A Self Checking Reed Solomon Encoder: Design and Analysis

G.C. Cardarilli, S. Pontarelli, M.Re, A. Salsano
{marco.re, g.cardarilli}@ieee.org
{pontarelli,salsano}@ing.uniroma2.it
University of Rome "Tor Vergata", Department of Electronic Engineering
Rome, ITALY

Abstract

Reed Solomon codes are widely used to identify and correct data errors in transmission and storage systems. Due to the vital importance of these blocks, a very important research topic is the study of the effects of faults on their behavior. The presented architecture exploits some properties of the arithmetic operations on GF(2^n) Galois Field, related to the parity of the binary representation of the elements of the field. The encoder has been mapped on an SRAM based FPGA, the self-checking property has been analyzed using a SEU fault model and the performances in terms of area and delay overhead are presented.

I. Introduction

High reliable data transmission and storage systems frequently use Error Correction Codes (ECC) to protect their data. So they are able to detect errors in binary configuration allowing, under suitable assumptions, the possibility to correct the coded words. The performance of a code is measured in terms of the maximum number of wrong bits it is able to detect in a coded word and the maximum numbers of bits it is able to correct. Other key element is the circuit complexity required for code realization. In fact, the actual implementation of a coding procedure on a real system requires the development of two basic blocks, the encoder and the decoder. The first one, starting from the non coded word computes the code bits realizing the codeword (composed by the data and code bits) that is the protected data to be stored or transmitted. On the contrary, the decoder receives in input the codeword and checks its correctness eventually correcting the wrong bits. In order to meet the system constraints, frequently these blocks must have high performance in terms of speed and error correction capabilities in order to process a large amount of data preserving their integrity. Large efforts are devoted to develop codes with increased capabilities of error detection and correction , but, normally, larger performances correspond to greater efforts for developing encoders and decoders. Usually system designers focus their attention on the data encoding and decoding performed at the system input and output, supposing that the primary goal is to protect data during their flow inside the system. In this approach very critical points are the input and output circuits, since any error in these circuits may introduce catastrophic effects on the overall system. A fault in the encoder can produce a non correct codeword, while a fault in the decoder can give a wrong data word even if no errors occurs during the transmission of the codeword. Moreover these errors will be present in each data flowing in the systems. Therefore great attention must be paid in order to detect and recover faults in encoding and decoding circuitry. These faults are caused either by the conventional sources as, for example, the fail of the technological process, the aging of the electronic devices or by new phenomena related to the new technologies since the shrinking of the elementary devices in the electronic systems causes a greater susceptibility of the components to the external environment. One of the main concerns in this field is related to the effects of radiations on silicon devices. A high energy ion during its traveling trough the device can inject charges into the substrate and these charges, due to the electrical biasing, can reach the active elements (transistors) changing the content of storage elements. These effects known as Single Event Upsets (SEU) have been widely studied for applications related to space environment [1], where they are very frequent and predominant with respect to other possible failures . Moreover, SEU's have been recently observed and studied at sea level [2] and in aircraft electronics [3] and therefore they are expected to be an important issue in the next years. ECC are widely used in space applications for the design of space-borne mass memories [4] and for the transmission of the collected data to the earth stations. These applications require high reliability, and related systems must be tolerant to the effects induced by mechanical stresses, thermal stresses and, especially, r related SEU phenomena. Nowadays, the most used error correcting codes are the Reed-Solomon codes, based on the properties of the finite field arithmetics. In particular, the finite fields with a number of elements of 2^n are suitable for a digital implementation due to the isomorphism between the addition operation, that is performed modulo 2, and the xor operation between the bits representing the elements of the field. We propose to exploit this relationship to detect faults that can occurs in the encoders, achieving the self-checking property for the arithmetic structures used in the design of the circuitry, and therefore in the overall encoder. The encoder is then implemented on a FPGA and the proposed solution is evaluated. The paper is organized as follows: Section II describe the fault model used for the FPGA, Section III illustrates the used background of the Reed Solomon codes, and Section IV describes the properties of the arithmetic structures used in the Reed Solomon encoders with respect to the parity of the arithmetic operands. In Section V the architecture of the proposed self-checking Reed Solomon encoder is presented while in Section VI some evaluation in term of area and delay overhead are provided. Finally, conclusions are drawn in Section VII.

