Transactions Briefs

Concurrent Error Detection in Reed–Solomon Encoders and Decoders

G. C. Cardarilli, S. Pontarelli, M. Re, and A. Salsano

***Abstract*—Reed–Solomon (RS) codes are widely used to identify and correct errors in transmission and storage systems. When RS codes are used for high reliable systems, the designer should also take into account the occurrence of faults in the encoder and decoder subsystems. In this paper, self-checking RS encoder and decoder architectures are presented. The RS encoder architecture exploits some properties of the arithmetic operations in $\mathrm{GF}(2^m)$. These properties are related to the parity of the binary representation of the elements of the Galois Field. In the RS decoder, the implicit redundancy of the received codeword, under suitable assumptions explained in this paper, allows implementing concurrent error detection schemes useful for a wide range of different decoding algorithms with no intervention on the decoder architecture. Moreover, performances in terms of area and delay overhead for the proposed circuits are presented.**

***Index Terms*—Error correction coding, fault tolerance, Reed–Solomon codes.**

I. Introduction

High reliable data transmission and storage systems frequently use error correction codes (ECC) to protect data. By adding a certain grade of redundancy these codes are able to detect and correct errors in the coded information. In the design of high reliable electronics systems both the Reed-Solomon (RS) encoder and decoder should be self checking in order to avoid faults in these blocks which compromise the reliability of the whole system. In fact, a fault in the encoder can produce a noncorrect codeword, while a fault in the decoder can give a wrong data word even if no errors occur in the codeword transmission. Therefore, great attention must be paid to detect and recover faults in the encoding and decoding circuitry. Nowadays, the most used error correcting codes are the RS codes, based on the properties of the finite field arithmetic. In particular, finite fields with 2^m elements are suitable for digital implementations due to the isomorphism between the addition, performed modulo 2, and the XOR operation between the bits representing the elements of the field.

The use of the XOR operation in addition and multiplication allows to use parity check-based strategies to check the presence of faults in the RS encoder, while the implicit redundancy in the codeword is used either for correct erroneous data and for detect faults inside the decoder block.

This paper is organized as follows. In Section II, a short background on RS codes is given, while Section III summarizes the main results of past works on this topic and describes the proposed methodology. In Section IV, the architecture of the proposed self-checking RS encoder is presented while Section V shows concurrent error detection (CED) schemes for the RS decoder. Conclusions are drawn in Section VI.

Manuscript received June 8, 2006.

The authors are with the Department of Electronic Engineering, University of Rome "Tor Vergata", 00133 Rome, Italy (e-mail: pontarelli@ing.uniroma2.it; salsano@ing.uniroma2.it; marco.re@ieee.org; g.cardarilli@ieee.org).

Digital Object Identifier 10.1109/TVLSI.2007.899241

II. RS Codes Background

In this section, a short background on RS codes is outlined. In [1], more information about finite fields and RS codes are provided. The finite fields used in digital implementations are in the form $\mathrm{GF}(2^m)$, where m represents the number of bits of a symbol to be coded. An element $a(x) \in \mathrm{GF}(2^m)$ is a polynomial with coefficients $a_i \in \{0,1\}$ and can be seen as a symbol of m bits $a = a_{m-1}, \ldots, a_1 a_0$. The addition of two elements $a(x)$ and $b(x) \in \mathrm{GF}(2^m)$ is the sum modulo 2 of the coefficients a_i and b_i, i.e., is the bitwise XOR of the two symbols a and b. The multiplication of two elements $a(x)$ and $b(x) \in \mathrm{GF}(2^m)$ requires the multiplication of the two polynomials followed by the reduction modulo $i(x)$, where $i(x)$ is an irreducible polynomial of degree m. Multiplication can be implemented as an AND-XOR network, as explained in [2].

The $\mathrm{RS}(n,k)$ code is defined by representing the data word symbols as elements of the field $\mathrm{GF}(2^m)$ and the overall data word is treated as a polynomial $d(x)$ of degree $k-1$ with coefficient in $\mathrm{GF}(2^m)$. The RS codeword is then generated by using the generator polynomial $g(x)$. All valid codewords are exactly divisible by $g(x)$. The general form of $g(x)$ is

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

where $2t = n-k$ and α is a primitive element of the field, i.e., $\forall \beta \in \mathrm{GF}(2^m) - \{0\} \exists i \in \mathbb{N} | \alpha^i = \beta$.

