Self-Checking Look-up Tables using Scalable Error Detection Coding (SEDC) Scheme

Jeong-A Lee, Zahid Ali Siddiqui, Natarajan Somasundaram, and Jeong-Gun Lee

Abstract—**In this paper, we present Self-Checking look-up-table (LUT) based on Scalable Error Detection Coding (SEDC) scheme for use in fault-tolerant reconfigurable architectures. SEDC scheme has shorter latency than any other existing coding schemes for all unidirectional error detection and the LUT execution time remains unaffected with self-checking capabilities. SEDC scheme partitions the contents of LUT into combinations of 1-, 2-, 3- and 4-bit segments and generates corresponding check codes in parallel. We show that the proposed LUT with SEDC performs better than LUT with traditional Berger as well as Partitioned Berger Coding schemes. For 32-bit data, LUT with SEDC takes 39% less area and 6.6 times faster for self-checking than LUT with traditional Berger Coding scheme.**

Index Terms—**Self-checking circuit, error detection coding, unidirectional transient errors, look-up table, reconfigurable architecture**

I. INTRODUCTION

Field Programmable Gate Array (FPGA) is a reconfigurable architecture suitable for embedded systems in providing various functionalities quickly. Scaling of silicon technology to nano dimensions has significantly improved FPGA technology but the interaction of neutron and alpha particles with nano-sized semiconductor devices may lead to transient hardware faults in addition to certain intermittent or persistent faults due to design and manufacturing defects. Studies show that the majority of errors are caused by transient faults [1]. The types of faults within a VLSI circuit have been analyzed and found to be of the type which would tend to affect the bits in a unidirectional manner. Unidirectional errors can alter the node logic from zero to one or from one to zero, but not both at the same time [2].

A Look-Up Table (LUT)-based function generator is a basic block of a reconfigurable architecture. In fault-tolerant reconfigurable systems, self-checking LUTs must detect all the unidirectional errors even when the system is functioning, that is, online fault detection. Online fault detection methods can be broadly classified as redundancy and error coding [3]. When using modular or time redundancy techniques there is an area overhead or latency overhead by two or three times [4]. Error coding technique is more efficient that the other fault detection methods in terms of area, speed and fault coverage and most suited for implementing self-checking circuits [5]. Many unidirectional error detecting codes like Parity code, Hamming code, Reed Solomon code, Berger code and Bose code have been reported in the literature. Among these coding algorithms, Berger coding algorithm [6] alone has 100% fault coverage for all unidirectional errors. The error-check code is generated by counting the number of logic 0's, or sometimes logic 1's, in the input data and representing the count as a binary number [7]. The sequential circuit implementation of this technique requires resource overhead to implement counter circuits and takes multiple clock cycles to detect the error. The

Manuscript received May. 23, 2012; accepted Jun. 25, 2013
School of Electronics Engineering, Kyungpook National University
1370 Sankyuk-dong, Buk-gu, Daegu 702-701, Korea
E-mail : sdyu@mail.knu.ac.kr

combinational circuit implementation of Berger can reduce latency with additional area resources. Incorporating Berger code into a delay-optimized 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 input data length.

In this paper, we presented method to employ SEDC scheme for designing self-checking LUTs and show that SEDC is better in terms of area as well as delay in comparison with Berger coding scheme. The rest of the paper is organized as follows: In Section II the introduction to SEDC scheme for 2-bit input data is given with method to scale it to n-bit data. The technique with which SEDC scheme detects all unidirectional errors is also illustrated in Section II. In Section III, we compared SEDC scheme with traditional Berger scheme and with modified version of Berger scheme, we called as "Partitioned Berger". The comparison is done with respect to area and delay. Note that the data partitioning has been used in the coding for data communication and our approach is similar to the partitioning used in the field of communication theory. However, the goal of our research is focused on the high speed code generation circuits for fault-tolerant hardware design with minimal overhead. The Totally-Self Checking property of SEDC is discussed in Section IV with a little introduction to TSC SEDC checker. In Section V we discuss how this TSC property of SEDC scheme can be used to design TSC LUTs while Section VI we compare the area and delay performance of SEDC with sequential and combinational implementations of Berger implement-tations for 6-input LUT. Finally Section VII concludes the paper.