II. SEU INDUCED FAULTS IN FPGA

An overview of the fault modeling for a SRAM based FPGA, with respect to the SEU occurrence is presented in this section. The proposed fault modeling makes use of the method recently proposed in [5] and used in [6] for the analysis of the routing error caused by a SEU in the configuration memory of the SRAM FPGA. A typical SRAM based FPGA is basically composed by an array of CLB (Complex Logic Blocks), providing a LUT for combinatorial operations and a memory element like a configurable flip-flop, and by a great number of interconnection resources grouped in interconnection matrix blocks. The configuration bits of the LUT and of the routing resources are stored in the so-called configuration memory. During the operating life of the application implemented in the FPGA a SEU can affect either the configuration memory of the FPGA or the memory elements used by the application. Therefore the effect of SEU on an FPGA can be divided as follows:

1) SEU in registers
2) SEU in LUTs
3) SEU in routing configuration RAM

A SEU in the memory element of the CLB can be seen as a bit-flip, changing the context of the bit stored in the flip-flop from 1 to 0 or from 0 to 1. A SEU in the LUT changes the functionality provided for the combinatorial operation. When the memory location affected by a SEU is addressed by the inputs signals the output provided by the LUT is the opposite of the expected one. For the SEU affecting the routing configuration RAM two case can show up:

1) open defect
2) short defect

In the fault free situation the configuration SRAM enables some interconnections in the matrix in order to connect different CLB's together. In Fig. 1 an example of the interconnection matrix of a Xilinx Virtex FPGA is presented, in which two interconnection are used to connect two adiacent CLB's with the other parts of the system.

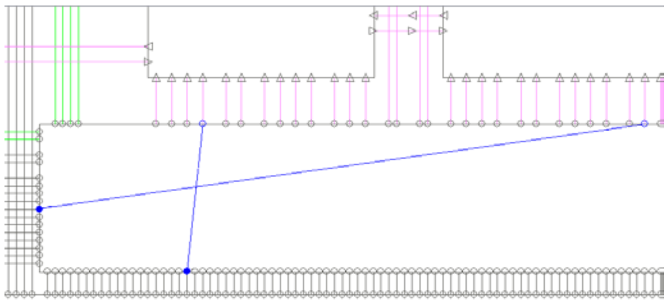
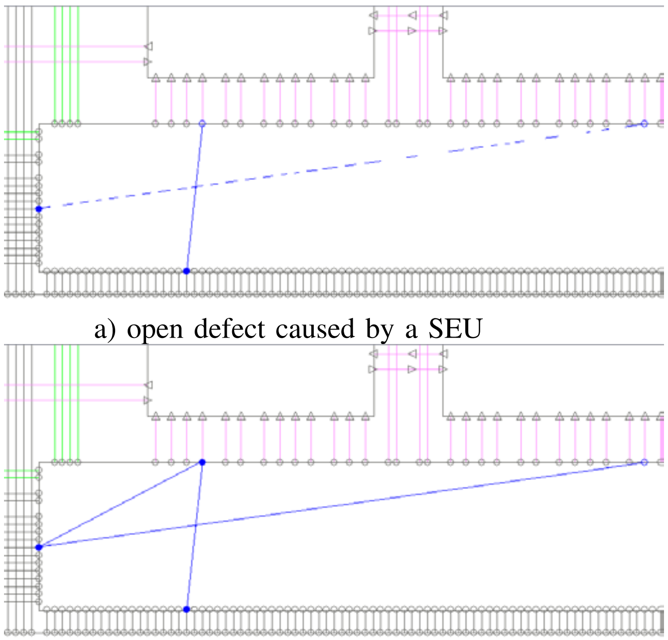


Fig. 1. A Fault-free interconnection matrix with two routed nets

It must be noticed that only few routing resources of the FPGA are used for each configuration of the device, while the other interconnecting resources remain unused. In the same way, although SEUs affecting the routing configuration memory may have many different effects, only a few of them are meaningful and corres modification of the netlist realized by the original configuration. In fig. 2a we show how a SEU can modify one net segment, interrupting the signal propagation from the CLB. This kind of defect can be modeled as an open defect. Instead, in fig. 2b the SEU affecting the configuration memory cause the activation of a routing net that connect the two original nets of the configuration, causing a defect modeled as a short defect.