The codewords of a separable $\mathrm{RS}(n,k)$ code correspond to the polynomial $c(x)$ with degree $n-1$ that can be generated by using the following formulas:

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

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

where $p(x)$ is a polynomial with degree less than $n-k$ representing the parity symbols. In practice, the encoder takes k data symbols and adds $2t$ parity symbols obtaining a n symbol codeword. The $2t$ parity symbols allows the correction of up to t symbols containing errors in a codeword.

Defining the Hamming distance of two polynomials $a(x)$ and $b(x)$ of degree n as the number of coefficients of the same degree that are different, i.e., $H(a(x),b(x)) = \#\{i \le n | a_i \neq b_i\}$, and the Hamming weight $W(a(x))$ as the number of non-zero coefficients of $a(x)$, i.e., $W(a(x)) = \#\{i \le n | a_i \neq 0\}$ it is easy to prove that $H(a(x),b(x)) = W(a(x)-b(x))$. In a $\mathrm{RS}(n,k)$ code the Hamming distance between two codewords is $n-k$. After the transmission of the coded data on a noisy channel the decoder receives as input a polynomial $\overline{c}(x) = c(x)+e(x)$, where $e(x)$ is the error polynomial. The RS decoder identifies the position and magnitude of up to t errors and it is able to correct them. In other words the decoder is able to identify the $e(x)$ polynomial if the Hamming weight $W(e(x))$ is not greater than t. The decoding algorithm provides as output the codeword that is the only codeword having an Hamming distance not greater than t from the received polynomial $\overline{c}(x)$.

III. Methodology and Previous Work

In this section, the motivations of the design methodology used for the proposed implementations are described starting from an overview of the presented literature.

In [3], a radiation-tolerant RS encoder hardened against space radiation effects through circuit and layout techniques is presented. In [4] and [5], single and multiple parity bits schemes are presented to check the correctness of addition and multiplication in polynomial basis representation of finite fields.

In [6] and [7], the authors extend the techniques presented in [4] and [5] to detect faults occurring in the RS encoder, achieving the self-checking property for the RS encoder implementation. Moreover, in [8] and [9], a method to obtain CED circuits for finite field multipliers and inverters has been proposed.

Since both the RS encoder and decoder are based on $\mathrm{GF}(2^m)$ addition, multiplication, and inversion, their self-checking implementation can be obtained by using CED implementations of these basic arithmetic operations. Moreover, in [10], a self-checking algorithm for solving the key equation (that is a part of the overall decoding algorithm) has been introduced. Exploiting the algorithm proposed in [10] and substituting the elementary operations with the corresponding CED implementation for the other parts of the decoding algorithm a self-checking decoder can been implemented. This approach can be used for the encoder, that use only addition and constant multiplication and is illustrated in the following subsection, but it is unusable for the decoder as described later in this paper and a specific technique will be explained in the successive section.

A. Characteristics of the RS Encoder

In order to design a self-checking RS encoder by using the multipliers proposed in [4], [5], [8], and [9], each fault inside these blocks should be correctly detected. This detection is not ensured for the entire set of stuck-at faults (neither for the SEU fault set) because no details on the logical net-list implementing the multipliers are given in those papers. In fact, the authors present an estimation of the probability of undetected faults different from zero. To overcome this limitation, obtaining a total fault coverage for the single stuck-at faults the solution proposed in [6] is used. First of all, the characteristics of the arithmetic operations in $\mathrm{GF}(2^m)$ used in the RS encoder are analyzed with respect to the parity of the binary representation of the operands. The following two operations are considered:

- Parity of the addition in $\mathrm{GF}(2^m)$;
- Parity of the constant multiplication in $\mathrm{GF}(2^m)$.

Defining the parity $P(a(x))$ of a symbol as the XOR of the coefficients a_i, and taking into account that in $\mathrm{GF}(2^m)$ the addition operation is realized by the XOR of the bits having the same index, the following property can be easily demonstrated:

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

Taking into account that in the RS encoder the polynomial used to encode the data is constant, the polynomial multiplication is implemented by the multiplication for the constant g_i, where g_i are the coefficients of the generator polynomial $g(x)$. The constant multiplier is implemented by using an suitable network of XOR gates. This allows to exploit the concept of "odd-observability" proposed in [11]. The parity $P(c(x))$ 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. For the input bits evaluated an even number of times additional outputs are added.