II. SCALABLE ERROR DETECTION CODING (SEDC) ALGORITHM

The new Scalable Error Detection Coding algorithm [8] presented is formulated and architecture is designed in such a way that only area is scaled, while latency remains same for n > 3.

For input binary data D of length n-bits represented as (D_{n-1},….., D_2, D_1, D_0) with $D_i \in \{0, 1\}$ for $0 \le i \le n-1$, two parameters ‘a’ and ‘b’ are computed using Eq. (1) where, parameter ‘a’ can only be a positive integer, and parameter ‘b’ can take values only from 1, 2, 3 or 4.

$$a = \frac{n - \max(b)}{4} \tag{1}$$

where b should be selected among 1, 2, 3, and 4 to make 'a' a positive integer number.

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_j, ..., C_2, C_1, C_0) with $C_j \in \{0, 1\}$ for $0 \le j \le m-1$, is then computed as per Eq. (2).

$$m = \lceil bg_{\ 2}(n + 1 - 4a) \rceil + 3a \tag{2}$$

After computing the values for parameters ‘a’ and ‘b’, the SEDC code ‘C’ for input binary data ‘D’ is computed. SEDC is designed to generate codes basically for 1, 2-, 3-, and 4-bit data and accordingly referred to as $SEDC_1$, $SEDC_2$, $SEDC_3$ and $SEDC_4$ scheme, respectively. It is then extended for any integer values of n, as shown in Fig. 1.

For 1-bit data, the complement of data bit is represented as $SEDC_1$ code.

Table 1 shows the code table for 2-bit data, i.e., $SEDC_2$ scheme. For discussion in this paper, code scheme 1 is considered. In order to detect unidirectional errors, we assign same code for both data $(01)_2$ and $(10)_2$ (since it contains both 1 and 0). The two cases in Table 1 can be shown to detect all unidirectional errors for 2-bit data and can be extended to detect all unidirectional errors for 3-bit data, as shown later.

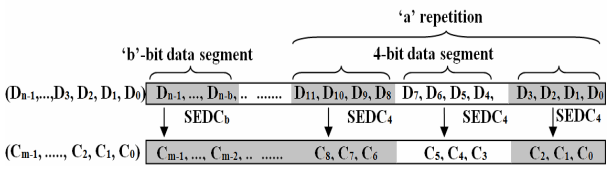


Fig. 1. SEDC scheme for given data word. $SEDC_b$ is one of $SEDC_1$, $SEDC_2$, $SEDC_3$ or $SEDC_4$.

Table 1. Code Table for $SEDC_2$

2-bit Data	$SEDC_2$ Scheme 1	$SEDC_2$ Scheme 2
00	11	11
01	01	10
10	01	10
11	10	01

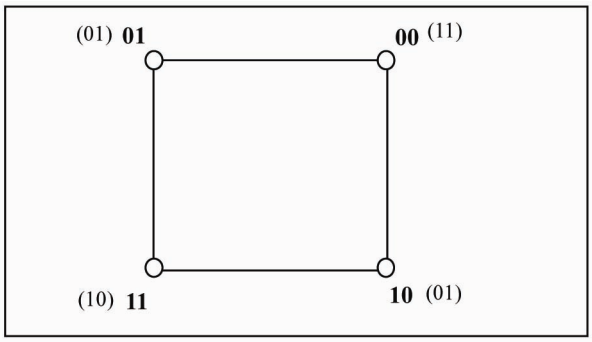


Fig. 2. 2-D square illustration of $SEDC_2$ scheme.

