad XL development board. The board hosts a TI CC3200 system on chip (SoC) device, with a 32-bit architecture ARM Cortex-M4 MCU clocked at 80 MHz. This device, thanks to an embedded Wi-Fi radio module, is widely used to implement devices and systems compatible with the Internet of Things and wireless sensor network paradigms [42–47], topics in which error correction techniques were demonstrated to gain an advantage in recent implementations [48,49]. Data about execution speeds were collected by measuring a general-purpose input/output (GPIO) pin voltage, which was toggled inside the C code at the computation's start and end. The time between the low-to-high and high-to-low transitions gives the code execution time. The measurements were carried out by means of a Tektronix MSO 2024 oscilloscope. In Figure 9, the Forney and Lu versions of the C code flow chart for the RS (15,11) configuration are reported.



(**a**)

Figure 9. *Cont.*

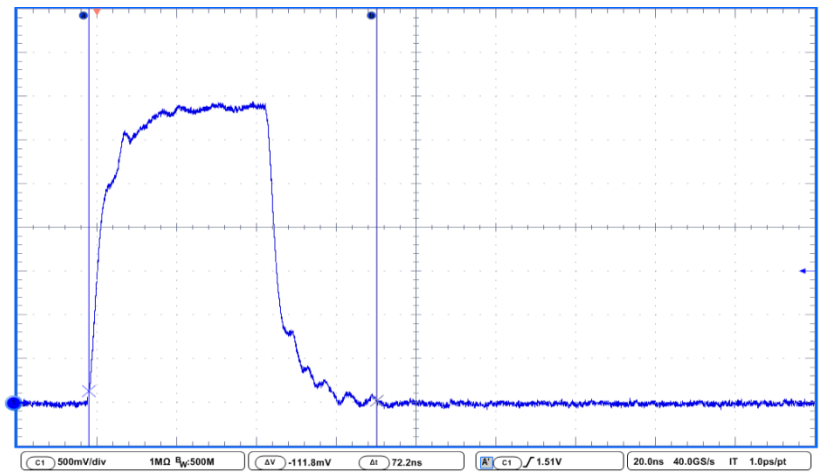


(**b**)

Figure 9. Flowcharts used for the C language implementation of the Forney (**a**) and Lu (**b**) methods. GPIO, general-purpose input/output.

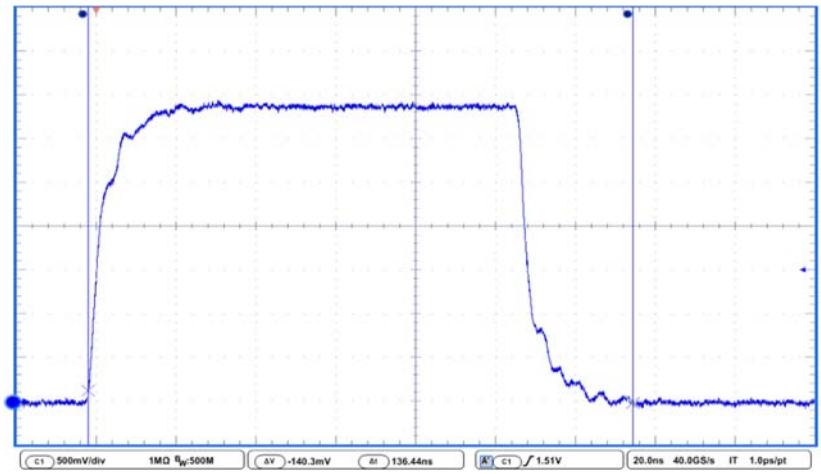
4. Results and Discussion

In Figure 10, an example of measurement carried out with the 40GSa/s Tektronix DPO7254 oscilloscope in the case of the FPGA platform is reported.



(**a**)

Figure 10. *Cont.*



(**b**)

Figure 10. Oscilloscope measurements of the GPIO pin for the configuration F of Table 3 in the case of FPGA implementation. (**a**) Measurement of the execution time of the Forney algorithm. (**b**) Measurement of the execution time of the Lu method.

In Table 4, the performance of the Forney and Lu methods when implemented on an FPGA platform are compared. Data about cell LUTs' (look-up tables') usage refer only to the block performing Forney or Lu algorithm (excluding logic used to compute the output delay, shown in Figure 8).

Table 4. Field programmable gate array (FPGA) implementations results. LUTs, look-up tables.

