FPGA Implementation of a Reed-Solomon CODEC for OTN G.709 Standard with Reduced Decoder Area

Tiago C. BARBOSA, Robson L. MORENO, Tales C. PIMENTA and Luis H. C. FERREIRA
Universidade Federal de Itajubá – UNIFEI
Av. BPS, 1303
Itajubá MG 37500-903 Brazil

ABSTRACT

The Reed-Solomon error correction code is widely used in digital telecommunication systems, including satellite communications and data recording systems such as CD and DVD. This article presents an implementation for the encoder and the decoder of optical communication systems, according to the ITU-T G.709 standard. It presents an approach that multiplexes the traditional decoder blocks. The implementation promotes an expressive area reduction in an FPGA. It also presents the circuit implementation in a Virtex 5 FPGA, using software Xilinx ISE 10.1 tools.

Keywords: Reed-Solomon, FPGA, FEC, RS Encoding, RS Decoding

1. INTRODUCTION

The error correction codes, also known as Forward Error Correction – FEC codes, allow the recovery of a certain amount of error during data transmission without having to resend the data itself, thus increasing the system transmission capacity [1 – 3]. The high transmission rate communication systems need high performance and low cost hardware implementations of error correction codes. The block code, one of the FEC code, adds a constant size redundancy and it is capable of correcting multiple errors[4].

Among other advantages, the error correction codes provide larger communication links, power gain and inter-channel interference correction. These advantages can be translated into fewer repeaters in the system, thus reducing its final price [4]. On the other hand, the system requires some processing capability [4].

The ITU-T G.709 optical communications standard [5] recommends the use of Reed-Solomon code RS (255,239), which is implemented and discussed in this article.

2. REED-SOLOMON

Reed-Solomon is a non-linear and non-binary cyclic block code, where the symbols are formed by sequences of m-bits [6]. The code is identified as RS (*n*,*k*), where *n* is the total number of symbols in a frame and *k* is the number of data symbols, as shown in Figure 1 [1,6,7].

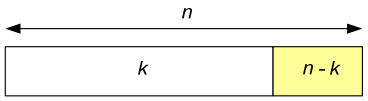

Figure 1 – Structure of a RS(*n*,*k*) code.

The Reed-Solomon code is capable of correcting *t* errors, where *t* is given by expression (1) [3, 6].

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

The RS code based system has an encoder and a decoder as indicated in Figure 2 [2, 3]. The data provided by the *Source* receives the additional check bits (n-k) at the *RS Encoder*. At the receiver, the data is decoded and up to *t* errors can be corrected. Those errors may occur during the transmission.

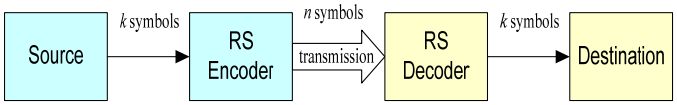

Figure 2 – Data transmission system.

The Reed-Solomon was developed based on abstract algebra and uses finite Field theory, known as Galois Field (after the French mathematician Évariste Galois). The necessary finite Field theory required for the Reed-Solomon is presented in [1, 3, 8]. The Galois Fields are implemented according to a primitive polynomial. The primitive polynomial for the RS (255,239), established by the G.709 standard [5], is given by expression (2).

$$P(X) = X^8 + X^4 + X^3 + X^2 + 1 \tag{2}$$

The polynomial must be primitive, so that all of its roots are primitive elements, represents as powers of α. It means that all field elements can be represented by using powers of α, ranging from 1 to the number elements of the field minus 1 [8]. The complete field elements can be found in [4].

Encoder

The data is systematically coded. It is first sent the original data, followed by the parity symbols, which are calculated according to the polynomial generator. It is used a "2t step" polynomial generator, given by expression (3) [1, 3, 7].

$$g(X) = \prod_{i=b}^{b+2t-1} (x + \alpha^i) \quad (3)$$

where *b* is a random number. It must be carefully chosen to avoid increasing the system complexity. The G.709 standard [5] takes *b* as 0.

Decoder

The decoder is comprised of 4 blocks: Syndrome Calculator, Key Equation Solver, Error Locator and Error Evaluator, as indicated in Figure 3 [9].

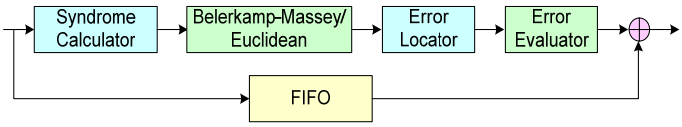

Figure 3 – The Reed-Solomon decoder.