The SEDC coding scheme assigns code word to different data words with a unique criteria. Whenever there is a change of bit (or bits) in data word from '0' → '1', the change is reflected in code word in opposite way, i.e., the code changes from '1' → '0', and vice versa. Also the data words which differ each other by unidirectional bit (or bits) flipping, are assigned different code words. Fig. 2 is a 2-D square illustration of $SEDC_2$ scheme where nodes represent data words and their corresponding code words (written in brackets).

For example, if there is an unidirectional error from '00' to '01', the code word also changes from '11' to '01' to alarm the error. Note that the LSB of the data word changes in one direction ('0' to '1') while the MSB of the code word changes in opposite direction ('1' to '0').

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

$$(C_1,C_0)=\begin{cases}\text{SEDC}_2(D_1,D_0), & \text{if } D_2=0\\ \text{1's complement}\left(\text{SEDC}_2\left(\overline{D_1},\overline{D_0}\right)\right), & \text{if } D_2=1\end{cases} \quad (3)$$

Table 2 illustrates the scaling of $SEDC_2$ to $SEDC_3$. $SEDC_2$ is embedded in the $SEDC_3$ scheme, i.e., the first four code words are same as $SEDC_2$ as shown in Fig. 3 with Red box. Remaining four code words are generated by following steps: For an example with '100' data,

(1) Invert all the data bits (we take '100'→'011') .

(2) Find the $SEDC_2$ code word corresponding to the inverted data resulting from step 1 ('011'→'01').

(3) Now invert the $SEDC_2$ code word which came from step 2 ('01'→'10').

(4) The inverted $SEDC_2$ code word resulted in step 3 becomes the code word for the data word selected in step 1('100' [$SEDC_2$] → '10', which can be verified from Table 1).

Table 2. Code Table for $SEDC_3$

D		C
D2	*D1D0*	*(C1C0)*
0	00	11
	01	01
	10	01
	11	10
1	00	01
	01	10
	10	10
	11	00

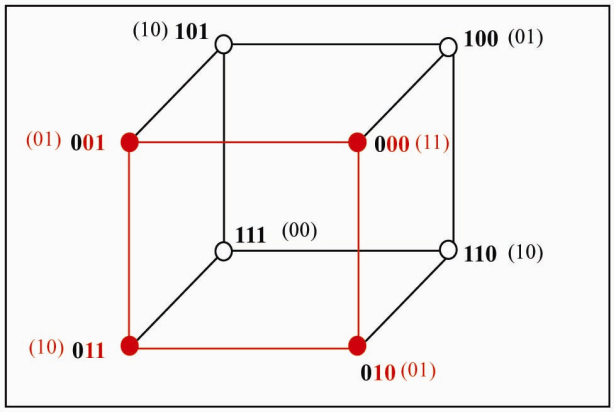


Fig. 3. 3-D cube illustration of $SEDC_3$ scheme.

Fig. 3 shows a 3-D cube diagram for understanding how $SEDC_3$ code works for detecting all unidirectional errors. Same notations are used in Fig. 3 as in Fig. 2. The red part of the cube shows the embedded $SEDC_2$ coding scheme in $SEDC_3$. Note that when there is a 2-bit unidirectional change in data word '001' to '111' (two MSB's changing from '00' to '11'), the code is changing in the opposite direction (LSB of the code changes from '1' to '0'). Further examine of the cube tells that all unidirectional errors are detectable by SEDC scheme.

SEDC function for 4-bit data, $SEDC_4$, is formulated as per Eqs. (4) and (5).

$$(C_1,\ C_0)=SEDC_3(D_2,\ D_1,\ D_0) \quad (4)$$

$$C_2=\text{NOT}(D_3) \quad (5)$$

From Eq. (4) it can be seen that in $SEDC_4$, the MSB of the code word is completely dependent upon MSB of the data word, hence any change in the MSB of the data

word is detected.

We also verified that every possible combination of unidirectional errors that can ccurre in real situation can be detected by exhaustive fault simulation for $SEDC_2$, $SEDC_3$ or $SEDC_4$ [8].