a) open defect caused by a SEU

b) The two net are shortened by the SEU

Fig. 2. Defect in the routing resources

In [5] the open defects are supposed equivalent to stuck-at 0 or stuck-at 1 defect, while the short defect can be characterized as either a wired-AND or a wired-OR model. Two special cases of short defects are stuck-at-0 and stuck-at-1 defects, where a line is shortened with the ground or power line respectively. In [6] the same defect (open and short) are described, while the effect of these defects at logic level are assumed as unknown logic values for the open defects and indefinite logic values for the short defects. In our work we model the SEU in the registers, the SEU in the LUTs and the open defects as faults affecting only one resource of the logic netlist implemented in the FPGA. In other words we model these kinds of defects at a higher abstraction level and consider all these defects as a erroneous value in an input or an output of one of the LUT or memory elements composing the circuit realizing the RS encoder. Analogous considerations can be done for the fault model of short defects: let us suppose that the short defect affecting two nets named A and B is modeled as a wired-AND (see fig.3). The two nets A and B corresponds to a block with two inputs Ain and Bin and two outputs Aout and Bouts. When the inputs of this block are the identical, Ain,Bin=(1,1) or (0,0), the output of the block is the same of the fault-free configuration. When the two inputs differs the two outputs provides a value of 0, and therefore only the output of one net differs from the fault-free configuration. This behavior can be considered as a fault that affect only one component of the logic netlist and therefore we can work at the abstraction level of a logic netlist composed of LUT and flip-flop in order to design the self-checking Reed Solomon encoder.

III. Background of Reed Solomon Codes

It is well know that given a prime number p and a number $n \in \mathbb{N}$ a field of p^n element can be constructed. This fields are known as Galois Field GF(p^n). For $n = 1$ the field is composed by the elements [0,1,..p-1] and addition and multiplication are performed modulo p. For $n > 1$ the Galois field is constructed as follows:

- the elements of the field are the polynomial $p(x)$ of degree $n - 1$ with coefficients in GF(p)
- addition and multiplication are performed modulo $i(x)$, where $i(x)$ is an irreducible polynomial of degree n.

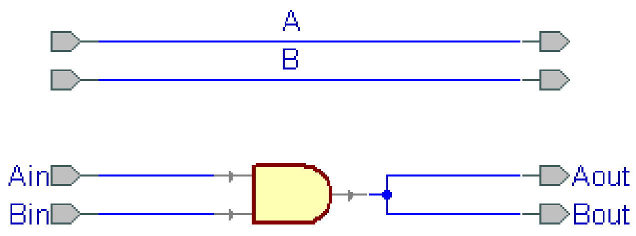


Fig. 3. Short defect modeled as a wired-AND

The finite fields used in digital implementations are in the form GF(2^s), where s is used avoid confusion in the notation and represents the number of bits of a symbol. An element $a(x) \in GF(2^s)$ is a polynomial with coefficients $a_i \in 0, 1$ and can be seen as a symbol of s bits $a = a_s \dots a_1 a_0$. The addition of two elements $a(x)$ and $b(x) \in GF(2^s)$ is the sum modulo 2 of the coefficients a_i and b_i, therefore can be seen as the bitwise xor of the two symbols a and b. The multiplication of two elements $a(x)$ and $b(x) \in GF(2^s)$ requires the multiplication of the two polynomial followed by the reduction modulo $i(x)$. These operation can be implemented as an AND-XOR network, as explained in [7],[8].

Now we can introduce the RS(n,k) code, where the symbols composing the data word are represented as elements of the field GF(2^s) and the overall data word is treated as a polynomial $d(x)$ of degree k with coefficient in GF(2^s). In other word $d(x)$ is an element of GF(2^s)[x], the ring of polynomial with coefficient in GF(2^s).

A Reed-Solomon codeword is generated using a special polynomial $g(x)$. All valid codewords are exactly divisible by the generator polynomial. The general form of the generator polynomial is:

$$g(x) = (x + \alpha^i)(x + \alpha^{i+1}) \dots (x + \alpha^{i+2t}) \tag{1}$$

where $2t = n - k$ and α is a root of the irreducible polynomial $i(x)$ used to construct the field.