System Configuration	Forney		Lu	
	Cell LUTs	Time (ns)	Cell LUTs	Time (ns)
A	56	6.7	62	8.3
B	163	10.2	138	15.2
C	263	11.7	268	22.5
D	828	20.8	760	32.8
E	7126	47	6081	71
F	27,100	72.2	21,082	136.4

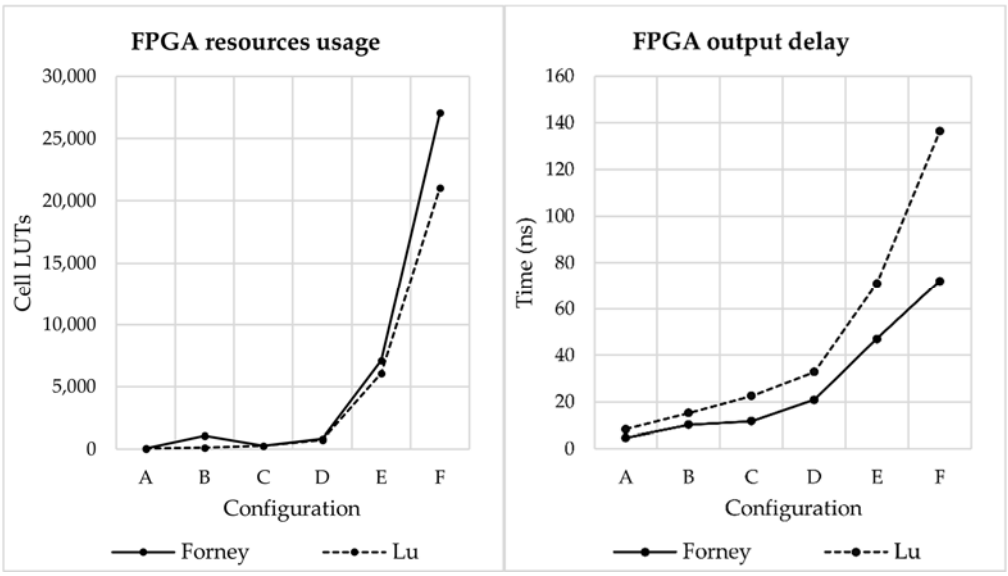
To better appreciate the results, the same are also represented in Figure 11.

As can be seen, as theorized by the model presented in the literature, Lu's method performs better in terms of resources usage in respect of Forney's one. The difference between the two algorithms increases with the system complexity (i.e., increasing the m and t parameters), and it is clearly appreciable for the RS(255,223) configuration. Nevertheless, from the output delay point of view, Forney's algorithm definitely performs better: this behavior is justified by the fact that Lu's algorithm computes the result in a strong, recursive way (as can be seen, for example, in the error magnitude extraction phase). This leads to deep architectures in which the last output is strongly dependent on the previous one, negatively affecting the overall latency. From this point of view, Forney's algorithm is more efficient thanks to its parallel-prone structure, as can be seen in the RTL resulting scheme (Figure 6).

Therefore, if the target application is to be implemented on an FPGA platform, Forney's should be preferred, unless the available resources in terms of area are not binding.

In Table 5, the performance of the two algorithms when the implementation platform is an MCU is reported. The measurement technique can be better visualized in Figure 12, where the outputs from the Tektronix MSO 2024 oscilloscope in the case of configuration RS (255,223) are reported. The output delays obtained in the case of MCU implementation are not intended for a direct comparison with ones measured in the case of FPGA implementation. The FPGA solution outperforms the MCU one: the

comparison must be made between the two methods, considering the same implementation platform and the same system configuration.

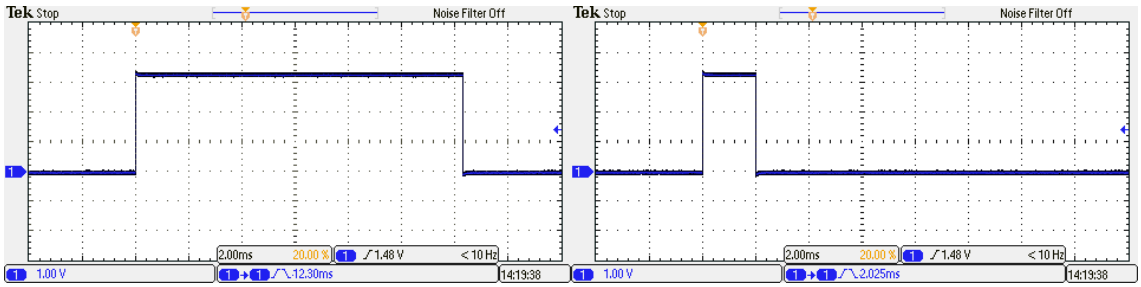


(**a**) (**b**)

Figure 11. Graphical representation of the results of Table 4. (**a**) FPGA implementations resources' usage in terms of cell look-up tables (LUTs), (**b**) FPGA implementations' output delay.