B. Characteristics of the RS Decoder

The design of the self-checking decoder starting by the CED implementation of the arithmetic blocks and using the self-checking algorithm given in [10] for solving the key equation presents the following drawbacks.

1) The internal structure of the decoder must be modified substituting the elementary operations with the corresponding CED ones. Therefore, the decoder performances in terms of maximum operating frequency, area occupation, and power consumption can be very different with respect to the nonself-checking implementation.
2) The self-checking implementation is strongly dependent from the chosen decoder architecture (e.g., Berlekamp–Massey algorithm or the modified Euclidean algorithm [1]).
3) A good knowledge of the finite field arithmetic is essential for the implementation of $\mathrm{GF}(2^m)$ arithmetic blocks.

In the solution presented in this paper, differently from the previously discussed approaches, the implementation of the self-checking RS decoder is based on the use of a standard RS decoder (see IP vendors [12] and [13] for example) completed by adding suitable hardware blocks to check its functionality. In this way, the proposed method can be directly used for a wide range of different decoder algorithms enabling the use of important design concepts such as reusability.

The proposed technique starts from the following two main properties of the fault-free decoder.

Property 1: The decoder output is always a codeword.

Property 2: The Hamming weight of the error polynomial is not greater than t.

If a fault occurs inside the decoder the previously outlined observation is able to detect the occurrence of the fault. When the fault is activated, i.e., the output is different from the correct one due to the presence of the fault , the following two cases occur.

- The first case the decoder gives as output a noncodeword, and this case can be detected by property 1. This is the most probable case because the decoder computes the error polynomial and obtains the output codeword by calculating $c(x) = \overline{c}(x) + e(x)$, where $\overline{c}(x)$ is the received polynomial.
- If the output of the faulty decoder is a wrong codeword the detection of this fault is easily performed by evaluating the Hamming weight of the error polynomial $e(x)$.

The error polynomial can be provided by the encoder as an additional output or can be evaluated by comparing the received polynomial and the provided output $\overline{c}(x)$.

If one of the two properties is not respected a fault inside the decoder is detected, while if all the observations are satisfied we can detect that no faults are activated inside the decoder. This approach is completely independent by the assumed fault set and it is based only on the assumption that the fault-free behavior of the decoder provides always a codeword as output. This assumption is valid for a wide range of decoder architectures. For some decoders that are able to perform a miscorrection detection for some received polynomials with more than t errors suitable modification of our proposed method could be done.

IV. Self-Checking RS Encoder

The implementation of RS encoders are usually based on an LFSR, which implements the polynomials division over the finite field [1].

In Fig. 1, the implementation of an RS encoder is shown. The additions and multiplications are performed on $\mathrm{GF}(2^m)$ and g_i are the coefficients of the generator polynomial $g(x)$. The RS encoder architecture is composed by slice blocks containing a constant multiplier, an adder, and a register (see shaded block in Fig. 1). The number of slices to implement for an $\mathrm{RS}(n,k)$ code is $n-k$. The self-checking implementation requires the insertion of some parity prediction blocks and a parity checker. The correctness of each slice is checked by using the architecture shown in Fig. 2.

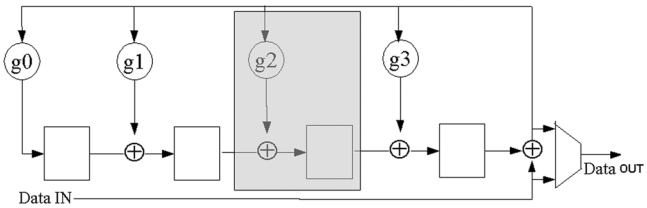

Fig. 1. RS encoder.

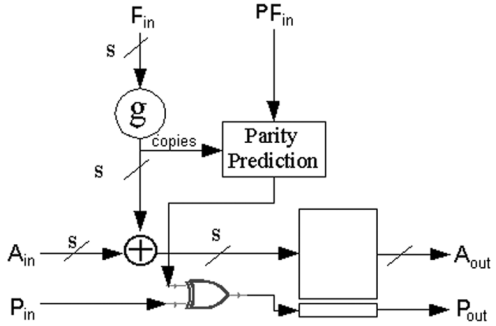

Fig. 2. Self-checking slice.

