Scalable Error Detection Coding© Algorithm for Totally Self-Checking (TSC) Circuits

SEDC© Algorithm for TSC Circuits

Natarajan Somasundaram[*1], Jeong A Lee[2], Farhad Mehdipour[3], Ramadass Narayanadass[4], Y V Ramana Rao[5]

[*1]Electronics and Communication Engineering, SSM College of Engineering, Tamilnadu, India
[2]Computer System Laboratory, Chosun University, Gwangju, South Korea
[3]E-JUST Center, Information Science and Electrical Engineering, Kyushu University, Fukuoka, Japan
[4, 5]Electronics and Communication Engineering, College of Engineering, Anna University, Chennai, Tamilnadu, India
[*1]somasundaramnatarajan@yahoo.com; [2]jalee@chosun.ac.kr; [3]farhad@ejust.kyushu-u.ac.jp; [4]ramadassn@annauniv.edu; [5]yvramana@annauniv.edu

***Abstract*- With continued scaling of silicon process technology, producing reliable electronic components in extremely denser technologies pose a challenge. Further, the systems fabricated in deep sub-micron technology are prone to intermittent or transient faults, causing unidirectional errors, upon exposure to ionizing radiations during system operation. The ability to operate in the intended manner even in the presence of faults is an important objective of all electronic systems. In order to achieve fault-tolerance, each module of the system must be fault-tolerant by possessing run-time (or online) fault detection capabilities. Totally Self-checking (TSC) circuits permit online detection of hardware faults. The Scalable Error Detection Coding (SEDC) algorithm used to design self-checking circuits with faster execution and lesser latency overhead for use in fault-tolerant reconfigurable architecture is presented. SEDC algorithm is formulated and architecture is designed in such a way that for any input binary data length, only area is scaled, with a constant latency of 2 logic gates and requires only a single clock cycle for generating SEDC code. It is shown that the proposed SEDC algorithm is found to be significantly efficient than the existing unidirectional error detection techniques in terms of speed, latency, area and achieving 100% error detection.**

Keywords- Fault Tolerance; Totally Self-Checking Circuits; Dependable Architecture; Error Detection Coding; Unidirectional Errors

I. INTRODUCTION

One of the major driving forces of the semiconductor industry is the continuous scaling of the silicon process technology. Improvements in lithographic and related VLSI manufacturing techniques have made Moore's law stand true. Over the last four decades, scaling of silicon technology offered smaller, faster, cheaper, denser and high performance devices. With continued scaling of silicon process technology, producing reliable electronic components in extremely denser technologies pose a challenge with a warning that it will result in devices that are much less reliable than the current devices [1]. Future technology devices will likely experience failures due to silicon defects occurring during system operation. When a metal-oxide-semiconductor (MOS) transistor is exposed to high-energy ionizing irradiation, electron-hole pairs are created in the transistor that may invert the logic state of the transistor [2]. The interaction of neutron and alpha particles with semiconductor devices may lead to unintended difference between implemented hardware and its intended design, resulting in an error. This difference is called as a fault which may be permanent, intermittent, or transient. Permanent faults can be due to fault in semiconductor material, manufacturing process, or age defect (electromigration). Intermittent faults can be due to design parameter error, timing problem, etc. Transient faults can be caused by external radiation or by electrostatic discharge. In case of permanent faults, the circuit is damaged permanently and cannot be repaired. On the other hand, intermittent or transient faults momentarily generate false outputs at random times, making it hard to detect. Thus, error detection becomes a greater concern for system reliability as transistor size decreases.

The errors occurring in an electronic circuit can be broadly classified as symmetric, asymmetric, and unidirectional errors [3]. The error is symmetric if both 0 to 1 and 1 to 0 transitions occur simultaneously in a data word. If only 0 to 1 or 1 to 0 transitions are likely, and the error type is known a priori, then the errors are asymmetric. If both 0 to 1 and 1 to 0 transitions can occur in data words, but in any particular word all errors are of one type, then the errors are called unidirectional errors. In Complementary Metal Oxide Semiconductor (CMOS) implementation, faults, which can be stuck-at-0 or stuck-at-1, appear only as unidirectional errors [4, 5, 6].

The ability to operate in the intended manner even in the presence of faults is an important objective of all electronic systems. In order to achieve fault-tolerance, each module of the system must be fault-tolerant by possessing concurrent fault detection capabilities. Totally Self-checking (TSC) circuits permit online detection of hardware faults. There are many error detection techniques but they either consume more hardware area or computational latency with increase in binary data length. This manuscript addresses this problem and introduces the new Scalable Error Detection Coding (SEDC) algorithm. SEDC algorithm is formulated and architecture is designed in such a way that only area is scaled, with latency and speed remaining constant with binary data length.

