A Reed-Solomon algorithm for FPGA area optimization in space applications

Gabriel Marchesan Almeida*, Eduardo Augusto Bezerra*, Luis Vitório CargniniΨ, Rubem Dutra Ribeiro FagundesΨ and Daniel Gomes Mesquita†

* Embedded Systems Group, PPGCC, Faculdade de Informática, Catholic University (PUCRS)
galmeida@inf.pucrs.br, Eduardo.Bezerra@pucrs.br
Porto Alegre, RS, Brazil, 90619-900

Ψ Instituto de Pesquisas Científicas e Tecnológicas, IPCT, PPGEE, FENG, PUCRS
lvcargnini@ieee.org, rubemdrf@sapienstek.com
Porto Alegre, RS, Brazil, 90619-900

† Instituto de Engenharia de Sistemas e Computadores, Instituto Superior Técnico de Lisboa (INESC)
Lisboa, Portugal, 1000-029
mesquita@inesc-id.pt

Abstract
This work describes an algebraic based design strategy targeting area optimization in reconfigurable computer technology (FPGA). Area optimization is a major issue as smaller components allow for better system adaptation, which is an expected feature of reconfigurable systems for space applications. The approach is applied in the design stage of a component for the communications module of an on-board computer system. The chosen component is a Reed-Solomon encoder, which has been implemented using a Hardware Description Language (VHDL) according to CCSDS recommendations, and targeting an FPGA platform. The paper investigates traditional alternatives for the encoder implementation, introduces the algebraic theory behind the proposed approach, describes the design process and discusses the area figures reached by the new design.

1. Introduction

An SRAM based Field Programmable Gate Array (FPGA) is an interesting platform for the implementation of evolvable hardware systems. In the field of space applications, commercial off-the-shelf FPGA devices from leading manufacturers [1] have been around for a while. However, these devices are still not in use as the main processing unit of on-board computers in space applications, mainly because of the risks related to the exposure of SRAM technology to radiation. Even though, scientists from space agencies around the world, and also from the Brazilian Space Program, are interested in this technology for several reasons [2][3]: easy adaptation to new mission requirements after launching; allows “hardware” bug fixes after launching; higher processing power; and reduced size. All these features are of great importance for space systems in general, but the idea behind having a hardware system changing on-the-fly in order to adapt to its environment and application requirements, is the main motivation for the research work at PUCRS [4][5][6].

An on-going project at PUCRS, funded by the Brazilian Space Agency, aims the implementation of the communications module for a future satellite mission, targeting reconfigurable computing technology (FPGA). This on-board module is responsible for receiving telecommands (TC) from a ground-station, and for sending out telemetry (TM) data, as shown in Figure 1. It is a vital module for the mission as a whole, as in case of a problem, a TC may not arrive at its destination resulting, for instance, in an engine for attitude control turned on/off at the wrong moment, or the solar panel pointed at the wrong direction. In order to make the reconfigurable computing system as robust as possible, several strategies have been investigated by the research group at PUCRS, and in other institutions as well: design for verification [7][8], design for testability [9][10], fault-tolerance [11][12], among others. All these efforts are justified as having a dependable system is the first step for a successful space mission.

![Figure 1. TC/TM flow](image-url)
EDAC, like Bose-Chaudhuri-Hocquenghem (BCH) and environment. There are many techniques to implement transmitted data unchanged under noisily hostile susceptible to burst errors, Reed-Solomon codes usually afterwards. For digital communication channels whereas RS codes group the bits in blocks to correct them can correct a given number of bits at any position, arithmetic, also known as Galois Field \( [14] \). BCH codes with multiple faults, and are based on finite-field communication systems, whenever it is necessary to keep noisy channel. It has been applied to a wide range of well-known techniques to protect transmitted data across a communication systems, in order to have a full communications system, and the present work selects one of them to discuss the algebraic strategy used in the design. A crucial component of the communications module is the algorithm for generating Error Detection and Correction Code (EDAC). EDAC is a well-known technique to protect transmitted data across a noisy channel. It has been applied to a wide range of digital communication channels, whenever it is necessary to keep transmitted data unchanged under noisily hostile environment. There are many techniques to implement EDAC, like Bose-Chaudhuri-Hocquenghem (BCH) and Reed-Solomon (RS) codes \([13]\). These codes can cope with multiple faults, and are based on finite-field arithmetic, also known as Galois Field \([14]\). BCH codes can correct a given number of bits at any position, whereas RS codes group the bits in blocks to correct them afterwards. For digital communication channels susceptible to burst errors, Reed-Solomon codes usually present the best coding scheme option. RS is a very popular error correcting code, and has been applied to several situations, especially in communication systems \([15]\).