The codeword of a separable RS(n,k) code can be seen as $c(x)$, an element of the ring GF(2^s)[x] with degree n that can be constructed in the following way:

$$c(x) = d(x) \cdot x^{n-k} + p(x) \tag{2}$$

$$p(x) = d(x) \cdot x^{n-k} \ mod \ g(x) \tag{3}$$

where $p(x)$ is an element of the ring GF(2^s)[x] with degree less than $n - k$ that represents the parity symbols. This means that the encoder takes k data symbols and adds $2t$ parity symbols to make an n symbol codeword. The $2t$ parity symbols allows to correct up to t symbols that contain errors in a codeword. Given a symbol size s, the maximum codeword length n for a Reed-Solomon code is $n = 2^s - 1$. For example, the maximum length of a code with 8-bit symbols (s=8) is 255 bytes.

IV. Parity of arithmetic structures in GF(2^n)

In this section the characteristics of the arithmetic operations in GF(2^s) used in the Reed Solomon encoder will be analyzed with respect to the parity of the binary representation of the operands. Two operation will be taken into account:

- Parity of adders in GF(2^n)
- Parity of constant multiplier in GF(2^n)

First of all we define the parity P($a(x)$) of a symbol as the xor of the bits a_i composing the symbol. For GF(2^n) the addition operation can be performed simply xoring the bits of the same index, therefore the following property can be easily demonstrated:

$$P(a(x) + b(x)) = P(a(x)) \oplus P(b(x)) \tag{4}$$

For the multipliers in GF(2^n) we focus our attention only to the multiplication for a constant symbol g, because in the Reed Solomon encoder the irreducible polynomial used to encode the data is constant and the polynomial multiplication can be implemented starting from the multiplication for the constant g_i, where g_i are the coe of the irreducible generator polynomial $g(x)$. In this case the multiplier can be implemented using an suitable network of xor gates. As an example we shown in fig. 4 the and-xor network implementing the two LSB of a multiplication over GF(2^4).

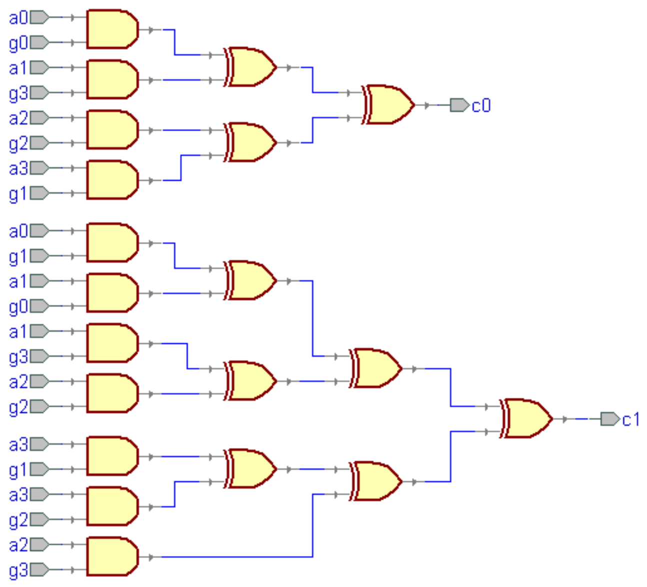


Fig. 4. GF(2^4) multiplier

If we suppose the constant g equal to 1010 the network is reduced to the one in fig. 5.

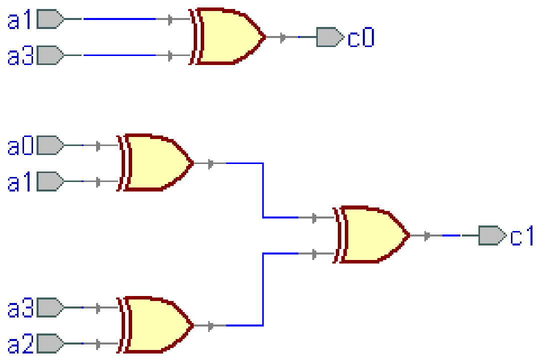


Fig. 5. GF(2^4) multiplier

Therefore, each bit of the result of the constant multiplier can be computed xoring some input bits depending from the constant g and from the chosen $i(x)$ polynomial described in section II.