The rest of this manuscript is organized as follows. In Section 2, we review related works in concurrent unidirectional error detection. In Section 3, we describe the SEDC algorithm and architecture. In Sections 4 and 5, the proof of unidirectional error detection property and Totally Self-Checking (TSC) property of SEDC algorithm are discussed respectively. In Section 6, the architecture of proposed SEDC technique is briefed. The area comparison of SEDC code generator is provided in Section 7. Conclusions are provided in Section 8.

II. RELATED WORKS

Fault detection methods [7, 8] can be broadly classified as Built-In Self-test (BIST), roving technique, redundancy technique, logic implications technique and error coding technique. BIST is an offline fault detection technique [8] which is widely used and which does not involve any external test equipments. This technique is not suited for reconfigurable embedded systems which require online fault detection capability, even when the system is in operation. Roving fault detection [9] uses run-time reconfiguration to carryout online fault testing. In roving detection, the computational array is split into equal-sized regions. One of these regions is configured to perform testing operation, while the remaining areas perform the designed function. Over time, the test region is swapped with functional regions one at a time so that the entire array can be tested while the system remains functional [10]. Although this is an online testing method, testing speed is slow because of swapping process involved. Redundancy is based on either modular redundancy or time redundancy [11, 12]. In modular redundancy, the functional module is replicated two or three times. In time redundancy, the same function is performed by the same functional module more than once. Any difference in these outputs indicates a fault. It is obvious that there is an area overhead or latency overhead by two or three times when using redundancy techniques. Logic implications method [13] takes an existing design and searches all internal circuit nodes for consistent logic patterns. When an implicit circuit pattern is found, extra checker module is appended in the circuit for detecting the faults. The drawback with this technique is that it is strategy-based and not well suited for concurrent error detection. Error coding technique is more efficient that the other fault detection methods in terms of area, speed and fault coverage [14, 15]. This method involves coding the data using error coding algorithms. Faults can be detected by verifying the code with binary data.

Many unidirectional error detecting codes like Parity code, Hamming code, Reed Solomon code, Berger code and Bose code have been reported in the Literature [15]. The simplest and cheapest error detecting code is parity code [16] which appends only one error-check bit to the information bits. This error-check bit is computed in such a way the number of 1's in information bits along with parity bit is made odd or even. This technique requires very little hardware overhead and is fast to compute [17]. But it can detect only single errors or all odd number of errors in the information bits. Hamming code is the first error detection technique to provide error correction capability [18]. This technique involves performing parity coding on different bits to generate hamming check bits. Hamming circuits requires slightly additional hardware overhead than parity circuits which is a performance penalty required for error correction process. Similar to parity code, this technique also detects only single errors and double errors and not all unidirectional errors. Reed Solomon code is a polynomial based error detection code [19] providing error correction capability also. This technique requires more area and speed overhead when compared to Hamming code and cannot detect all unidirectional errors. Berger code can detect all multiple unidirectional errors but provides no error correction capability. The error-check code is generated either by B0 encoding scheme or B1 encoding scheme [5, 20]. In B0 encoding scheme, error-check code is generated by counting number of Logic 0 bits and representing the count as a binary number. In B1 encoding scheme, error-check code is generated by counting number of Logic 1 bits and representing the ones complement of the count as a binary number. Self-checking circuit [21] using Berger code can have Berger encoder implemented as a sequential circuit or as a combinational circuit. The sequential circuit implementation requires more resource overhead to implement counter circuits and takes multiple clock cycles to detect the error. The combinational circuit implementation takes more hardware latency. Incorporating Berger code into a delay-optimised circuit to make it self-checking thereby affects the system clock speed and timing constraints of the circuit, due to the dependency of Berger code on the binary data length. Bose code [22] is similar to Berger code in the sense that code is computed by counting number of Logic 0's or Logic 1's and performing modulo 4 or 8 to detect t-unidirectional errors, where t refers to double or triple unidirectional errors. Similar to Berger code, error correction capability is not available. There are similar codes available, such as Dong code [23] with reduced unidirectional error detection capability. A summary of all these error detecting codes is shown in Table 1.

TABLE I SUMMARY OF ERROR DETECTION CODES

Method	Resource Overhead	Performance Overhead	Error Detection	Number of Code bits (d-bit data, c-bit code)
Parity Code	Very small	Very small	Single errors or all odd number of errors	$c = 1$
Hamming Code	Large	Small	Single and Double errors	c is chosen as $2^c - c - 1 > d$
Reed Solomon Code	Very large	Large	Not all unidirectional errors	$c = 2 \times t$, where t is number of errors to be corrected
Berger Code	Very large	Very large	All multiple unidirectional errors	$c = \lceil \log_2(d + 1) \rceil$
Bose Code	Very large	Very large	Not all multiple unidirectional errors	c is chosen to detect $(5 \times 2^{c-4}) + c - 4$ errors