The present work discusses the design and implementation of an optimized RS core for encoding operation. All the core components have been implemented targeting FPGA area optimization and high-speed throughput in the encoding process. This area optimization investigation is of great importance for the satellite communications module as a whole, as more components could be used at the same time in the FPGA, or they could be loaded and unloaded on demand, improving system adaptation features.

The paper starts by introducing the Consultative Committee for Space Data Systems (CCSDS) recommendation for TC and TM. Next, the Reed-Solomon (RS) algorithm is detailed and the necessary mathematic background is presented. The proposed RS core is introduced, following by its implementation details and practical results analysis.

2. Background and Related Work

2.1. CCSDS Telecommand and Telemetry

The implementation of TC and TM systems for space applications following CCSDS recommendations is a trend among space agencies around the world. CCSDS is the international organization responsible for making recommendations for the implementation of TC and TM systems \([16] [17]\). The basic flow for a ground station to send TC and to receive TM is as follows: first, the frame to be sent is built as a CCSDS standard package. After, the TC is encoded using the error correcting code BCH. The TC is then sent to the space segment, which will have a module responsible for receiving and decoding the TC data. This module is also responsible for the correction of possible errors occurred during the transmission process. The TC frame is then processed on-board, and the TC functions are distributed to the different instruments of the spacecraft.

The basic flow for a ground station to receive TM is very similar to the one for sending TC. The main difference is that the frame is encoded using the Reed-Solomon algorithm.

2.2. The Reed-Solomon Algorithm

RS is an error correcting code designed to correct multiple errors \([15]\). The specification of an RS code is \(RS(n, k)\). Each symbol (block of bits) has \(m\) bits. The total number of symbols used in the code is represented by \(n\), and the number of symbols used to store useful information is \(k\). Then, \(m \times n\) is the number of bits of information to be encoded. The remaining symbols, \((n-k)\), are used as parity symbols. For a given symbol width, \(n\) has a maximum value of \(2^m-1\). The correction capacity, \(t\), is given by the number of parity symbols divided by two, e.g. \(2t = n-k\), as shown in Figure 2.

The theory of RS codes is based on finite-field arithmetic, e.g. Galois Field (GF). Two arithmetic operations, addition and multiplication, can be defined for GF\((2^m)\) finite field. The rules of addition define that the sum of any field elements can be obtained by adding (modulo-2) the respective coefficients of their basis elements. The multiplication rules follow the usual
procedure in which the product of the field elements is obtained by adding their exponents module-(2^m–1).

A convenient way to perform both multiplication and addition in GF(2^m) is to use two look-up tables [13]. This allows one to change between polynomial (vector) and power representations of a GF(2^m) element.

![Figure 2. Reed-Solomon codeword](image)

Anti-log tables A(i) are useful when performing additions. The tables give the value of a binary vector, represented as an integer in natural representation, A(i), that corresponds to the element a^i, which is an element belonging to GF. The log tables L(i) are used when performing multiplications. These tables give the value of a power of alpha, a^{L(i)} that corresponds to the binary vector represented by the address i, i.e., a^{L(i)} = A(i).

**Example:** In order to understand the tables, consider GF(2^3) having a primitive polynomial p(a) = a^3 + a + 1, and a^3 = 1 (also, a = a, a^0 = a^0, …). The log and anti-log tables are shown in Table 2.

![Table 2. Log and Anti-log tables](table)

Using the log and anti-log tables, the computation of \( \gamma \) proceeds as follows: \( \gamma = A(L(A(3) \oplus A(5)) \ast 3 + 1) = A(L(3 \oplus 7) \ast 3 + 1) = A(L(4) \ast 3 + 1) = A(2 \ast 3 + 1) = A(7) = A(0) = 1 \). In the last step, it is straightforward that \( a^7 = 1 \).

On the other hand, considering the classical way of processing an element \( \gamma = a(a^3 + a^5)^3 \) in its vector form, and using the properties of GF(2^3), \( \gamma \) can be obtained as follows: \( a^3 + a^5 = 110 \oplus 111 = 001 = a^2 \). Thus, \( \gamma = a(a^5)^3 = a^{1+6} = a^7 (=1) \). In this case, two matrices, multiply and addition, are needed in order to obtain the operation result. Through log and anti-log tables, the same result can be achieved using just two vectors.

### 2.3. Related Work

In [18], the authors present a high level technique based on correcting codes to protect SRAM memories against multiple upsets (radiation related problem). Their technique combines Reed Solomon code and Hamming code to assure reliability in presence of multiple bit flips with reduced area and performance penalties. Multiple upsets were randomly injected in several combinations of memory cells to evaluate the method robustness. The experiment was emulated in a Xilinx Virtex FPGA platform. The Reed-Solomon encoder standard used by the technique was RS(155,145), which is able to correct up to 5 bits of errors if they are in the same symbol. Table 3 presents their obtained results.