TABLE I
AREA OF THE BUILDING BLOCKS

	# 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

(*) mean value

The input and output signals to the slice are as follows.

- A_{in} is the registered output of the previous slice.
- P_{in} is 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 multiplication and addition operation.
- P_{out} is the predicted parity of the result.

The parity prediction block is implemented by using (5). It must be noticed that some constrains in the implementation of the constant multiplier must be added (see [11]) in order to avoid interference between different outputs when a fault occurs. These interferences are due to the sharing of intermediate results between different outputs and, therefore, can be avoided by using networks with fan-out equal to one: considering the field-programmable gate array (FPGA) implementation of constant multiplier, this constrain is not a serious drawback. In fact, each output bit is computed by implementing a XOR network requiring a very limited number of LUTs: for example, considering the field $\text{GF}(2^8)$ and an FPGA based on four-inputs LUTs, three LUT's in the worst case are required. Table I reports the overhead introduced for different constant g_i without resource sharing in the case of $\text{GF}(2^8)$.

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 informs if a difference between the predicted parity bit and the parity of the m slice outputs is detected.

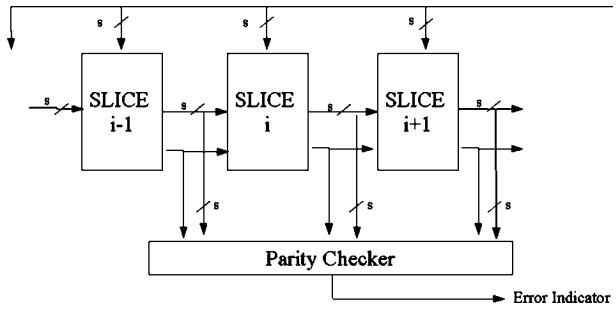

Fig. 3. Self-checking RS encoder.

The parity checker block checks if the parity of the inputs is even or odd.

The self checking implementation of the parity checker is realized with a two-rail circuit. The two outputs, are each equal to the parity of one of the two disjoint subsets of the inputs, as proposed in [14].

The fault-free behavior of the checker, when a correct set of inputs is provided (i.e., no faults occur in the slices) is the following: the output codes 01 or 10 are generated 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 provides the output codes 11 or 00 for an odd parity checker or the output codes 01 or 10 for an even parity checker.

Also, if a fault occurs in the checker the outputs provided are 11 or 00 for an odd parity checker or the output codes 01 or 10 for an even parity checker.

This considerations guarantee the self-checking property of the checker. It can be noticed that, due to the LFSR-based structure of the RS encoder, there are no control state machines to be protected against faults. Therefore, the use of the described self-checking arithmetic structures allows to check the entire RS encoder. The evaluations in terms of area and delay of this structure has been carried out by using a Xilinx Virtex II FPGA as the target device and the design flow has been performed by using the Xilinx ISE foundation framework.

Table I reports the area of each of the blocks described in this section. The adder is implemented by using one LUT for each output, while the area of the constant multipliers and of the parity prediction block depends by the coefficients g_i.

In Table I, the row named "additional logic" represents the logic added to the slice in order to predict the parity bit. The number of LUTs 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, the number of LUTs is $\lceil((n-k)(m+1))/(3)\rceil$.

Starting from the result shown in Table I, the area overhead has been computed for the given case. The overhead 50%, and it is independent from the number of check symbols $(n-k)$. In fact, for each check symbol $(m=8)$ the overhead for the single slice is about six LUTs, plus the overhead due to the parity checker (three LUTs). The equation describing the overhead is

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

The characterization of the critical path is different for each slice, depending on the complexity of the constant multiplier g_i. In the worst case, the constant multiplier g_i implemented by using an eight XOR network requires three LUTs, therefore, in the worst case path five LUTs are crossed.

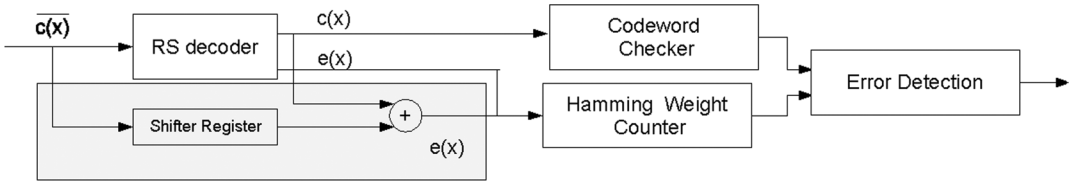

Fig. 4. CED scheme of the RS decoder.