Table 5. Microcontroller unit (MCU) implementations results.

System Configuration	Forney	Lu
	Time (µs)	Time (µs)
A	186.1	169.8
B	668.1	389.3
C	1400	699.2
D	1100	607.3
E	15,900	4500
F	110,600	18,300



(**a**) (**b**)

Figure 12. Oscilloscope measurements of the GPIO pin for the configuration F of Table 3 in the case of MCU implementation. (**a**) Measurement of the execution time of Forney's algorithm, (**b**) Measurement of the execution time of Lu's method.

As can be seen in MCU implementation, the improvements made by the method proposed by Lu et al. are fully evident. In this case, as opposed to the FPGA implementation, all the operations are computed sequentially, and Lu's method can take full advantage of the minor number of operations required to produce the result. Also, in this case, the advantage grows with the system complexity: in

the RS(255,223) configuration, the time needed to get a valid result with Lu's method is three times lower than that required with Forney's algorithm. To better appreciate these results, the execution times are also compared in Figure 13.

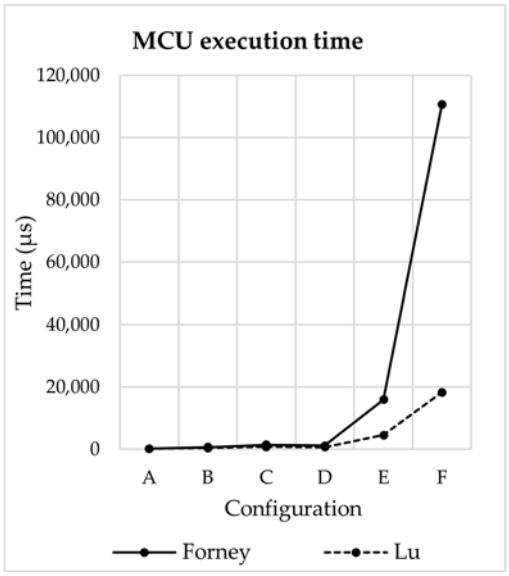


Figure 13. Graphical representation of the results of Table 5 regarding the MCU implementations' code execution time.

Considering the overall results, it appears that the conclusions drawn by Lu et al. in the paper introducing their method are fully valid only in the case of MCU implementation. In case of FPGA (or in general parallel architectures), the choice between Lu and Forney's algorithms depends on what optimization strategy has to be followed. If the output delay is a strong constraint, Forney's method should be preferred, owing to the minor delay required. However, if a circuit with less occupied area is needed, Lu's method is the better choice.

5. Conclusions

In this paper, an innovative method, originally introduced by Lu et al. [25] to compute error magnitude values in an RS decoder, was implemented to evaluate its realistic performance. Two platforms, an ARM based MCU and an Artix-7 FPGA, were considered to assess different implementation strategies, fully sequential and fully combinational, respectively. As a benchmark for the analysis, we selected Forney's algorithm because it is widely used and, to date, it is considered the standard method for error magnitude evaluation in Reed–Solomon code. For both implementations, the two algorithms were designed considering six practical configurations, normally used in specific applications. All the solutions were evaluated in terms of output delay, that is, time to get a valid result when the system is fed with valid inputs. The FPGA based solutions were also evaluated in terms of resources' (LUTs') usage. The results show that, when implemented with a parallel strategy, Lu's method performs better in respect to Forney's one in terms of occupied LUTs, while suffering from the strongly recursive approach in terms of output time. However, Lu's method performs definitively better in case of MCU, when it can take advantage of the minor number of operations required to produce the output, with an output delay up to three times lower in the configurations examined in this paper. From these results, we can conclude that, if an FPGA platform has been considered for the implementation of the whole decoder, the designer should select Forney's method if there are stringent time constraints, while Lu's method is eligible when the area occupation is critical. Instead, when the decoder has to be implemented on an MCU platform, Lu's method should be preferred owing to the reduced execution time.