In general, for $SEDC_n$ function, the n-bit binary data is grouped into one 'b'-bit segment and 'a' number of 4-bit segments, on which $SEDC_b$ and $SEDC_4$ functions are applied. This is a unique feature of this scalable algorithm. Fig. 1 shows SEDC code generation for n-bit input data. It is noteworthy that each group of data segment and corresponding code segment is independent to each other. This *independency* makes our SEDC scheme scalable. Moreover, the overall latency is fixed for n bigger than 3 which equals to the latency of $SEDC_4$ module.

We observed that $SEDC_3$ embedding $SEDC_2$ with scheme 2 in Table 1 happens to be Berger code for 3-bit data. In the case of $SEDC_4$, it is different and performs better than Berger code. With 4-bit data, the maximum number of '1' or '0' is 4 so we have to encode the five possible cases, 0, 1, 2, 3, 4 that needs at least 3-bit check bits for Berger Code. With 3-bit check bits, Berger code only uses 5 cases out of 8 cases. On the other hand, our $SEDC_4$ uses all 8 cases produced from the binary combinations of 3-bits as check bits. Note that the code assignment of our $SEDC_4$ help minimizing logic circuits for code generation compared with that of Berger code due to the better placement of '1' in Karnaugh-map. The circuit of $SEDC_4$ code generation is more efficient than the circuit of the 4-bit Berger code generation with respect to area and performance.

III. Area and Delay Comparison of SEDC Code Generator

To the best of our knowledge there has been no such implementation in which the input bits are partitioned for scalable code generation. We have exploited this partitioning method for parallelism in hardware, and thus resulting in faster implementation of Scalable Error Detection Scheme. We also apply this partitioning for Berger scheme for performance comparison and named it as "Partitioned Berger".

Parallelism in any hardware requires more area resources, as does with the SEDC. When compared to partitioned Berger coding scheme, SEDC scheme requires less area.

Table 3 contains total number of transistors for traditional Berger, partitioned Berger and SEDC scheme. The transistor counts for Berger scheme are taken from [7] and [9]. Since the Berger codes have used the number of '1' or '0' in data bits, a counter [7] or an adder [9] based implementations of Berger code generation circuits have been used traditionally. Due to the different implementation styles, the circuit implementations can be different in size and complexity. As SEDC code size increases faster than Berger scheme, so for fair comparison, we have also considered the code storage part (1 bit storage is implemented using 12 transistors) for all three schemes in Table 3. It is clear that even after having bigger code size, SEDC implementation takes less area in total than traditional Berger codes as shown in Table 3.

Table 3. Circuit Size Comparison

Data Length	Traditional Berger code generator [9]			Partitioned Berger (using the implementation of [9])			Partitioned Berger (using the implementation of [7])			SEDC		
	Area	*Code Area*	*Total*	*Area*	*Code Area*	*Total*	*Area*	*Code Area*	*Total*	*Area*	*Code Area*	*Total*
2	12	24	36	12	24	36	22	24	46	12	24	36
3	28	24	52	28	24	52	67	24	91	28	24	52
4	52	36	88	52	36	88	142	36	178	30	36	66
5	80	36	116	52	48	100	142	48	190	32	48	80
7	136	36	172	80	60	140	209	60	269	58	60	118
8	148	48	196	104	72	176	284	72	356	60	72	132
16	356	60	416	208	144	352	568	144	712	120	144	264
32	788	72	860	416	288	704	1136	288	1424	240	288	528

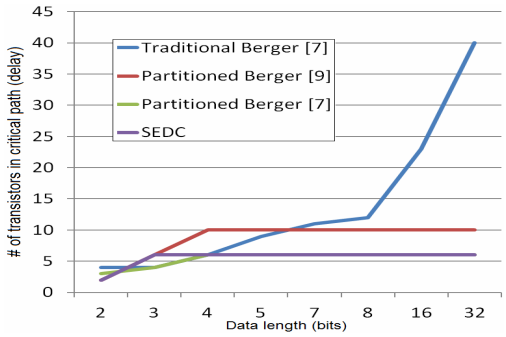


Fig. 4. Delay comparison between traditional Berger, partitioned Berger and SEDC scheme.