In order to compute the critical path for the overall self checking encoder architecture, the following additional signal paths must be considered:

- path crossing the parity prediction block that is comparable with the path of the worst-case constant multiplier;
- path crossing the parity checker. This path depends by the number of bits provided as input to the checker. In fact, the number of required LUTs is equal to the number of levels of the four inputs XOR network, that is $\lceil \log_4(n-k)(m+1) \rceil$.

The number of levels of the two-rail parity checker increases very slowly with the growth of the number of check symbols, and therefore, do not represent a problem for the maximum frequency of the self-checking decoder.

V. Concurrent Error Detection Scheme of The RS Decoder

In Fig. 4, the CED implementation of the RS decoder is shown. Its main blocks are as follows.

- RS decoder, i.e., the block to be checked.
- An optional error polynomial recovery block (the shaded block shown in Fig. 4). This block is needed if the RS decoder does not provide at the output the error polynomial coefficients.
- Hamming weight counter, that checks the number of nonzero coefficients of the error polynomial.
- Codeword checker, that checks if the output data of the RS decoder form a correct codeword.
- Error detection block that take as inputs the output of the Hamming weight counter and of the codeword checker and provides an error detection signal if a fault in the RS decoder has been detected.

The RS decoder can be considered as a black box performing an algorithm for the error detection and correction of the input data (the coefficients of the received data forming the polynomial $\overline{c}(x)$.

The error polynomial recovery block is composed by a shifter register of length L (the latency of the decoder) and by a $\mathrm{GF}(2^m)$ adder having as operands the coefficients of $c(x)$ and $\overline{c}(x)$.

The Hamming weight counter is composed by the following:

1) a comparator indicating (at each clock cycle) if the $e(x)$ coefficients are zero;
2) a counter that takes into account the number of nonzero coefficients;
3) a comparator between the counter output and t that is the maximum allowed number of nonzero elements.

The codeword checker block checks if the reconstructed $c(x)$ is a codeword, i.e., if it is exactly divisible for the generator polynomial $g(x)$. The following two implementations of this block are proposed.

Implementation 1: It is based on the computation of the remainder of the polynomial division between $c(x)$ and $g(x)$. If all the coefficients of the remainder polynomial are zero then the polynomial $c(x)$ is a correct codeword. The remainder of the division by $g(x)$ is exactly the function of the systematic RS encoder. Therefore, a systematic RS encoder with the same $g(x)$ polynomial of the decoder is used if $c(x)$ is a codeword. Faults in the decoder can be detected ignoring either $g(x)$ and also ignoring how the operation in $\mathrm{GF}(2^m)$ is performed. We only need to reuse the same RS encoder used to create the codeword for the computation of the remainder $c(x)$ obtained from the decoder. The drawback of this implementation is the additional latency introduced by the RS encoder, that is $n-k$ clock cycles. This latency must be considered by the error detection block that must wait $n-k$ clock cycles to check the two properties defined in Section III. The area occupation of the RS encoder is smaller than the area occupation of the decoder (see, e.g., [12] and [13]), therefore, the overhead introduced by this block is about 15% of the decoder area.

Implementation 2: The codeword checker block is based on the so-called syndrome calculation. This operation is the first to be performed in the decoder, therefore, conceptually this approach implies a partial duplication of the RS decoder and implies the knowledge of the used Galois field and the roots of $g(x)$. The syndrome calculation imply the evaluation of the received polynomial $\overline{c}(x)$ for the values of x in the set A, with $A = \{\alpha^{i+j} | 0 \leq j \leq 2t\}$, i.e., A is the set of the roots of $g(x)$. The received polynomial $\overline{c}(x)$ is exactly divisible for $g(x)$ if and only if it is exactly divisible for all the monomials $(x-\alpha^{i+j})$, where α is a root of $g(x)$. The polynomial is divisible by $(x-\alpha^{i+j})$ if $\overline{c}(\alpha^{i+j})$ is zero. Therefore, the received polynomial is a codeword if and only if all the computed syndromes are zero. The syndromes computation block is composed by a $\mathrm{GF}(2^m)$ constant multiplier, an adder and an m-bit register. The output of this block is valid one clock cycle later than the computation of the last coefficient of the polynomial. The area occupation of the syndrome calculation block is equivalent to the encoder area occupation. In fact, in both cases we need $n-k$ blocks composed by an adder, a constant multiplier and an m-bit register.