Author Contributions: Conceptualization, V.B. and I.D.M.; Methodology, V.B., M.B., and I.D.M.; Validation, V.B., M.B., and I.D.M.; Formal Analysis, V.B. and M.B.; Investigation, V.B., M.B., and I.D.M.; Data Curation, V.B. and M.B.; Writing—Original Draft Preparation, V.B., M.B., and I.D.M.; Writing—Review & Editing, V.B., M.B., and I.D.M.; Visualization, V.B., M.B., and I.D.M.; Supervision, I.D.M.; Project Administration, I.D.M.; Funding Acquisition, I.D.M. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

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

References

1. Wang, J.; Yang, J.Y.; Fazal, I.M.; Ahmed, N.; Yan, Y.; Huang, H.; Ren, Y.; Yue, Y.; Dolinar, S.; Tur, M.; et al. Terabit free-space data transmission employing orbital angular momentum multiplexing. *Nat. Photonics* **2012**, *6*, 488–496. [CrossRef]
2. Saltzberg, B.R. Performance of an Efficient Parallel Data Transmission System. *IEEE Trans. Commun. Technol.* **1967**, *15*, 805–811. [CrossRef]
3. Peterson, W.W.; Brown, D.T. Cyclic Codes for Error Detection. *Proc. IRE* **1961**, *49*, 228–235. [CrossRef]
4. Zhang, Z. Linear network error correction codes in packet networks. *IEEE Trans. Inf. Theory* **2008**, *54*, 209–218. [CrossRef]
5. McAuley, A.J. Reliable Broadband Communication Using a Burst Erasure Correcting Code. In Proceedings of the ACM Symposium on Communications Architectures and Protocols, SIGCOMM, Philadelphia, PA, USA, 26–28 September 1990; Association for Computing Machinery Inc.: New York, NY, USA, 1990; pp. 297–306.
6. Sudan, M. Decoding of Reed Solomon Codes beyond the Error-Correction Bound. *J. Complexity* **1997**, *13*, 180–193. [CrossRef]
7. Sathananathan, K.; Tellambura, C. Forward Error Correction Codes to Reduce Intercarrier Interference in OFDM. In Proceedings of the ISCAS 2001–2001 IEEE International Symposium on Circuits and Systems, Sydney, Australia, 6–9 May 2001; IEEE: Piscataway, NJ, USA, 2002; Volume 4, pp. 566–569.
8. Hamming, R.W. Error Detecting and Error Correcting Codes. *Bell Syst. Tech. J.* **1950**, *29*, 147–160. [CrossRef]
9. Bose, R.C.; Ray-Chaudhuri, D.K. On a class of error correcting binary group codes. *Infect. Control* **1960**, *3*, 68–79. [CrossRef]
10. Hocquenghem, A. Codes correcteurs d'erreurs. *Chiffres* **1959**, *2*, 147–156.
11. Poolakkaparambil, M.; Mathew, J.; Jabir, A. Multiple Bit Error Tolerant Galois Field Architectures Over GF(2m). *Electronics* **2012**, *1*, 3–22. [CrossRef]
12. Carlitz, L. The Arithmetic of Polynomials in a Galois Field. *Am. J. Math.* **1932**, *54*, 39. [CrossRef]
13. Reed, I.S.; Solomon, G. Polynomial Codes Over Certain Finite Fields. *J. Soc. Ind. Appl. Math.* **1960**, *8*, 300–304. [CrossRef]
14. Kieseberg, P.; Leithner, M.; Mulazzani, M.; Munroe, L.; Schrittwieser, S.; Sinha, M.; Weippl, E. QR Code Security. In Proceedings of the MoMM 2010—8th International Conference on Advances in Mobile Computing and Multimedia, Paris, France, 8–10 November 2010; pp. 430–435.
15. Sun, X.; Skillman, D.R.; Hoffman, E.D.; Mao, D.; McGarry, J.F.; McIntire, L.; Zellar, R.S.; Davidson, F.M.; Fong, W.H.; Krainak, M.A.; et al. Free space laser communication experiments from Earth to the Lunar Reconnaissance Orbiter in lunar orbit. *Opt. Express* **2013**, *21*, 1865. [CrossRef] [PubMed]
16. Tan, G.; Herfet, T. Application Layer Hybrid Error Correction with Reed-Solomon Code for DVB Services Over Wireless LANs. In Proceedings of the 2007 International Conference on Wireless Communications, Networking and Mobile Computing, WiCOM 2007, Shanghai, China, 21–25 September 2007; pp. 2952–2955.
17. Bocharova, I.; Kudryashov, B.; Lyamin, N.; Frick, E.; Rabi, M.; Vinel, A. Low Delay Inter-Packet Coding in Vehicular Networks. *Future Internet* **2019**, *11*, 212. [CrossRef]
18. Shao, H.; Truong, T.; Deutsch, L.; Yuen, J.; Reed, I. A VLSI Design of a Pipeline Reed-Solomon Decoder. *IEEE Trans. Comput.* **1985**, *34*, 393–403. [CrossRef]
19. Moon, H.L.; Seung, B.C.; Jin, S.C. A High Speed Reed-Solomon Decoder. In Proceedings of the IEEE Workshop on VLSI Signal Processing, Sakai, Japan, 16–18 September 1995; IEEE: Piscataway, NJ, USA, 1995; pp. 362–367.
20. Torres, V.; Valls, J.; Canet, M.J.; García-Herrero, F. Soft-decision low-complexity chase decoders for the RS (255,239) code. *Electronics* **2019**, *8*, 10. [CrossRef]