The Syndrome Calculator is the first block. It monitors the arriving data to evaluate if data correction processing is necessary. It verifies if an error occurred during the transmission by obtaining all the polynomial roots throughout the data. If the number of errors is smaller than or equal to *t*, then the syndrome is delivered to the next block, the Key Equation Solver. If the number of errors is larger than *t*, an error signal is generated to indicate that no correction is possible. Finally, if no errors are found, the data is delivered without any further processing, besides removing the parity symbols [10].

There are two possible algorithm options for the Key Equation Solver: the Euclidian and the Berlekamp-Massey. The Euclidian algorithm is simpler to implement but requires more FPGA area due to the larger complexity of the finite field equation divider. The Berlekamp-Massey algorithm is more complex, but takes less hardware [7].

The Error Locater finds the error locations using the Chien Search algorithm. It also determines the error magnitude to be added to those locations in order to correct the symbol [7].

3. IMPLEMENTATION

The Reed-Solomon (255,239) was implemented using VHDL hardware description language, which offers high abstraction level during the implementation [11]. The VHDL description was simulated and implemented in a XC5VFX70T Virtex 5 FPGA.

Encoder

As previously described, the polynomial generator can be obtained by expression (3), where *i* is zero, according to the G.709 standard [5]. The obtained polynomial generator is given by expression (4).

A Linear Feedback Shift Register was used to implement the polynomial generator, as shown in Figure 4 [12].

$$g(X) = X^{16} + 59X^{15} + 13X^{14} + 104X^{13} + 189X^{12} + 68X^{11} + 209X^{10} + 30X^9 + 8X^8 + 163X^7 + 65X^6 + 41X^5 + 229X^4 + 98X^3 + 50X^2 + 36X + 39 \quad (4)$$

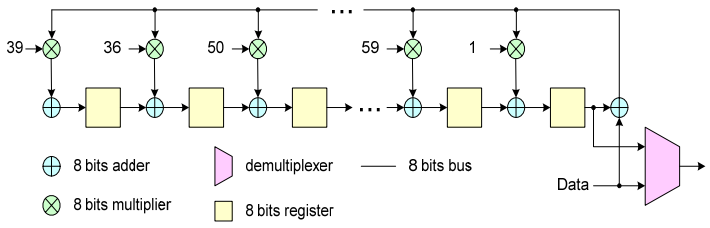

Figure 4 – Reed-Solomon encoder.

An eight bits bus interconnects the byte size registers, and the adder and multiplier finite filed arithmetic circuits. The adding circuit is implemented by using bit to bit XOR logic. The multiplication is a combination of AND and XOR gates, as described by [13].

An example of VHDL multiplication is shown in Figure 5. The multiplication is implemented as a function since it will be also used in other blocks.

```
FUNCTION alpha_225 (
    bb: STD_LOGIC_VECTOR (7 DOWNTO 0))
    RETURN STD_LOGIC_VECTOR IS
    VARIABLE cc: STD_LOGIC_VECTOR (7 DOWNTO 0);
BEGIN
    cc(0) := bb(3) XOR bb(6) XOR bb(7);
    cc(1) := bb(4) XOR bb(7);
    cc(2) := bb(0) XOR bb(3) XOR bb(5) XOR bb(6) XOR bb(7);
    cc(3) := bb(1) XOR bb(3) XOR bb(4);
    cc(4) := bb(2) XOR bb(3) XOR bb(4) XOR bb(5) XOR bb(6) XOR bb(7);
    cc(5) := bb(0) XOR bb(3) XOR bb(4) XOR bb(5) XOR bb(6) XOR bb(7);
    cc(6) := bb(1) XOR bb(4) XOR bb(5) XOR bb(6) XOR bb(7);
    cc(7) := bb(2) XOR bb(5) XOR bb(6) XOR bb(7);
    RETURN cc;
END alpha_225;
```

Figure 5 – Multiplication implemented in VHDL.

The example shows the multiplication of a Galois field element (represented by bb) by the constant element α^{225}. The AND logic does not appear in the listing due to the circuit simplification. An AND 0 operation results 0 and therefore is eliminated. An AND 1 operation is represented by the element itself. Therefore, the number of logic gates used to implement an operation depends on the operands. The same operation occurs both at the transmitter and the receiver.

The encoding of line in the OTN frame requires 16 Reed-Solomon encoders. Each line contains 3824 symbols that are supplied by the "interleaver" in a multiplexed fashion in 16 sets of 239 symbols each [5]. The symbols are transmitted in an interleaved fashion, as indicated in Figure 6.