If we compare the delay performance of SEDC scheme with Berger codes, SEDC scheme totally outperform the traditional Berger scheme (see Fig. 4), due to the parallelism in the SEDC hardware architecture. On the other hand, partitioned Berger (both [7] and [9] shows improvement in delay performance, which is achieved at the cost of more area overhead and thus more power dissipation. Hence, the overall performance of SEDC scheme is better as compared to traditional Berger as well as partitioned Berger scheme.

In the next section, we discuss how this SEDC scheme is totally self-checking, and can be used to design TSC LUTs.

IV. Totally Self-Checking Property of SEDC Circuits

The following definitions can be used to describe a Totally Self-Checking (TSC) system [9-12].

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 modeled 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

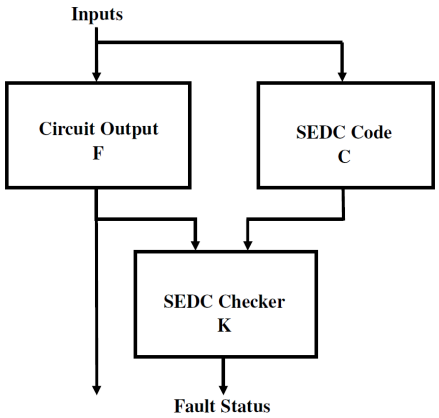


Fig. 5. Totally Self-Checking circuit using SEDC Algorithm.

interval exists [13, 14]. Fig. 5 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.

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 fault-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 totally self-checking circuit. In the next section we apply this TSC property of SEDC scheme to design TSC LUTs.

V. SEDC-based Self-Checking LUT Architecture

Fig. 6 shows the SEDC-based self-checking single or multiple output K-bit LUT architecture. The SEDC code bits for LUT contents are pre-computed and are downloaded at the time of configuration of FPGA.

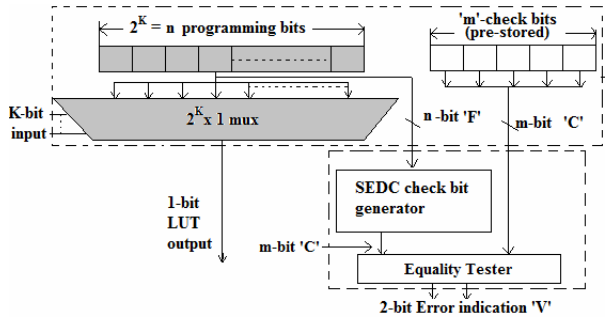


Fig. 6. SEDC-based Self-Checking LUT Architecture.

During LUT operation, the SEDC code bits are generated for LUT content and compared with the pre-computed code bits by the equivalence tester for validating the LUT output as correct or faulty. These operations are performed continuously. A faulty output indicates the presence of unidirectional error in the LUT block. In this paper, only the SEDC code bit generator and pre-computed code bits circuits are considered for comparison with Berger scheme.

As mentioned in Section III, the SEDC code generation unit requires few gates, that are implemented with [2 + (a × 30)] MOS transistors if 'b = 1', [12 + (a × 30)] MOS transistors if 'b = 2', [28 + (a × 30)] MOS transistors if 'b = 3' or [30 + (a × 30)] MOS transistors if 'b = 4'. As we already discussed in Section II that the overall latency of SEDC scheme is fixed for n > 2.

In Berger coding scheme, the number of code bits for data length of n bits is $m = \lceil \log_2(n + 1) \rceil$. The sequential implementation of Berger code generator unit requires n-bit shift register implemented using D flip flops and m-bit counters implemented using T flip flops. Here we have assumed that each D and T-flip flop is implemented using 12 MOS transistors. Hence, the Berger code generation unit for data length n bits and m-bit code bits requires (n + m) × 12 MOS transistors. The computational latency for computing Berger code bits for data length n bit is n clock cycles. Also, the clock cycles must have a minimum time period which depends upon the critical path delay of the circuit. In Table 4, we show the critical path of the sequential Berger code generator scheme in terms of numbers of transistors. The computational latency of Berger code generator will be 'n' times to the critical path time since n number of cycles are required to complete the operation.