21. Lee, H. An Area-Efficient Euclidean Algorithm Block for Reed-Solomon Decoder. In Proceedings of the IEEE Computer Society Annual Symposium on VLSI, Tampa, FL, USA, 20–21 February 2003; IEEE: Piscataway, NJ, USA, 2003.
22. Sarwate, D.V.; Shanbhag, N.R. High-speed architectures for Reed-Solomon decoders. *IEEE Trans. Very Large Scale Integr. Syst.* **2001**, *9*, 641–655. [CrossRef]
23. Forney, G.D. On Decoding BCH Codes. *IEEE Trans. Inf. Theory* **1965**, *11*, 549–557. [CrossRef]
24. Komo, J.J.; Joiner, L.L. Fast Error Magnitude Evaluations for Reed-Solomon Codes. In Proceedings of the IEEE International Symposium on Information Theory, Whistler, BC, Canada, 17–22 September 1995; IEEE: Piscataway, NJ, USA, 1995.
25. Lu, E.H.; Chen, T.C.; Lu, P.Y. A new method for evaluating error magnitudes of Reed-Solomon codes. *IEEE Commun. Lett.* **2014**, *18*, 340–343. [CrossRef]
26. Mhaske, S.D.; Ghodeswar, U.; Sarate, G.G. Design of Area Efficient Reed Solomon Decoder. In Proceedings of the 2014 2nd International Conference on Devices, Circuits and Systems (ICDCS), Combiatore, India, 6–8 March 2014; IEEE: Piscataway, NJ, USA, 2014. [CrossRef]
27. Li, X.; Zhang, W.; Liu, Y. Efficient architecture for algebraic soft-decision decoding of Reed-Solomon codes. *IET Commun.* **2015**, *9*, 10–16. [CrossRef]
28. Lee, H. A High-Speed Low-Complexity Reed—Solomon Decoder for Optical Communications. *IEEE Trans. Circuits Syst. II Express Briefs* **2005**, *52*, 461–465. [CrossRef]
29. Clark, G.C.; Cain, J.B. *Error-Correction Coding for Digital Communications*; Springer: New York, NY, USA, 1981.
30. Blahut, R.E. *Theory and Practice of Error Control Codes*; Addison-Wesley Pub. Co.: Boston, MA, USA, 1983; ISBN 9780201101027.
31. Lin, S.; Costello, D.J. *Error Control Coding: Fundamentals and Applications (Prentice-Hall Computer Applications in Electrical Engineerin)*; Prentice Hall: Upper Saddle River, NJ, USA, 1983; ISBN 013283796X.
32. Rabaey, J.M.; Potkonjak, M.; Wakabayashi, K. Efficient Throughput Optimization of Feedback Linear Computations Using Generalized Horner's Scheme. In Proceedings of the 1995 International Conference on Acoustics, Speech, and Signal Processing, Detroit, MI, USA, 9–12 May 1995; IEEE: Piscataway, NJ, USA, 2002. [CrossRef]
33. Sugiyama, Y.; Kasahara, M.; Hirasawa, S.; Namekawa, T. A method for solving key equation for decoding goppa codes. *Infect. Control* **1975**, *27*, 87–99. [CrossRef]
34. Berlekamp, E.R. *Algebraic Coding Theory*; McGraw-Hill: New York, NY, USA, 1968.
35. Chien, R.T.; Watson, T.J. Cyclic Decoding Procedures for Bose-Chaudhuri-Hocquenghem Codes. *IEEE Trans. Inf. Theory* **1964**, *10*, 357–363. [CrossRef]
36. Hu, Q.; Wang, Z.; Zhang, J.; Xiao, J. Low Complexity Parallel Chien Search Architecture for RS Decoder. In Proceedings of the 2005 IEEE International Symposium on Circuits and Systems, Kobe, Japan, 23–26 May 2005; IEEE: Piscataway, NJ, USA, 2005. [CrossRef]
37. Lin, Y.; Yang, C.; Hsu, C.; Chang, H.; Lee, C. A MPCN-Based Parallel Architecture in BCH Decoders for NAND Flash Memory Devices. *IEEE Trans. Circuits Syst. II Express Briefs* **2011**, *58*, 682–686. [CrossRef]
38. Nergui, M.; Sripati Acharya, U.; Rajendra Acharya, U.; Yu, W.; Dua, S. Reliable Transmission of Retinal Fundus Image with Patient Information using Encryption, Watermarking, and Error Control Codes. In *Computational Analysisi of the Human Eye with Applications*; World Scientific Publishing: Singapore, 2011; pp. 319–348.
39. Ejaz, M.Z.; Khurshid, K.; Abbas, Z.; Aizaz, M.A.; Nawaz, A. A Novel Image Encoding and Communication Technique of B/W Images for IOT, Robotics and Drones Using (15,11) Reed Solomon Scheme. In Proceedings of the 2018 Advances in Science and Engineering Technology International Conferences (ASET), Abu Dhabi, UAE, 6 February–5 April 2018; IEEE: Piscataway, NJ, USA, 2018.
40. European Telecommunications Standards Institute. *Digital Video Broadcasting (DVB); Framing Structure, Channel Coding and Modulation for Digital Terrestrial Television*; ETSI-EN-300-744; European Telecommunications Standards Institute: Sophia Antipolis, France, June 2009.
41. CCSDS. *TM Synchronization and Channel Coding—Summary of Concept and Rationale*; CCSDS: Washington, DC, USA, November 2012.
42. Bassoli, M.; Bianchi, V.; De Munari, I. A plug and play IoT Wi-Fi smart home system for human monitoring. *Electronics* **2018**, *7*, 200. [CrossRef]