Dong Code	Very large	Very large	Not all multiple unidirectional errors	$c = \lceil \log_2(m + 1) \rceil$, where $m < d$

III. SCALABLE ERROR DETECTION CODING (SEDC) TECHNIQUE

A need arises to formulate an error detection algorithm and design the corresponding architecture in order to achieve 100% unidirectional error detection with minimum hardware overhead and without compromising performance in terms of speed and latency. These requirements motivate the need for development of new Scalable Error Detection Coding (SEDC) algorithm. SEDC algorithm is formulated and architecture is designed in such a way that only area is scaled, with latency and speed remaining constant with binary data length. SEDC algorithm is composed of techniques to generate SEDC codeword for 2-, 3-, 4- and n-bit data, where $n \geq 5$, called as $SEDC_2$ algorithm, $SEDC_3$ algorithm, $SEDC_4$ algorithm and $SEDC_n$ algorithm, respectively.

A. Number of SEDC Code Bits

For input binary data D of length n-bits represented as (D_{n-1},….., D_2, D_1, D_0), two parameters 'a' and 'b' are computed as per Eq. (1) where, parameter 'a' can take only integer values from 0 to infinity, and parameter 'b' can take values only from 2, 3 or 4.

$$a = \frac{n - \max(b)}{3} \tag{1}$$

Satisfying the condition for parameter 'a', the maximum possible value for parameter 'b' is selected. The length of SEDC Code C represented as (C_{m-1},….., C_2, C_1, C_0) is then computed as per Eq. (2).

$$m = \lceil \log_2[n + 1] - (3 \times a) \rceil + (2 \times a) \tag{2}$$

B. SEDC Code Bit Generation

SEDC algorithm for 2-bit data, called $SEDC_2$ algorithm, is formulated as per Eq. (3) and Eq. (4).

$$C_1 = \begin{cases} 1, \text{if number of 0's in D} \geq \text{number of 1's in D} \\ 0, \text{otherwise} \end{cases} \tag{3}$$

$$C_0 = \begin{cases} 0, \text{if number of 0's in D} = \text{number of 1's in D} \\ 1, \text{otherwise} \end{cases} \tag{4}$$

SEDC algorithm for 3-bit data, called $SEDC_3$ algorithm, is formulated as per Eq. (5).

$$(C_1, C_0) = \begin{cases} SEDC_2(D_1, D_0), & \text{if } D_2 = 0 \\ 1\text{'s complement}\left(SEDC_2\left(\overline{D_1}, \overline{D_0}\right)\right), & \text{if } D_2 = 1 \end{cases} \tag{5}$$

SEDC algorithm for 4-bit data, called $SEDC_4$ algorithm, is formulated as per Eq. (6) and Eq. (7)

$$(C_1, C_0) = SEDC_3(D_2, D_1, D_0) \tag{6}$$

$$C_2 = NOT(D_3) \tag{7}$$

SEDC algorithm for n-bit data, where $n \geq 5$, called $SEDC_n$ algorithm, is formulated by grouping n-bit binary data into one 'b'-bit segment and 'a' number of 3-bit segments, on which $SEDC_b$ and $SEDC_3$ algorithms are applied.

C. SEDC Architecture

$SEDC_2$ architecture requires a 2-input XNOR and 2-input NAND gate to implement C_0 and C_1 respectively. $SEDC_3$ architecture requires a 3-input XNOR gate, a 2-input AND gate and a 2-input Or-And-Invert gate to implement C_0 and C_1. $SEDC_4$ architecture requires a 2-input XNOR gate, a 2-input XOR gate, a 2 × 1 inverting MUX and an inverter to implement C_0, C_1 and C_2. In general, for any binary data length, the hardware latency remains a constant, equal to the latency of two logic gates. This is a unique feature of this scalable algorithm. The SEDC architecture does not affect the performance of delay-optimized circuit when the circuit is made self-checking.

IV. UNIDIRECTIONAL ERROR DETECTION PROPERTY OF SEDC CODE

Theorem: The SEDC code can detect all types of single and multiple unidirectional errors if and only if the encoded data bits, which is a combination of data bits appended with SEDC code bits, is unique. Also, the SEDC algorithm is scalable with binary data length. The following lemma proves this theorem.

Lemma: For scalability, the SEDC algorithm for 3-bit data must be a function of SEDC algorithm for 2-bit data, SEDC algorithm for 4-bit data must be a function of SEDC algorithm for 3-bit data, and so on. This scalability must be supported without affecting error detection performance.

***Proof*:** The SEDC algorithm for 2-bit data is formulated using following rules:

1. The possible 2-bit data are $(00)_2$, $(01)_2$, $(10)_2$ and $(11)_2$. In order to detect all unidirectional errors, $(00)_2$ must have a separate code, $(11)_2$ must have a separate code, and both $(01)_2$ and $(10)_2$ can have same codeword. This is a condition for having optimal number of codeword. So, optimal number of codeword is 3.