Data for latency optimized combinational implementation of Berger code generator is taken from [7]. The circuit diagrams for up to 32-bit Berger code generator are given in [7], from where we have calculated the number of transistors for Berger implementation. The scaling scheme given in [7] is such that the information bits I are divided into two parts, $I_r = I/2$ and $I_{r*} = I-[I/2]$. If we suppose the case when I = 32, then $I_r = 16$ and $I_{r*} = 16$, which means that at least two 16-bit Berger code generators are required, along with some additional circuitry. This additional circuit is made by different type of binary adders, which depends on the information bit length. As the circuit diagram for 64-bit Berger code generator is not given in the paper, so we approximated the number of transistors by using two 32-bit Berger code generators and neglecting the extra adder circuitry.

Table 4. Implementation results for Fault-tolerant 6-bit LUT with single output

Resource Requirements	SEDC Algorithm	Berger coding Algorithm (Sequential Implementation)	Berger coding Algorithm (Combinational Implementation) [7]
(Data bit size = 64 bits)			
Code size (bits)	48 bits	7 bits	7 bits
Area required for code storage (# of transistors)	576 MOS	84 MOS	84 MOS
Code generation logic	480 MOS	852 MOS	3878 MOS*
Total # of transistors	1056 MOS	936 MOS	3962 MOS
Clock cycle	-	64	-
Latency (#of transistors in critical path)	6 MOS	192 MOS	46 MOS*

VI. Comparison Results of Area and Delay of TSC LUTs

Comparison of the proposed SEDC-based implementation of code bit generator circuit for self-checking 6-bit LUT and implementation using Berger coding algorithm [7] is shown in Table 4. Logic given in Section II for implementing $SEDC_2$, $SEDC_3$ and $SEDC_4$ circuits is utilized to develop all the SEDC circuits given in Tables 3 and 4. Circuits are evaluated using *Logic Friday* software and tested with Verilog HDL code, synthesized by Altera's Quartus II. It can be seen from the tables that for the case of Berger coding algorithm,

with increase in binary data length, either the clock cycle increases due to shift register and binary counters, or there is an increase in latency due to adder tree [7], both of which are undesired for delay-optimized reconfigurable embedded architectures. SEDC scheme, on the other hand, requires more storage resources but the computation is done within a single clock cycle and the maximum latency is limited to 6 equivalent MOS transistor levels, which does not affect the optimized LUT performance.

VII. CONCLUSIONS

A new error detection coding algorithm achieving all unidirectional error detection with a maximum latency equivalent to 6 MOS transistor levels and requires only a single clock cycle is proposed. The algorithm outperforms other error detection coding algorithms. An architecture for fault-tolerant self-checking LUT is also proposed. We show that only area is scaled with SEDC architecture and latency remains constant regardless of input data length. The proposed SEDC architecture can also be extended to self-checking Programmable Arithmetic and Logic Unit (PALU)-based architectures.

ACKNOWLEDGMENTS

This study was supported in part by research fund from Chosun University, 2012

REFERENCES

[1] J. R. Schwank et al., “Radiation effects in MOS Oxides,” IEEE Trans. Nucl. Sci. vol. 55, no. 4, pp. 1833-1853, 2008.

[2] B. Bose, and D. K. Pradhan, “Optimal Unidirectional Error Detecting/Correcting Codes,” IEEE Trans. Comput. vol. C-31, no. 6, pp. 564-568, 1982.

[3] E. Stott, P. Sedcole, and P. Cheung, “Fault Tolerant methods for reliability in FPGAs,” Proc. Int. Conf. Field Programmable Logic, pp. 415-420, 2008.

[4] R. V. Kshirsagar, and R. M. Patrikar, “A novel fault tolerant design and an algorithm for tolerating faults in digital circuits,” Proc. 3rd Int. Design and Test Workshop (IDT) pp. 148-153, 2008.