...	2nd byte 2nd line	2nd byte 1st line	1st byte 16th line	...	1st byte 3rd line	1st byte 2nd line	1st byte 1st line

Figure 6 – OTN transmission structure.

The interleaving is used to minimize the chances of burst errors compromising the communication. In the event of burst errors, the error will not be concentrated in one line, but will be distributed over the lines, and thus increasing the probability of correcting the error. The OTN line encoding structure is shown in Figure 6.

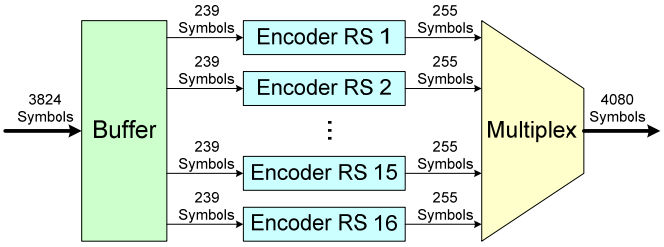

Figure 7 – Structure of an RS OTN encoder.

As indicated in Figure 7, the encoder receives 3,824 symbols. It adds the parity symbols and outputs 4,080 symbols, which corresponds to line 1 of an OTN frame.

The VHDL description was implemented in a XC5VFX70T Virtex 5 FPGA using the Xilinx ISE version 10.1. Table 1 summarizes the logic utilization.

TABLE 1 – ENCODING LOGIC UTILIZATION.

Blocks	Amount Used
Combinational ALUT	2,592
Dedicated Logic Registers	2,320
Block Memory	0

As can be observed, the FPGA does not require a large area.

Decoder

The decoder receives 4,080 symbols sequentially, corresponding to an OTN frame line. The 4,080 symbols, corresponding to 16 channels, are delivered one byte at a time, as previously shown in Figure 6.

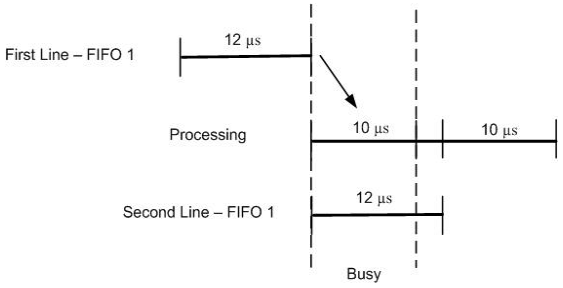

Figure 8 – Decoding timing diagram.

Since the system runs at 166 MHz and two symbols arrive at each 8 clock cycles, the FIFO indicated in Figure 3 takes 12 µs to load one frame line. The implemented decoder takes a maximum of 10 µs to process the error correction. The timing diagram in Figure 8 represents that situation.

From the analysis of the timing diagram of Figure 8, it can be observed that a single set of 16 decoders would not be enough to implement the system. The decoder can start its operation only after the FIFO is completely loaded. Therefore, while it processes one line, the other line is being transmitted and consequently must be stored. Another set of 16 decoders is necessary for the second line, requiring a large FPGA area.

The proposed solution in this work is the duplication of the Syndrome and the FIFO blocks, while maintaining a single set of the other blocks, as indicated in Figure 9.

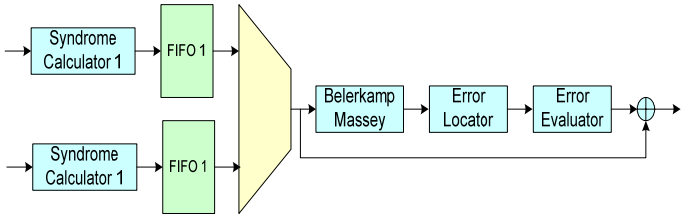

Figure 9 – Modified RS decoder.

Initially, each line passes through a Syndrome Calculator and them it is loaded into a FIFO memory. If necessary, each line is processed by the Belerkamp-Massey, Error Locator and Error Evaluator blocks. Wile the line is being processed, the other memory is loading another incoming line, as can be observed by the timing diagram in Figure 10.

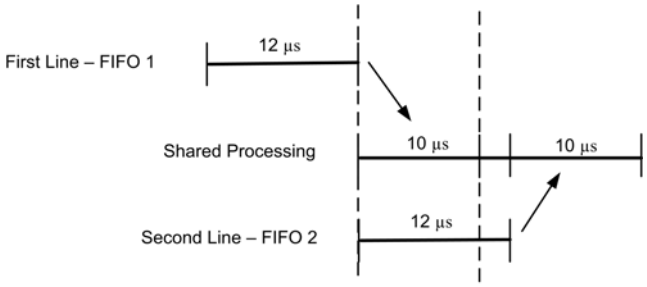

Figure 10 – Modified decoding timing diagram.