2. The optimal codeword length for 2-bit data is 2. So the possible 2-bit codeword are $(00)_2$, $(01)_2$, $(10)_2$ and $(11)_2$. Out of these 4 codeword, only 3 must be selected. The possible codeword combinations are {$(00)_2$, $(01)_2$ and $(10)_2$}, {$(00)_2$, $(01)_2$ and $(11)_2$}, {$(00)_2$, $(10)_2$ and $(11)_2$} and {$(01)_2$, $(10)_2$ and $(11)_2$}. These 3 codeword must be mapped to 4 binary data.

Case 1 for combination {$(00)_2$, $(01)_2$ and $(10)_2$}:

If codeword $(00)_2$ is assigned to data $(00)_2$, then we cannot detect all unidirectional errors in any of the 2-bit data. Hence, codeword $(00)_2$ cannot be assigned to data $(00)_2$. If codeword $(00)_2$ is assigned to data $(01)_2$ and $(10)_2$, then we cannot detect all unidirectional errors in any of the 2-bit data. Hence, codeword $(00)_2$ cannot be assigned to data $(01)_2$ and $(10)_2$. If codeword $(00)_2$ is assigned to data $(11)_2$, then we can assign codeword $(01)_2$ to data $(00)_2$ and codeword $(10)_2$ to both data $(01)_2$ and $(10)_2$, which is Case 1a. We can also assign codeword $(10)_2$ to data $(00)_2$ and codeword $(01)_2$ to both data $(01)_2$ and $(10)_2$, which is Case 1b. In Case 1a, we cannot detect all unidirectional errors for the 3-bit data after scaling the algorithm. In Case 1b, the codeword is not scalable for 3-bit data. Hence, codeword $(00)_2$ cannot be assigned to data $(11)_2$.

Case 2 for combination {$(00)_2$, $(01)_2$ and $(11)_2$}:

If codeword $(00)_2$ is assigned to data $(00)_2$, then we cannot detect all unidirectional errors in any of the 2-bit data. Hence, codeword $(00)_2$ cannot be assigned to data $(00)_2$. If codeword $(00)_2$ is assigned to data $(01)_2$ and $(10)_2$, then we cannot detect all unidirectional errors in any of the 2-bit data. Hence, codeword $(00)_2$ cannot be assigned to data $(01)_2$ and $(10)_2$. If codeword $(00)_2$ is assigned to data $(11)_2$, then we can assign codeword $(01)_2$ to data $(00)_2$ and codeword $(11)_2$ to both data $(01)_2$ and $(10)_2$, which is Case 2a. We can also assign codeword $(11)_2$ to data $(00)_2$ and codeword $(01)_2$ to both data $(01)_2$ and $(10)_2$, which is Case 2b. In Case 2a, we cannot detect all unidirectional errors for the 2-bit data. In Case 2b, we cannot detect all unidirectional errors for the 3-bit data after scaling the algorithm. Hence, codeword $(00)_2$ cannot be assigned to data $(11)_2$.

Case3 for combination {$(00)_2$, $(10)_2$ and $(11)_2$}:

If codeword $(00)_2$ is assigned to data $(00)_2$, then we cannot detect all unidirectional errors in any of the 2-bit data. Hence, codeword $(00)_2$ cannot be assigned to data $(00)_2$. If codeword $(00)_2$ is assigned to data $(01)_2$ and $(10)_2$, then we cannot detect all unidirectional errors in any of the 2-bit data. Hence, codeword $(00)_2$ cannot be assigned to data $(01)_2$ and $(10)_2$. If codeword $(00)_2$ is assigned to data $(11)_2$, then we can assign codeword $(10)_2$ to data $(00)_2$ and codeword $(11)_2$ to both data $(01)_2$ and $(10)_2$, which is Case 3a. We can also assign codeword $(11)_2$ to data $(00)_2$ and codeword $(10)_2$ to both data $(01)_2$ and $(10)_2$, which is Case 3b. In Case 3a, we cannot detect all unidirectional errors for the 2-bit data. In Case 3b, we cannot detect all unidirectional errors for the 3-bit data after scaling the algorithm. Hence, codeword $(00)_2$ cannot be assigned to data $(11)_2$.

3. If codeword $(11)_2$ is assigned to data $(01)_2$ and $(10)_2$, then we cannot detect all unidirectional errors in any of the 2-bit data. Hence, codeword $(11)_2$ cannot be assigned to data $(01)_2$ and $(10)_2$. For similar reasons, codeword $(11)_2$ cannot be assigned to data $(11)_2$ also.

4. Hence the data $(00)_2$ must be assigned the codeword $(11)_2$, data $(01)_2$ and $(10)_2$ must be assigned the codeword $(10)_2$ or $(01)_2$ and data $(11)_2$ must be assigned the codeword $(01)_2$ or $(10)_2$, giving rise to two different schemes. Either of the schemes may be used.