The main difference between implementation 1 and 2 is the latency of the codeword checker block. The error detection block takes as inputs the outputs of the Hamming weight counter and the outputs of the codeword checker. Its implementation depends from the chosen implementation of the codeword checker. If we use implementation 1 the error detection block must delay the output of the Hamming weight counter for $n-k$ clock cycles and checks if all the coefficients of the remainder polynomial are zero. On the other hand, if we use the syndromes calculation block the inputs are the computed syndromes and the error detection block checks if all the received symbols are zero. The additional blocks used to detect faults inside the decoder are susceptible to faults and, therefore, their implementation must assure the self-checking property, in order to face the age old question of "who checks the checker." For the codeword checker and the error polinomiyal generator blocks only register and $\mathrm{GF}(2^m)$ addition and constant multiplication are used and, therefore, the same consideration of Section IV can be used to obtain the self-checking property of these blocks. For the counters and the comparator used in the Hamming weight counter and error detection blocks, many efficient techniques can be found in literature (see, e.g., [15]).

VI. Conclusion

In this paper self-checking architectures for an RS encoder and decoder are described. The parity properties of the binary representation of the elements of $\mathrm{GF}(2^m)$ has been studied and a method for a self-checking implementation of the arithmetic structures used in the

RS encoder has been proposed. The problems related to the presence of undetected faults in parity check-based schemes has been faced by imposing some constrains in the logical net-list implementation for the constant multiplier. Evaluations of area and delay overhead for the self-checking RS encoder has been provided. For the self-checking RS decoder two main properties of the fault free decoder have been identified and used to detect faults inside the decoder. The proposed method can be used for a wide range of algorithm implementing the decoder function. Some concurrent error detection schemes have been explained in the paper and some evaluations of area overhead has been provided. Our method is nonintrusive, i.e., the decoder architecture is not modified. This fact enables the use of the reusability concept, for the design of very complex digital systems.

REFERENCES

[1] R. E. Blahut, *Theory and Practice of Error Control Codes*. Reading, MA: Addison-Wesley Publishing Company, 1983.

[2] A. R. Masoleh and M. A. Hasan, "Low complexity bit parallel architectures for polynomial basis multiplication over GF(2m), computers," *IEEE Trans. Comput.*, vol. 53, no. 8, pp. 945–959, Aug. 2004.

[3] J. Gambles, L. Miles, J. Has, W. Smith, and S. Whitaker, "An ultra-low-power, radiation-tolerant reed solomon encoder for space applications," in *Proc. IEEE Custom Integr. Circuits Conf.*, 2003, pp. 631–634.

[4] A. R. Masoleh and M. A. Hasan, "Error Detection in Polynomial Basis Multipliers over Binary Extension Fields," in *Lecture Notes in Computer Science*. New York: Springer-Verlag, 2003, vol. 2523, pp. 515–528.

[5] S. B. Sarmadi and M. A. Hasan, "Concurrent error detection of polynomial basis multiplication over extension fields using a multiple-bit parity scheme," in *Proc. IEEE Int. Symp. Defect Fault Tolerance VLSI Syst.*, 2005, pp. 102–110.

[6] G. C. Cardarilli, S. Pontarelli, M. Re, and A. Salsano, "Design of a self checking reed solomon encoder," in *Proc. 11th IEEE Int. On-Line Test. Symp. (IOLTS'05)*, 2005, pp. 201–202.

[7] G. C. Cardarilli, S. Pontarelli, M. Re, and A. Salsano, "A self checking Reed Solomon encoder: Design and analysis," in *Proc. IEEE Int. Symp. Defect Fault Tolerance VLSI Syst.*, 2005, pp. 111–119.

[8] M. Gossel, S. Fenn, and D. Taylor, "On-line error detection for finite field multipliers," in *Proc. IEEE Int. Symp. Defect Fault Tolerance VLSI Syst.*, 1997, pp. 307–311.

