Design of a Self Checking Reed Solomon Encoder

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

***Abstract*— In this paper, an innovative self-checking Reed Solomon encoder architecture is described. The presented architecture exploits some properties of the arithmetic operations in GF(2^s) related to the parity of the binary representation of the field elements. Moreover, a method for introducing self-checking capabilities on all the arithmetic structures used in the Reed Solomon encoder is presented. Finally the self-checking encoder architecture has been mapped on a FPGA evaluating its area overhead.**

I. Summary and Conclusions

High reliability data transmission and storage systems frequently make use of Error Correction Codes (ECC) to protect the data. The coder and decoder blocks are critical for the design of high reliable system, in fact any error in this circuits may introduce catastrophic effects on the overall system. A fault in the encoder can produce an uncorrect codeword, while a fault in the decoder circuit 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 system. Therefore great attention must be paid in order to detect and recover faults in encoding and decoding circuitry. Nowadays, one of the most used error correcting code class is the Reed-Solomon one. It is based on the properties of the finite field arithmetic. In this paper, the property of finite fields arithmetic will be exploited to detect faults occurring in the coders, achieving the self-checking property in the arithmetic structures used in the design of the Reed Solomon encoder. The coder has been implemented on a Xilinx FPGA and the area overhead has been evaluated showing that the overhead is about 50% independently from the number of check symbols used in the code.

The paper is organized as follows: Section II describes the properties of the arithmetic structures used in the Reed Solomon encoders with respect to the parity of the arithmetic operands. In Section III the architecture of the proposed self-checking Reed Solomon encoder is presented while in Section IV some area overhead evaluations are provided.

II. Parity of arithmetic structures in GF(2^s)

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 operations will be analyzed:

- Parity of adders in GF(2^s)
- Parity of constant multipliers in GF(2^s)

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^s) the addition operation can be implemented simply xoring the bits of the same index, therefore the following property can be easily demonstrated:

$$P(a(x)+b(x)) = P(a(x)) \oplus P(b(x)) \tag{1}$$

For multipliers in GF(2^s) we focus our attention only on the multiplication by 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 coefficients of the irreducible generator polynomial $g(x)$. In this case the multiplier can be implemented by using a suitable network of XOR gates. As an example Fig. 1 shows the network implementing the two LSb of a multiplication by constant g equal to 1010 over GF(2^4)

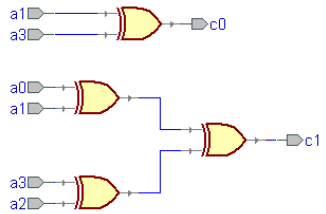

Fig. 1. GF(2^4) multiplier for g=1010

Therefore, each bit of the result of the constant multiplier can be computed by xoring some input bits depending from the constant g_i and from the chosen $g(x)$ irreducible polynomial.

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:

$$P(c) = \bigoplus_{i \in A} a_i \tag{2}$$

where A is the set of inputs that are evaluated an odd number of times. In this paper 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 can be explained using the concept of *odd observability* proposed in [3]. In this way, the parity of the output word o is:

$$P(o) = P(c) \oplus P(copy) = P(a) \tag{3}$$

Proceedings of the 11th IEEE International On-Line Testing Symposium (IOLTS'05)
1530-1591/05 $20.00 © 2005 **IEEE**

III. The Self Checking Reed Solomon Encoder

The implementation of a Reed Solomon encoder is usually realized through an LFSR, which implements the polynomial division over the finite field [4]. The RS encoder is composed by several slice blocks composed by a constant multiplier, an adder and a register. 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. 2.

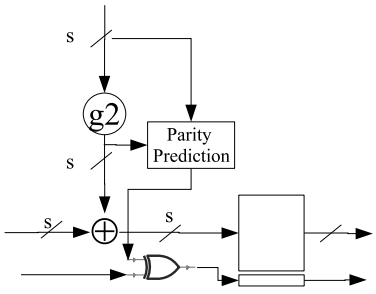

Fig. 2. Self-Checking Slice

The parity prediction block is implemented by using equation (3). It must be noticed that some constraints in the implementation of the constant multiplier should be added (see [3],[5]), in order to avoid interferences between different outputs when a fault occurs. These interferences are due to the share of intermediate results between different outputs of the constant multiplier and can be avoided using networks with fan-out equal to one. This constraint is not a serious drawback in the FPGA implementation of constant multiplier, since each output bit is computed by implementing a XOR network that requires a very limited number of LUTs: depending on the number N of inputs, the implementation of the constant multiplier requires in the worst case 3 LUTs for 8 XORs, one LUT if N less or equal 4 and 2 LUTs if N up to 7. As an example of this considerations in Table I are reported the overhead factors introduced for different constant coefficients.

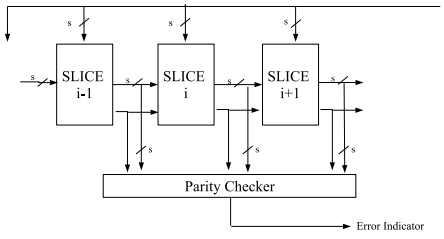

Fig. 3. Self-Checking RS encoder

The predicted parity bit and the output of each slice are evaluated by the parity checker block as shown in Fig. 3, and an error indicator signal informs if a difference between the predicted parity bit and the parity of the slice outputs is detected.

IV. Area overhead evaluations

The area overhead of the proposed encoder architecture has been carried out by using a Xilinx Virtex II as the target device. We used the Galois Field GF(2^8) with the polynomial $i(x) = x^8+x^4+x^3+x^2+1$ and the evaluation of the area of the four constant multipliers 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 realizes a encoder with $n-k=4$ has been carried out.

In table I the area used for each of the blocks described in Section III is shown in terms of number of used LUTs. The adder is implemented by using one LUT for each output, while the area of the constant multiplier and of the parity prediction block depends by the coefficient g_i. The parity checker is implemented 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.

	Normal (# of LUTs)	Self-Checking (# of LUTs)	# FF
adder	8	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
Parity Checker	-	12	0

TABLE I

Area used for the building blocks

*mean value

By using the results in Table I the area overhead for the given encoder can be estimated as the ratio between the implementation of the four slices without constraints 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, the equation estimating the overhead between the self-checking implementation and the ones without self-checking capabilities is:

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

References

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

[2] 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

[3] 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, March 1997

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

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