The Code table for $SEDC_2$ algorithm is shown in Table II.

TABLE III $SEDC_2$ CODE TABLE

2-bit data	$SEDC_2$ Code Scheme 1	$SEDC_2$ Code Scheme 2
00	11	11
01	10	01
10	10	01
11	01	10

V. TOTALLY SELF-CHECKING PROPERTY OF SEDC CIRCUITS

The following definitions can be used to describe a Totally Self-Checking (TSC) system [5, 14, 24, 25].

Definition 1: A circuit is *fault-secure* for a set of faults, if for any valid input and for any fault among the fault set, the circuit either produces a faulty codeword, or correct output.

Definition 2: A circuit is *self-testing* for a set of faults, if for every fault among the fault set, the circuit produces a faulty codeword for at least one valid input.

Definition 3: A circuit is *TSC* if it is both *fault-secure* and *self-testing*.

The number of faults in a system is typically modelled as a Poisson process. Hence, it is assumed that in case of self-checking circuits, faults from the fault set occur one at a time, and between any two faults a sufficient time interval exists [25, 26]. Fig. 1 shows the block diagram for a totally self-checking circuit using SEDC algorithm. The unidirectional error can occur in one of the blocks: Circuit Output F, SEDC Code C or in SEDC Checker K.

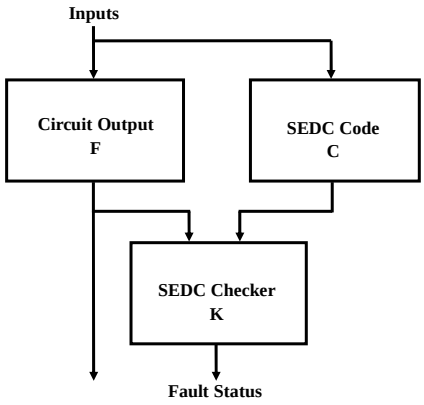

Fig. 1 Totally Self-Checking Circuit using SEDC Algorithm

Accordingly, 3 different cases arise.

Case 1 – Circuit Output F is faulty: In this case, the SEDC Code C generated for the inputs will not match with the Circuit Output F. Thus, unidirectional fault is indicated by the SEDC Checker K.

Case 2 – SEDC Code C is faulty: Even in this case, the SEDC Code C generated for the inputs will not match with the Circuit Output F. Thus, unidirectional fault is indicated by the SEDC Checker K.

Case 3 – SEDC Checker K is faulty: In this case, the SEDC Code C generated for the inputs will match with the Circuit Output F. If the SEDC Checker K is faulty, only a false-alarm is generated and the output is indicated as faulty. The unidirectional error is not propagated to further stages of the system.

This proves that the circuit encoded using SEDC algorithm is a totally self-checking circuit.

VI. IMPLEMENTATION RESULT

The SEDC architecture was generated from Verilog HDL code synthesized for ASIC technology using Mentor Graphics LeonardoSpectrum. The synthesized circuits are shown in Fig. 2, Fig. 3 and Fig. 4 for $SEDC_2$, $SEDC_3$ and $SEDC_4$, respectively.

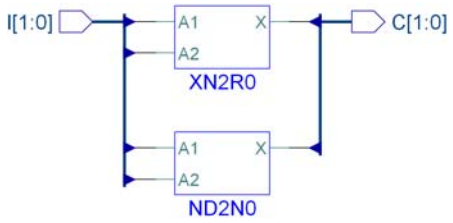

Fig. 2 Synthesized $SEDC_2$ Architecture

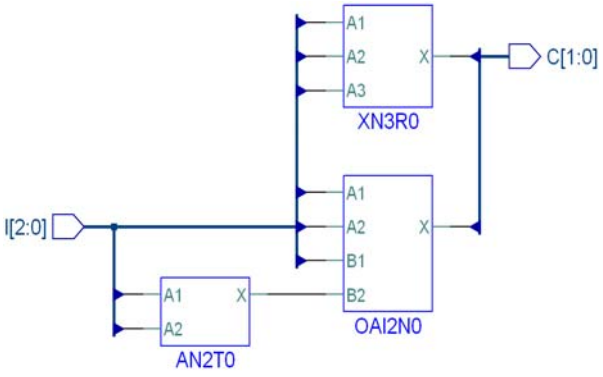

Fig. 3 Synthesized $SEDC_3$ Architecture

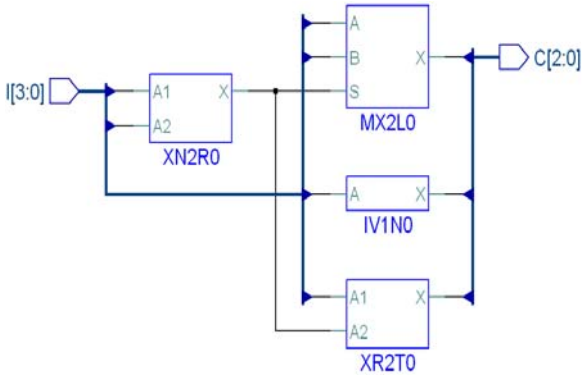

Fig. 4 Synthesized $SEDC_3$ Architecture

It can be seen from Fig. 2, Fig. 3 and Fig. 4 that the SEDC algorithm computation is done within a single clock cycle and latency is that of only two gates, which does not affect the optimized circuit performance.