According to the timing diagram, there is a 2 µs safety margin on the system operation.

The proposed decoding architecture saves 16 Berlekamp-Massey blocks, 16 Error Locator and 16 Error Evaluator blocks. These blocks require large FPGA area, as shown by the summary presented in Table 1.

TABLE 2 – DECODING COMPARISON.

Blocks	Total Available	32 RS	16 RS
LTUs	44,800	18,383	12,270
Registers	44,800	35,144	23,991
Block Memory	148	16	16
Response Time	-	5 ns	5.1 ns

As can be observed, the proposed decoding architecture saves FPGA area. The number of registers is reduced by 14% and the number of look-up tables is reduced by 25%, while delay is increased by just 0.1 ns (increase of 2%) due to the inclusion of a multiplexer. The delay is acceptable, considering that a clock cycle at 166MHz requires 6 ns. Therefore, the OTN frame decoding requires just 16 RS decoders instead of 32.

As can be observed from the decoding structure of Figure 11, the 4080 symbols are decoded by 16 RS (255,239) decoders that produce 3,824 symbols, which correspond to an OTN frame line without the parity symbols.

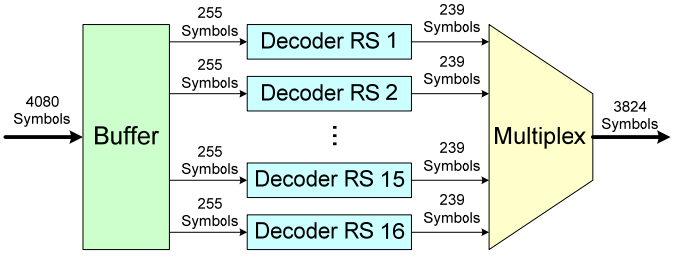

Figure 11 – OTN decoding structure.

4. CONCLUSIONS

This article presented a new Reed-Solomon decoding structure that can save a lot of FPGA area. The saved area represents a cost reduction for the system since a smaller and cheaper FPGA can be used.

5. ACKNOWLEDGMENTS

The authors would like to thank the Microelectronics Group at Universidade Federal de Itajubá.

6. REFERENCES

1. R. H. Morelos-Zaragoza, *The Art of Error Correcting Coding*, p. 1, 74, John Wiley & Sons Ltd, Chichester (2006).
2. A. Betten, *Error-Correcting Linear Codes: Classification by Isometric and Applications*, p. 7, 4, Springer, Berlin (2006).
3. J. C. Moreira and P. G. Farrell, *Essentials of Error-Control Coding*, p. 2, 166, 3, 166 , John Wiley & Sons Ltd, Chichester (2006).
4. M. A. Ingale, *Error Correcting Codes in Optical Communication Systems*, Gothenburg (2003).
5. ITU-T G.709/Y.1331, *Interfaces for Optical Transport Network (OTN)*, (2001).
6. B. Sklar. *Reed-Solomon Codes*, (2001).
7. K. C. C. Wai and S. J. Yang, *Field Programmable Gate Array Implementation of Reed-Solomon Code, RS(255,239)*, New York (2006).
8. A. Ferreira, *Aplicação da Teoria dos Campos de Galois na Codificação de Canal*, Lisboa (1999).
9. M. Song, S. Kuo, I. Lan, in *Consumer Electronics, IEEE Transactions*, p. 265-273, A Low Complexity Design of Reed Solomon Code Algorithm for Advanced RAID System, Rosemont, IL (2007).
10. T. Le-Ngoct, M. T. Vot, B. Mallettt and V. K. Bhargava, in *Military Communications Conference, 1990. MILCOM '90, Conference Record, A New Era/1990*, p. 121-125, A gate-array-based programmable Reed-Solomon codec:structure-implementation-applications, Monterey, CA (1990).
11. S. Smith, M. Benaissa and D. Taylor, in *VHDL (Very High Speed Integrated Circuits Hardware Description Language) - Applications and CAE Advances/1993*, High Level Synthesis of an (N,K) Reed-Solomon Encoder Using VHDL, London (1993).
12. J. Koeter, *What is an LFSR?*, Texas Instruments Incorporated, Dallas (1990).
13. A. Halbutogullari and Ç. K. Koç, *in IEEE Transactions on Computers/2000*, p. 503-518, Mastrovito Multipler for General Irreducible Polynomials, Corvallis (2000).