43. Zantalis, F.; Koulouras, G.; Karabetsos, S.; Kandris, D. A Review of Machine Learning and IoT in Smart Transportation. *Future Internet* **2019**, *11*, 94. [CrossRef]
44. Tang, X.; Wang, X.; Cattley, R.; Gu, F.; Ball, A.D. Energy Harvesting Technologies for Achieving Self-Powered Wireless Sensor Networks in Machine Condition Monitoring: A Review. *Sensors* **2018**, *18*, 4113. [CrossRef]
45. Bianchi, V.; Bassoli, M.; Lombardo, G.; Fornacciari, P.; Mordonini, M.; De Munari, I. IoT Wearable Sensor and Deep Learning: An Integrated Approach for Personalized Human Activity Recognition in a Smart Home Environment. *IEEE Internet Things J.* **2019**, *6*, 8553–8562. [CrossRef]
46. Giannetto, M.; Bianchi, V.; Gentili, S.; Fortunati, S.; De Munari, I.; Careri, M. An integrated IoT-Wi-Fi board for remote data acquisition and sharing from innovative immunosensors. Case of study: Diagnosis of celiac disease. *Sens. Actuators B Chem.* **2018**, *273*, 1395–1403. [CrossRef]
47. Bianchi, V.; Boni, A.; Fortunati, S.; Giannetto, M.; Careri, M.; De Munari, I. A Wi-Fi cloud-based portable potentiostat for electrochemical biosensors. *IEEE Trans. Instrum. Meas.* **2019**. [CrossRef]
48. Brokalakis, A.; Chondroulis, I.; Papaefstathiou, I. Extending the Forward Error Correction Paradigm for Multi-Hop Wireless Sensor Networks. In Proceedings of the 2018 9th IFIP International Conference on New Technologies, Mobility and Security, NTMS 2018, Paris, France, 26–28 February 2018; IEEE: Piscataway, NJ, USA, 2018; Volume 2018, pp. 1–5.
49. Bettayeb, M.; Ghunaim, S.; Mohamed, N.; Nasir, Q. Error Correction Codes in Wireless Sensor Networks: A Systematic Literature Review. In Proceedings of the 2019 3rd International Conference on Communications, Signal Processing, and their Applications, ICCSPA 2019, Sharjah, UAE, 19–21 March 2019; IEEE: Piscataway, NJ, USA, 2019.