VII. AREA AND LATENCY COMPARISON FOR SEDC CODE GENERATOR

The area overhead for SEDC and Berger code is computed in terms of number of MOS transistors required [27, 28]. For Berger coding, either B0 encoding scheme or B1 encoding scheme can be considered for analysis. Recall that for SEDC technique, there are two parameters 'a' and 'b'.

$SEDC_2$ architecture requires 12 MOS transistors, $SEDC_3$ architecture requires 30 MOS transistors and $SEDC_4$ architecture requires 24 MOS transistors. In general, for n-bit binary data, the number of MOS transistors required for SEDC code generation is given by Eq. (8).

$$MOS = \begin{cases} (a \times 30) + 12, \text{if } b = 2 \\ (a + 1) \times 30, \quad \text{if } b = 3 \\ (a \times 30) + 24, \text{if } b = 4 \end{cases} \tag{8}$$

The MOS transistor counts for Berger code generation combinational implementation are taken from [29]. The number of code bits for n-bit binary data is $m = \lceil \log2(n + 1) \rceil$.

Memory is required to store the code bits. A single D flip flop requires 12 MOS transistors [28].

The Berger code generation sequential implementation for n-bit binary data bits and m-bit binary code bits require n-bit shift register implemented using D flip flops and m-bit counters implemented using T flip flops for sequential implementation. A single T flip flop requires 22 MOS transistors [30]. Hence, the sequential Berger code generation unit for n-bit data bits and m-bit code bits require n × 12 MOS transistors for shift register and m × 22 MOS transistors for counter.

In Table 3, the area comparison between SEDC technique and Berger technique (combinational circuit implementation and sequential circuit implementation) is provided. It can be seen from Table 3 that when compared to Berger code technique, SEDC technique generates more code bits, but it has much simpler code generation logic. In essence, SEDC architecture has lesser overall area overhead when compared to Berger architecture. The computational latency for computing SEDC code bits for 1-bit data is equal to 1 logic gate delay and for data length n greater than 1 is equal to two logic gate delays, requiring only 1 clock cycle for computation. The sequential implementation of Berger code generation for n-bit data requires n clock cycles. The combinational of Berger code generation for n-bit data requires only a single clock but latency is equal to the number of levels of combinational logic which is definitely greater than two logic gate delays.

TABLE IIIII AREA COMPARISON

Data Bit	Berger Technique (Combinational)			Berger Technique (Sequential)			SEDC Technique		
	MOS Transistors for Code Storage	MOS Transistors for Code Generation	Total MOS	MOS Transistors for Code Storage	MOS Transistors for Code Generation	Total MOS	MOS Transistors for Code Storage	MOS Transistors for Code Generation	Total MOS
2	24	22	46	24	68	92	24	12	36
3	24	74	98	24	80	104	24	30	54
4	36	180	216	36	114	150	36	24	60
5	36	170	206	36	126	162	48	42	90
6	36	222	258	36	138	174	48	60	108
7	36	328	364	36	150	186	60	54	114

8	48	546	594	48	184	232	72	72	144

VIII. CONCLUSIONS

A new error detection coding algorithm that outperforms other error detection coding algorithms is presented. In the SEDC architecture only area is scaled, with a constant latency of 2 logic gates and requires only a single clock cycle for generating SEDC code. It is shown that the proposed SEDC algorithm is found to be significantly efficient than the existing unidirectional error detection techniques in terms of speed, latency, area and achieving 100% error detection. The SEDC architecture can be applied to any architecture at circuit level, block level or system level to make it self-checking.

ACKNOWLEDGMENT

The author Dr.S.Natarajan would like to acknowledge Chosun University, South Korea, for funding all the patents which has been registered now. The author also acknowledge All India Council for Technical Education (AICTE), Government of India, for funding this research under Research Promotion Scheme (RPS Application No.:1-1405001802).

REFERENCES

[1] J. Srinivasan, S. V. Adve, P. Bose, and J. A. Rivers, "The impact of technology scaling on lifetime reliability", in *Proc. International Conference on Dependable Systems and Networks (DSN'04),* 2004, pp. 177-186.