It must be noticed that, if an input a_i is evaluated for an odd number of output bits, the parity P($c(x)$) of the result depend from the parity of the inputs bits. In other words, the parity of the result can be evaluated as:

$$P(c) = \bigoplus_{i \in A} a_i \tag{5}$$

where A is the set of inputs that are evaluated an odd number of times.

We propose to modify the constant multiplier block reporting as additional outputs the inputs bits that are evaluated an even number of times. The proposed modification is can be explained using the concept of "‘odd observability"’ proposed in [9]. In this way, the parity of the output word o is:

$$P(o) = P(c) \oplus P(copy) = P(a) \tag{6}$$

V. The Self Checking Reed Solomon Encoder

The implementation of a Reed Solomon encoder is mainly realized through an LFSR, which implements the polynomials division over the finite field [10].

The fig. 6 shown the implementation of an RS encoder. The additions and multiplication are performed on GF(2^s) and the constant multiplication g_i represent the coefficients of the generator polynomial $g(x)$.

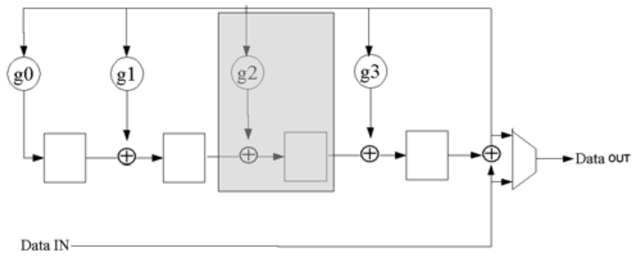


Fig. 6. RS encoder

The RS encoder can be constructed using the slice block composed by a constant multiplier, an adder and a register (the shaded block in fig. 6). The number of slices for a RS(n,k) code is $n - k$. The self-checking implementation requires the insertion of some parity prediction blocks and of a parity checker block. We propose to check the correctness of each slice using the structure presented in fig. 7.

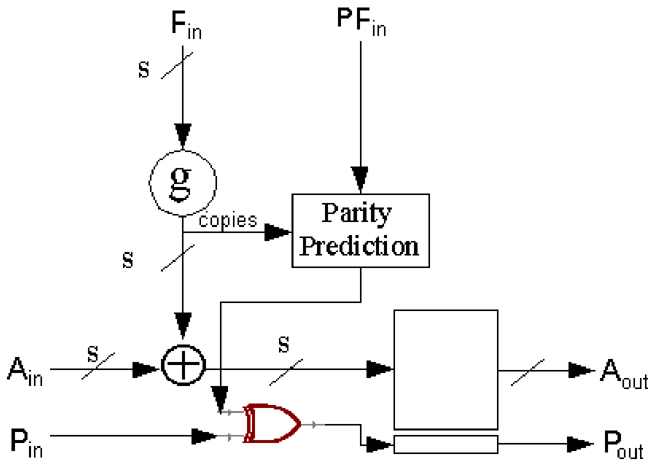


Fig. 7. Self-Checking Slice

The I/O of each slice are:

- A_{in} is the registered output of the previous slice
- P_{in} the registered parity of the previous slice
- F_{in} is the feed-back of the LFSR
- PF_{in} is the parity of the feed-back input
- A_{out} is the result of the multiply and addition operation
- P_{out} is the predicted parity of the result

The parity prediction block is implemented using the equation (6). It must be noticed that some constrains in the implementation of the constant multiplier should be added (see [9],[11]), in order to avoid interference between different output when a fault occurs. These interferences are due to the sharing of intermediate results between different output and therefore can be avoided using networks with fan-out equal to one: considering the FPGA implementation of constant multiplier, this constrain is not a serious drawback.. In fact, each output bit is computed implementing a simple XOR network that requires a very limited number of LUTs: in particular, if we con

FPGA with a 4-inputs LUT, if the number of inputs is less or equal 4 the implementation requires only a LUT, if the number of inputs is up to 7 requires 2 LUT's, while the worst case of a network of 8 XOR's requires 3 LUT's. As an example of this consideration in table I are reported the overhead introduced for different constant g_i.