[9] Y.-C. Chuang and C.-W. Wu, "On-line error detection schemes for a systolic finite-field inverter," in *Proc. 7th Asian Test Symp.*, 1998, pp. 301–305.

[10] I. M. Boyarinov, "Self-checking algorithm of solving the key equation," in *Proc. IEEE Int. Symp. Inf. Theory*, 1998, p. 292.

[11] C. Bolchini, F. Salice, and D. Sciuto, "A novel methodology for designing TSC networks based on the parity bit code," in *Proc. Eur. Design Test Conf.*, 1997, pp. 440–444.

[12] Altera Corp., San Jose, CA, "Altera Reed-Solomon compiler user guide 3.3.3," 2006.

[13] Xilinx, San Jose, CA, "Xilinx logicore Reed-Solomon decoder v5.1," 2006.

[14] D. Nikolos, "Design techniques for testable embedded error checkers, computers," *Computer*, vol. 23, no. 7, pp. 84–88, Jul. 1990.

[15] P. K. Lala, *Fault Tolerant and Fault Testable Hardware Design*. Englewood Cliffs, NJ: Prentice-Hall, 1985.

A Low-Power Multiplier With the Spurious Power Suppression Technique

Kuan-Hung Chen and Yuan-Sun Chu

***Abstract*—This paper provides the experience of applying an advanced version of our former spurious power suppression technique (SPST) on multipliers for high-speed and low-power purposes. To filter out the useless switching power, there are two approaches, i.e., using registers and using AND gates, to assert the data signals of multipliers after the data transition. The SPST has been applied on both the modified Booth decoder and the compression tree of multipliers to enlarge the power reduction. The simulation results show that the SPST implementation with AND gates owns an extremely high flexibility on adjusting the data asserting time which not only facilitates the robustness of SPST but also leads to a 40% speed improvement. Adopting a 0.18-μm CMOS technology, the proposed SPST-equipped multiplier dissipates only 0.0121 mW per MHz in H.264 texture coding applications, and obtains a 40% power reduction.**

***Index Terms*—H.264, low-power, multiplier, spurious power suppression technique (SPST).**

I. INTRODUCTION

Lowering down the power consumption and enhancing the processing performance of the circuit designs are undoubtedly the two important design challenges of wireless multimedia and digital signal processor (DSP) applications, in which multiplications are frequently used for key computations, such as fast Fourier transform (FFT), discrete cosine transform (DCT), quantization, and filtering. To save significant power consumption of a VLSI design, it is a good direction to reduce its dynamic power that is the major part of total power dissipation.

The designs [1]–[7] are existing works that reduce the dynamic power consumption by minimizing the switched capacitance. The design [1] proposes a concept called *partially guarded computation* (PGC), which divides the arithmetic units, e.g., adders, and multipliers, into two parts, and turns off the unused part to minimize the power consumption. The reported results show that the PGC can reduce power consumption by 10% to 44% in an array multiplier with 30% to 36% area overheads in speech related applications. Design [2] proposes a 32-bit 2's complement adder equipping a master-stage flip-flop and a slave-stage flip-flop for both operands of the adder, a *dynamic-range determination* (DRD) unit, and a sign-extension unit. This design tends to reduce the power dissipation of conventional adders for multimedia applications. Additionally, design [3] presents a multiplier using the DRD unit to select the input operand with a smaller effective dynamic range to yield the Booth codes. The direct report of [3] shows that the multiplier can save over 30% power dissipation than conventional ones. Design [4] incorporates a technique for glitching power minimization by replacing some existing gates with functionally equivalent ones that can be "frozen" by asserting a control signal. This technique can be applied to replace layout-level descriptions and guarantees predictable results. However, it can only achieve savings of 6.3% in total power dissipation since it operates in

Manuscript received October 5, 2006; revised March 1, 2007. This work was supported in part by the National Science Council of the Republic of China, Taiwan, R.O.C., under Contract NSC 95-2221-E-194-093-MY2.

K.-H. Chen is with Feng-Chia University, Tai-Chung 40724, Taiwan, R.O.C. (e-mail: kuanhung@fcu.edu.tw).

Y.-S. Chu is with the National Chung-Cheng University, Chia-yi 62102, Taiwan, R.O.C. (e-mail: chu@ee.ccu.edu.tw).

Digital Object Identifier 10.1109/TVLSI.2007.899242