[2] J. R. Schwank, M. R. Shaneyfelt, D. M. Fleetwood, J. A. Felix, P. E. Dodd, P. Paillet, and V. F. Cavrois, "Radiation Effects in MOS Oxides", in *IEEE Transactions on Nuclear Science,* vol. 55, no. 4, pp. 1833-1853, 2008.

[3] B. Bose, and T. R. N. Rao, "Theory of Unidirectional Error Correcting/Detecting Codes", in *IEEE Transactions on Computers,* vol. C-31, no. 6, pp. 521-530, 1982.

[4] F. Somenzi, and S. Gai, "Fault Detection in Programmable Logic Arrays", in *Proceedings of the IEEE,* vol. 74, no. 5, pp. 655-668, 1986.

[5] N. K. Jha, and S. J. Wang, "Design and Synthesis of Self-Checking VLSI Circuits", in *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 12, no. 6, pp. 878-887, 1993.

[6] D. L. Tao, P. K. Lala, and C. R. P. Hartmann, "A MOS Implementation of Totally Self-checking Checker for the 1-out-of-3 Code", in *IEEE Journal of Solid-State Circuits*, vol. 23, no. 3, pp. 875-877, 1988.

[7] E. Stott, P. Sedcole, and P. Cheung, "Fault tolerant methods for reliability in FPGAs", in *Proc. International Conference on Field Programmable Logic (FPL)*, pp.415-420, 2008.

[8] H. Al-Asaad, B. T. Murray, and J. P. Hayes, "Online BIST for Embedded Systems", in *IEEE Design and Test of Computers*, vol. 15, no. 4, pp. 17-24, 1998.

[9] M. Abramovici, J. M. Emmert, and C. E. Stroud, "Roving STARs: An Integrated Approach to On-Line Testing, Diagnosis, and Fault Tolerance for FPGAs in Adaptive Computing Systems", in *Proc. 3rd NASA/DoD Workshop on Evolvable Hardware (EH-2001),* 2001, pp. 73-92.

[10] J. M. Emmert, C. E. Stroud, and M. Abramovici, "Online Fault Tolerance for FPGA Logic Blocks", in *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 15, no. 2, pp. 216-226, 2007.

[11] W. Chen et al., "Two New Space-Time Triple Modular Redundancy Techniques for Improving Fault Tolerance of Computer Systems", in *Proc. 6th International Conference on Computer and Information Technology (CIT'06)*, 2006, pages 175.

[12] X. She, and K. S. McElvain, "Time Multiplexed Triple Modular Redundancy for Single Event Upset Mitigation", in *IEEE Transactions on Nuclear Science,* vol. 56, no. 4, pp. 2443-2448, 2009.

[13] K. Nepal, N. Alves, J. Dworak, and R. I. Bahar, "Using Implications for Online Error Detection", in *Proc. IEEE International Test Conference (ITC 2008)*, 2008, pp. 1-10.

[14] D. K. Pradhan, and J. J. Stiffler, "Error-Correcting Codes and Self-Checking Circuits", in *Computer*, vol. 13, no. 3, pp. 27-37, 1980.

[15] N. Alves, "State-of-the-Art Techniques for Detecting Transient Errors in Electrical Circuits", in *IEEE Potentials*, vol. 30, no. 3, pp. 30-35, 2011.

[16] M. Boudjit, M. Nicolaidis, and K. Torki, "Automatic Generation Algorithms, Experiments and Comparisons of Self-checking PLA schemes using parity codes", in *Proc. 4th European Conference on Design Automation*, 1993, pp. 144-150.

[17] M. K. Stojcev, and M. D. Krstic, "Parity Error Detection in Embedded Computer System", in *Proc. 5th International Conference on Telecommunications in Modern Satellite, Cable and Broadcasting Service (TELSIKS 2001)*, vol. 2, 2001, pp. 445-450.

[18] R. K. James, T. K. Shahana, K. P. Jacob, and S. Sasi, "Fault Tolerant Error Coding and Detection using Reversible Gates", in *Proc. 2007 IEEE Region 10 Conference (TENCON 2007)*, 2007, pp. 1-4.

[19] G. C. Cardarilli, S. Pontarelli, M. Re, and A. Salsano, "Concurrent Error Detection in Reed–Solomon Encoders and Decoders", in *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 15, no. 7, pp. 1-4, 2007.

[20] S. J. Piestrak, "Design of Fast Self-Testing Checkers for a Class of Berger Codes", in *IEEE Transactions on Computers*, vol. C-36, no. 5, pp. 629-634, 1987.

[21] A. Morozov, V. V. Saposhnikov, VI. V. Saposhnikov, and M. Gossel, "New self-checking circuits by use of Berger-codes", in *Proc. 6th IEEE International On-Line Testing Workshop*, 2000, pp. 141-146.

[22] B. Bose, and D. J. Lin, "Systematic Unidirectional Error-Detecting Codes", in *IEEE Transactions on Computers*, vol. C-34, no. 11, pp. 1026-1032, 1985.