In [19], the authors show a new technique to further improve the RS core proposed in [18], by individually optimizing the multipliers used in the RS algorithm. Table 3 shows the area cost for their strategy, comparing to [18] and to our approach proposed in the present work, which will be discussed in the next sections.

### 3. A Reed-Solomon encoder core

A Linear Feedback Shift Register (LFSR) has been used in the RS implementation. An LFSR has a shift register and a feedback function, which are used to shift the contents of the LFSR around the registers or to the output. An LSFR can be used among others, in a serial/parallel conversion, or to delay a serial bit stream. In our RS implementation an LFSR is used for mapping symbols (field elements) in terms of its basis elements. Figure 3 shows the LFSR circuit generating (with m=3) the 2^m–1 nonzero elements of the field. The feedback connections are the coefficients of the polynomial \( f(x) = 1 + X + X^3 \). By initializing the circuit, feeding it with \( a^1 a^3 a^5 \), and performing a right-shift at each clock cycle, it is possible to observe that each of the symbols (field elements) will appear in the stages of the shift register. In this example, the three symbols to be encoded are represented in binary notation as follows (for mapping details see Figure 8.7 in [20]):

\[
\begin{array}{ccc}
0 & 1 & 0 \\
0 & 1 & 0 \\
1 & 1 & 1 \\
\end{array}
\]

\( \alpha^1 \quad \alpha^3 \quad \alpha^5 \)

The nonbinary operation implemented by the RS encoder of Figure 3, generating codewords in a systematic format, proceeds in the same way as the binary one (i.e. BCH).
After the third clock cycle, the register contents are the 4 parity symbols, $a^0$, $a^2$, $a^4$, and $a^6$, as shown in Figure 4.

Finally, switch 1 is opened, switch 2 is toggled to the up position, and the parity symbols are shifted to the output. The output codeword written in polynomial form can be expressed as follows:

$$U(X) = a^0 + a^2X + a^4X^2 + a^6X^3 + a^1X^4 + a^5X^5 + a^8X^6$$

$$= (100) + (001)X + (011)X^2 + (101)X^3 + (010)X^4 + (110)X^5 + (111)X^6$$

At this point it is important to highlight that the LFSR’s add and multiply operations are very area consuming when implemented in hardware. An important contribution of this work is to minimize the circuit size, by replacing add and multiply operations by two matrices. These matrices have all elements of $GF(2^8)$ in its rows and columns and the operation results (add and multiply) in its cells.

The CCSDS recommendation suggests the coding format RS(255,223). Following the matrices approach, we would need to use 2 tables of dimensions 255 x 255 each, representing the 255 elements of the $GF(2^8)$. This would result in 65,025 symbols to represent the addition table and 65,025 symbols to represent the multiplication table, meaning 130,050 symbols to represent both arithmetic tables. This approach is better than the first one, but it is still not appropriate for hardware implementation, as the area occupation can be further optimized by the proposed approach.

In our implementation, we have used two vectors to represent the log and anti-log tables introduced in Section 2.2. When performing sum operations between two elements, we must search Table 2 looking for the anti-log values that correspond to the address $i$, which represents the power of the element (see Example in Section 2.2). For multiply operations, the process is almost the same, as the search must be performed in the log column instead of anti-log. This proposed approach represents a reduction around 255 times in area consumption, as it would need only 510 symbols in the log and anti-log tables against the 130,050 required by the add and multiplication matrices.

The proposed RS core has been implemented using the hardware description language VHDL [21]. This allows the Intellectual Property (IP) core generation which could be employed to build a System-on-a-chip (SOC). FPGAs represent a low cost fast time-to-market alternative for hardware implementation and validation. The RS core is designed targeting a hardware implementation, and its VHDL description can be synthesized for FPGA devices. Despite these advantages, as stated before, SRAM based FPGA dependability issues are still limiting their widespread use in mission critical applications, but there are several on-going research works dealing with this matter [9][11][22].

The first step in a VHDL design is to define the circuit interface. This is done by writing an “entity” block which has all the input and output signals. The entity for the RS encoder is shown in Figure 5. The RS core has three input and two output signals: a clock signal, responsible for the system timing and control; a reset signal which is enabled to start the processing; a data signal which receives the 223 data symbols to be encoded; the ready signal changes from low to high to indicate that the processing is done, and a new code word has been generated; the encoded_data signal outputs 255 symbols, where 223 are data symbols and 32 are parity symbols generated by the RS encoder.

After describing the circuit’s interface, the LFSR and the tables are written in VHDL. The logic synthesizes is then performed, in order to validate all implemented functions of the Reed-Solomon algorithm.

After the synthesis process, the circuit can be simulated. A testbench has been written in order to generate input signals ranging from “1” to “223”, represented in binary notation having 8 bits per symbol. Initially, the circuit is fed with the 223 input symbols, and the reset signal goes high to start the processing. Figure 6 shows a simulation output for the RS core initialized with input data. This is the first step in the circuit validation process.