[5] N. Alves, “State-of-the-Art techniques for detecting transient errors in electrical circuits,” IEEE Potentials vol. 30, no. 3, pp. 30-35, 2011.

[6] A. Morozov et al., “New self-checking circuits by use of Berger-codes,” Proc. IEEE Int. On-Line Testing pp. 141-146, 2000.

[7] 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.

[8] S. Natarajan, J. A. Lee, and J. G. Lee, Scalable Error Detection Coding (SEDC) Generator, Self-Checking look-up Table having the Generator and Method of Scalable Error detection Coding, Korean Patent Application no. 10-2011-0098730, filed 29 September 2011.

[9] 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.

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

[11] 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.

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

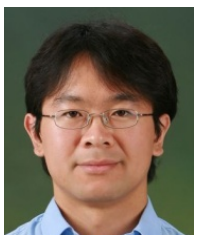
[13] J. E. Smith, and G. Metze, "Stronlgy Fault Secure Logic Networks", in *IEEE Transactions on Computers*, vol. C-27, no. 6, pp. 491-499, 1978.

[14] C. Metra, M. Favalli, and B. Ricco, "Self-Checking Detection and Diagnosis of Transient, Delay, and Crosstalk Faults Affecting Bus Lines", in *IEEE Transactions on Computers,* vol. 49, no. 6, pp. 560-574, 2000.

Jeong-A Lee received B.S. in Computer Engineering from Seoul National University in 1982, M.S. in Computer Science from Indiana University in Bloomington and Ph.D in Computer Science from UCLA in 1990. From 1990 to 1995, she was an Assistant Professor with the Department of Electrical and Computer Engineering, University of Houston, Texas. Since 1995 she has been Professor and Head of Computer Systems Laboratory, Chosun University in Korea. From 2008 to 2009, she served as a director of Electrical and Computer Science and Engineering, National Research Foundation of Korea. Her research activities cover high-performance computer architecture, fast digital and CORDIC arithmetic, configurable computing and bio-inspired fault-tolerant computing. She is the author and coauthor of more than 100 reviewed journal and conference papers. Professor Lee is a member of National Academy of Engineering, Korea.

Zahid Ali Siddiqui received B.E. degree in Electronics from NED University of Engineering and Technology, Karachi in 2009 and got lectureship in the Department of Electronic Engineering at NED University in 2010. He is recipient of Global IT scholarship and currently doing M.S. in Computer Science from Chosun University. His research interests are related to embedded systems design, fault tolerant FPGA architecture and reconfigurable computing.

Natarajan Somasundaram received the B.E. degree in Electronics and Communication Engineering from Amrita Institute of Technology and Science, Bharathiyar University in 2002, M.E. degree in Embedded System Technologies from College of Engineering, Anna University Chennai in 2005, and Ph.D. degree from College of Engineering, Anna University Chennai in 2009, respectively. From 2010 to 2011, he was an Assistant Professor at Department of Electronics and Communication Engineering, Bannari Amman Institute of Technology, Sathyamangalam, INDIA. From 2011 to 2012, he was a post-doctoral student with the Chosun University, Gwangju, South Korea. In 2012, he joined the Department of Electronics and Communication Engineering, SSM College of Engineering, Komarapalayam, INDIA, where he is currently an Associate Professor. His research interests include Reconfigurable embedded architecture, bio-inspired computing and VLSI design.



Jeong-Gun Lee received the B.S. degree in computer engineering from Hallym University in 1996, and M.S. and Ph.D degree from Gwangju Institute of Science and Technology (GIST), Korea, in 1998 and 2005. He is currently an Associate Professor in the Computer Engineering department at Hallym University. Prior to joining the faculty of Hallym University in 2008, he was a postdoctoral researcher of the Computer Lab. at the University of Cambridge, UK. His research interest focus on an asynchronous circuit design, network-on-chip, reconfigurable processor and multi-core systems for multi-media applications.