[23] H. Dong, "Modified Berger Codes for Detection of Unidirectional Errors", in *IEEE Transactions on Computers*, vol. C-33, no. 6, pp. 572-575, 1984.

[24] D. A. Anderson, and G. Metze, "Design of Totally Self-Checking Check Circuits for m-Out-of-n Codes", in *IEEE Transactions on Computers*, vol. C-22, no. 3, pp. 263-269, 1973.

[25] G. M. Koob, and C. G. Lau, *Foundations of Dependable computing: System Implementation*, Kluwer Academic Publishers, 1994.

[26] J. E. Smith, and G. Metze, "Strongly Fault Secure Logic Networks", in *IEEE Transactions on Computers*, vol. C-27, no. 6, pp. 491-499, 1978.

[27] J. M. Rabaey, A. P. Chandrakasan, and B. Nikolic, "Digital Integrated Circuits: A Design Perspective", 2nd Edition, Pearson Education, 2003.

[28] http://en.wikipedia.org/wiki/Transistor_count

[29] D.A. Pierce Jr, and P.K. Lala, "Modular Implementation of Efficient Self-Checking Checkers for the Berger Code," *J. of Electronic Testing: Theory and Applicat.,* vol. 9, no. 3, pp. 279-294, 1996.

[30] S. Ziabakhsh, and M. Zoghi, "Design of a Low-Power High-Speed T-Flip-Flop Using the Gate-Diffusion Input Technique," in *Proc. 17th Telecommunications Forum TELFOR2009,* 2009, pp. 1470-1473.

Natarajan Somasundaram was born in Coimbatore, Tamilnadu, India in 1980. He received the B.E. degree in Electronics and Communication Engineering from Amrita Institute of Technology and Science, Bharathiar University, Coimbatore in 2002, M.E. degree in Embedded System Technologies and Ph.D. from Anna University, Chennai, India in 2005 and 2009, respectively.

He joined the research team of Dr.A.P.J.Abdul Kalam at Anna University, Chennai in 2002 as a Junior Scientist. He has been awarded various fellowships. He was entitled as "Scientist" by Dr.A.P.J.Abdul Kalam in 2003. He was awarded a university gold medal for his M.E. degree. He was a post-doctoral researcher at Computer System Lab, Chosun University, South Korea during June 2011 to January 2012 and later on honored and promoted as a Visiting Researcher. His post-doctoral research was on Bio-Inspired Reconfigurable Fault Tolerant Architecture based on which 7 International Patents has been filed, out of which 5 has been registered and 2 are under examination. He joined the Department of Electronics and Communication Engineering at SSM College of Engineering, Komarapalayam, Tamilnadu in 2012 where he is currently a Professor.

Dr.S.Natarajan is a senior member of IACSIT, member of IAENG and reviewer of reputed journals including IET, ACM and IASIR. His research interests include Reconfigurable Architectures, Fault-tolerant Systems, VLSI Design and Digital Image Processing.

Farhad Mehdipour received the B.Sc. degree from Sharif University of Technology in 1996, and the M.Sc. and Ph.D. degrees in Computer Systems Architectures from the Amirkabir University of Technology in 1999 and 2006, respectively.

He was a visiting researcher at System LSI Lab, Kyushu University during November 2005 to June 2006. He joined the School of Information Science and Electrical Engineering at the Kyushu University as a post-doctoral researcher in December 2006. Since August 2010, he has been an Associate Professor in E-JUST Center at Kyushu University.

Dr.Farhad Mehdipour is a member of IEEE and his research interests include cyber-physical systems, electronic design automation, and high-performance and low-power micro-architectures.

Ramadass Narayanadass was born in Chennai, Tamilnadu, India in 1975. He received the B.E. degree in Electrical and Electronics Engineering from Madras University in 1997, M.E. degree in Applied Electronics and Ph.D. from Anna University, Chennai, India in 2001 and 2008, respectively.

He has been associated with the Faculty of Information and Communication Engineering, Anna University, Chennai, since 2001. He is currently an Associate Professor in Department of Electronics and Communication Engineering at Anna University, Chennai.

His research interests include Embedded Systems, VLSI design and Reconfigurable Computing.

Yeragudipati Venkata Ramana Rao was born in Gudur, Andhra Pradesh, India in 1963. He received the B.E. degree in Electronics and Communication Engineering from S.V. University, Tirupathi in 1985, M.Tech. degree in Communication Systems and Ph.D. from Indian Institute of Technology, Chennai, India in 1986 and 1990, respectively.

He joined the Department of Electronics and Communication Engineering at Anna University, Chennai in 1991 where he is currently a Professor.

His research interests include Digital CMOS VLSI, VLSI Signal Processing, Digital Signal Processing, Digital Image Processing and Neural Networks.