After the initialization step, the input data are processed by the functions implementing the LFSR and the log/anti-log strategy described before. At the end of the process, the circuit outputs the 32 parity symbols.
calculated by the CCSDS RS core. These symbols are then attached to the message before being sent out to a receiver. Figure 7 shows this output.

Having concluded the logical synthesis and the RS core validation through simulation, the next step is the physical synthesis. Figure 8 shows the modules to be synthesized in order to generate a hardware representation of the system, which will be used to configure the FPGA.

An extra IP core, shown in Figure 8, was necessary to allow the serial communication between an RS-232C host computer interface and the FPGA hardware device. This IP core was also developed at PUCRS [23]. A piece of software, written in Java, is used to provide the serial communication between the host computer and the FPGA.

In our case study, the Java application sends 223 TM bytes to the RS-232C IP core plus 1 byte for synchronization (SB). Next, the RS-232C IP core sends the 223 TM bytes to the RS IP core which returns the 32 parity bytes to the Java application. Figure 9 shows the Java application, sending TM data to the RS core. Through several analyses of the received parity symbols (“Receiver” box in Figure 9), it is possible to conclude that all modules of the implemented RS IP core are functional and correct.

The FPGA board indicated in Figure 8 has a Virtex-II Pro FPGA device XC2VP20 from Xilinx [1]. All experiments have been performed using this platform. The hardware area usage has been measured in terms of lookup tables (LUTs), which are the smallest logic units present in a Xilinx FPGA.

The RS IP core approach proposed in this paper is compared against approaches proposed by other authors, as discussed in the related work section. In our approach, the RS encoder has used just 48 LUTs against 215 [18] and 134 [19], respectively.

<table>
<thead>
<tr>
<th>FPGA Device</th>
<th>Available area (#4-LUTS)</th>
<th>Occupied area (#4-LUTS)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Our approach</td>
<td>XC2VP20</td>
<td>18.560 (100%)</td>
</tr>
<tr>
<td>RS/Hamming [18]</td>
<td>XCV600E</td>
<td>13.824 (100%)</td>
</tr>
<tr>
<td>RS-OPGE [19]</td>
<td>XCV600E</td>
<td>13.824 (100%)</td>
</tr>
</tbody>
</table>
From these results, it is possible to observe that in our approach the RS encoder has used much less area than the optimized approaches presented in [18] and [19].

Table 4 presents the synthesis results for the proposed RS core. We use the ISE tool from Xilinx for both, the logical and the physical synthesis, and also to download the configuration file to the FPGA. The circuit presented in this work has used just 47 LUTs out of 18,560 available in the selected FPGA device.

### Table 4. Encoder area summary after synthesis of the RS encoder core

<table>
<thead>
<tr>
<th>Selected Device FPGA: Xilinx xc2vp20-7ff1152</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Slices: 28 out of 9,280</td>
</tr>
<tr>
<td>Number of Slice Flip Flops: 35 out of 18,560</td>
</tr>
<tr>
<td>Number of 4 input LUTS: 47 out of 18,560</td>
</tr>
<tr>
<td>Number of bonded IOBs: 3 out of 564</td>
</tr>
<tr>
<td>Number of GCLKs: 1 out of 16</td>
</tr>
</tbody>
</table>

Figure 10 shows the actual execution time, captured from the FPGA board using a logic analyzer. It is possible to observe that the execution time of the RS core is just 4.460µs. Unfortunately, the execution time is a piece of information missing in the related work, and a comparative analysis could not be performed.

Figure 10. Execution time: logic analyzer screenshot

5. Conclusion

This work shows that when employing the algebraic approach in the design of RS algorithms, the final FPGA implementation takes less area than the traditional implementations.

The results prove the algebraic approach to be a good alternative to replace not only the traditional RS hardware implementation using adders and multipliers, but also the RS software based matrices algorithms. This new approach would be interesting for error correcting codes in communication systems, when targeting area optimization and speed in the RS encoding process. The concern with area optimization is of great importance when considering the target application for this work, which is the on-board computer of a spacecraft.

An important finding of this work is the impressive reduction of around 255 times in area consumption of the log/anti-log proposed strategy, when compared to the add and multiplication matrices method. As shown in Sections 3 and 4, the log and anti-log tables use only 510 symbols, when the matrices approach needs 130,050.

The complete CCSDS TM/TC design is described in the MSc dissertation of one the authors [24]. The BCH module design and implementation, following also an algebraic approach, has been accepted for publication in another conference [25]. Other important modules needed to introduce system dependability are not shown in Figure 8, but they are well discussed in past publications [7][9][11].

As a future work, all the implemented IP cores will be integrated, in order to build a full communication system, in a single FPGA, device for the Brazilian Space Program, following the CCSDS standard.

References