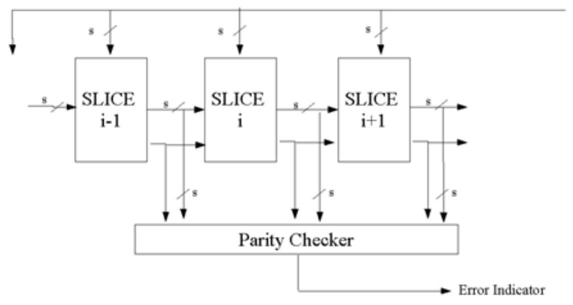


Fig. 8. Self-Checking RS encoder

The predicted parity bit and the output of each slice are evaluated by the parity checker block as shown in fig. 8, and an error indicator signal informs if a difference between the predicted parity bit and the parity of the slices outputs is detected. The parity checker block signals if the parity of the inputs is even or odd. An even (odd) parity checker corresponds to a even (odd) number $n-k$ of inputs. The self checking implementation of the parity checker can be realized with a two rail circuit with two outputs, each equal to the parity of one of the two disjoint subsets of the inputs [12]. In this way, the fault free behaviour of the checker, with a correct set of inputs (i.e. no faults occur in the slices), is to provide the output codes 01 or 01 for an odd parity checker or the output codes 00 or 11 for an even parity checker. If the checker receive as input an erroneous codeword (i.e. a fault occurs in a slice) the checker provide the output codes 11 or 00 for a odd parity checker or the output codes 01 or 10 for a even parity checker. Also if a fault occurs in the checker, when the fault is activated, the outputs provided are 11 or 00 for an odd parity checker or the output codes 01 or 10 for an even parity checker. Therefore, the self-checking property of the checker can be obtained simply using two networks of XOR gates for computing the two disjoint sets of inputs.

VI. AREA AND DELAY OVERHEAD EVALUATIONS

In this section an evaluation of the area overhead of the proposed structure is presented. The evaluations has been carried out using a Xilinx Virtex II as the target technology and the design flow has been performed using the ISE Foundation framework. We used the Galois Field GF(2^8) with the polynomial $i(x) = x^8 + x^4 + x^3 + x^2 + 1$ and evaluate the area of the four constant multiplier used to compute the generator polynomial

$$g(x) = (x+1)(x+\alpha)(x+\alpha^2)(x+\alpha^3) =$$
$$= x^4 + g_3x^3 + g_2x^2 + g_1x + g_0$$

that realize a encoder with $n-k=4$. In table I we report the area occupation of each of the blocks described in Section IV. The adder is implemented using one LUT for each output, while the area of the constant multipliers and of the parity prediction block depends by the coefficient g_i. The row named 'additional logic' represents the logic introduced into the slice in order to predict the parity bit and the number of LUT's required to implement the parity checker depends by the number of slices of the encoder, $i.e.$ the number $n-k$ of check bits of the RS code. In particular, implementing the parity checker as a network of xor gates, and using a LUT to implement a 4 input xor, the number of LUTs of the checker can be evaluated as $\lceil \frac{(n-k)(s+1)}{3} \rceil$, where s is the number of bits of a symbol.

With the result of table I the area overhead for the given can be estimated as the ratio between the implementation of the four slices without constrains and the four slices without sharing plus the additional logic and the parity checker. The overhead for the given case is exactly 50 %, and it is independent from the number of check symbols $(n-k)$. In fact, for each check symbol of 8 bits (s=8) we have an overhead of the single slice of about 6 LUTs, plus

	# LUT	# LUT without sharing	# FF
adder	8	-	0
g0_mult	8	8	0
g1_mult	13	16	0
g2_mult	9	10	0
g3_mult	11	14	0
slice*	18	20	8
additional* Logic	4	-	1
Parity Checker	12	-	0

TABLE I

AREA OF THE BUILDING BLOCKS

*mean value

an overhead due to the Parity checker of 3 LUTs. The equation estimating the overhead between the self-checking implementation and the one without self-checking capabilities is:

$$\frac{(n-k)*(6+3)}{(n-k)*18} = 50\%$$

The maximum frequency of the Reed Solomon encoders depends on the critical path from the output of the last slice to the register of another slice. The characterization of the critical time is different for each slice, depending on the complexity of the constant multiplier g_i. We can estimate this time evaluating the numbers of LUTs composing the path from the outputs of the last slice to the registers in the worst case. For the implementation of the constant multiplier g_i realized with a network of 8 xor the number of LUT's is 3, therefore the worst path can be estimated as the path going through 5 LUT's. In the self-checking implementation of the encoder two consideration can be done to estimate the delay overhead introduced. First of all the path of the parity prediction block, that provide the parity of the vector composed by the copies of the even computed bits and of the feed-back inputs, is comparable with the path of the worst-case constant multiplier. Moreover, an another path can be estimated, which is the path from the A_{out} and P_{out} registers storing the result and its predicted parity, to the register latching the error indicator signals provided by the error indicator block. This block is realized as a two-rail parity checker, therefore the number of the LUTs in the critical path can be estimated from the number of bits provided as input to the checker. In fact, the number of LUTs is the number of levels of the network composed by four inputs xor realized, that is $\lceil log_4(n-k)(s+1)\rceil$. Therefore, the worst path to be used to the delay overhead estimation is the maximum between the worst path of the standard RS encoder and the path of the two-rail parity checker. It must be noticed that the number of levels of the two-rail parity checker increases very slowly with the grow of the number of check symbols, and therefore actually do not represents a problem for the maximum frequency obtainable for the self-checking implementation of the encoder.

VII. CONCLUSIONS

In this paper an innovative self-checking Reed Solomon encoder architecture is described. The parity properties of the binary representation of the elements of the fields GF(2^n) are studied and a method for a self checking implementation of the arithmetic structures used in the Reed Solomon encoder has been proposed. The method has been applied to obtain an FPGA implementation of a self-checking encoder, the self-checking property has been analyzed using a SEU fault model and both area and delay overhead of the proposed solution has been evaluated.

REFERENCES

[1] A.H. Johnston, Radiation effects in advanced microelectronics technologies, IEEE Trans. Nucl. Sci., Vol. 43, no. 3, pp. 1339-1354, June 1998.
[2] E. Normand, Single Event Upset at Ground Level, IEEE Trans. Nucl. Sci., 43, 2742, (1996)
[3] E. Normand, Single Event Effects in Avionics, IEEE Trans. Nucl. Sci., 43, 461, (1996)

[4] G.C. Cardarilli, A. Leandri, P. Marinucci, M. Ottavi, S. Pontarelli, M. Re, A. Salsano,Design of a fault tolerant solid state mass memory, Reliability, IEEE Transactions on Volume 52, Issue 4, Dec. 2003 Page(s):476 - 491

[5] M. Violante, M. Ceschia, M. Sonza Reorda, A. Paccagnella, P. Bernardi, M. Rebaudengo, D. Bortolato, M. Bellato, P. Zambolin, A. Candelori, Analyzing SEU Effects in SRAM-based FPGAs, On-Line Testing Symposium, 2003. IOLTS 2003, 7-9 July 2003 pp. 119 - 123

[6] Reddy, E.S.S.; Chandrasekhar, V.; Sashikanth, M.; Kamakoti, V.; Vijaykrishnan, N.; Detecting SEU-caused routing errors in SRAM-based FPGAs, VLSI Design, 2005. 18th International Conference on 3-7 Jan. 2005 pp. 736 - 741

[7] H. Wu, Bit-parallel finite field multiplier and squarer using polynomial basis, Computers, IEEE Transactions on , Vol. 51 , no. 7 , pp. 750 - 758, July 2002

[8] A. Reyhani-Masoleh, M.A. Hasan, Low complexity bit parallel architectures for polynomial basis multiplication over GF(2m), Computers, IEEE Transactions on , Vol. 53 , no 8, pp. 945 - 959, Aug. 2004

[9] C. Bolchini, F. Salice, D. Sciuto, A novel methodology for designing TSC networks based on the parity bit code, European Design and Test Conference, 1997. ED&TC 97. Proceedings , pp. 440 - 444, 17-20 March 1997

[10] R.E. Blahut, Theory and Practice of Error Control Codes, Addison-Wesley Publishing Company, 1983

[11] N. A. Touba, E. J. McCluskey, Logic Synthesis Techniques for Reduced Area Implementation of Multilevel Circuits with Concurrent Error Detection, Proc. of Int. Conf. on Computer Aided Design (ICCAD), pp. 65l-654, 1994.

[12] D. Nikolos, Design Techniques for testable embedded error checkers, Computers, Vol. 23, Issue 7, pp. 84 - 88, July 1990