August 2013

Master’s Degree Thesis

Online Fault Detection and Reconfiguration of ALU using Scalable Error Detection Coding Scheme

Graduate School of Chosun University

Department of Computer Engineering

Zahid Ali

확장가능한 에러탐지코딩기법 활용 실시간 오류탐지 및 재구성이 가능한 연산기

Online Fault Detection and Reconfiguration of ALU using Scalable Error Detection Coding Scheme

August 23, 2013

Graduate School of Chosun University

Department of Computer Engineering

Zahid Ali

Online Fault Detection and Reconfiguration of ALU using Scalable Error Detection Coding Scheme

Advisor: Prof. Jeong-A Lee

This Thesis is submitted to Graduate School of Chosun University in partial fulfillment of the requirements for a Master's degree

April 2013

Graduate School of Chosun University

Department of Computer Engineering

Zahid Ali

□

Thesis Examination Committee

위원장 조선 대학교 교수 [illegible] (인)

위 원 조선 대학교 교수 김 영식 (인)

위 원 조선 대학교 교수 이 정아 (인)

2013 년 4 월

Graduate School of Chosun University

I dedicate this thesis to my parents, Roshan Ali & Rahat Tameez for their love and support, and to my teachers, especially Dr. Shoaib Zaidi for his trust in me.

Table of Contents

List of Figures

List of Tables

List of Acronyms

ASO	Add/Subtract Operation
TSC	Totally Self Checking
CDF	Chain Description File
ALU	Arithmetic and Logic Unit
BCP	Berger Check Prediction
BCH	Bose Chaudhuri Hocquenghem
CO	Compare Operation
SEDC	Scalable Error Detection Coding
$SEDC_2$	SEDC codes for 2-bit input
$SEDC_3$	SEDC codes for 3-bit input
$SEDC_4$	SEDC codes for 4-bit input
SFS	Strongly Fault Secure
AUED	All Unidirectional Error Detection
MOS	Metal Oxide Semiconductor
CED	Concurrent Error Detection
BO	Boolean Operation
SRO	Shift/Rotate Operation
DMR	Double Modular Redundancy
FPGA	Field Programmable Gate Array
FF	Flip Flop
LSB	Least Significant Bit
MSB	Most Significant Bit
SEU	Single Event Upset
VLSI	Very Large Scale Integration
SRAM	Static Random Access Memory
TMR	Triple Modular Redundancy

초 록

확장가능한 에러탐지코딩기법 활용 실시간 오류탐지 및 재구성이 가능한 연산기

시디키 샤이크 자히드 알리
지도교수: 이정아, 교수, Ph.D.
컴퓨터공학과
조선대학교 대학원

마이크로 전자 회로 및 SRAM 기반의 FPGA 디바이스는 트랜지스터의 크기 감소 및 높은 패키징 밀도로 인한 고장 및 오류에 더 취약해지고 있기 때문에, 오류 검출은 시스템 신뢰성 측면에서 아주 중요한 문제이다.
오류는 크게 소프트 오류와 하드 오류로 분류할 수 있다. 하드 오류는 지속적인 결함으로 발생하나, 소프트 오류는 일시적 또는 간헐적으로 결함이 발생한다. 지금까지 알려진 바에 의하면, 오류의 대부분이 일시적인 오류로 인해 발생된다.
일시적인 오류로 인한 소프트 오류를 검출하는 기술은 일반적으로 프로세서의 성능감소를 가져오거나, 오류 검출에 필요한 추가적인 하드웨어 자원 및 전력소모가 요구되어 이의 적절한 조율과정을 거치게 된다. 이중 모듈 방식은 공통 모드 고장(CMF)을 포함하여 대부분의 오류에 대응할 수 있지만, 구현상의 변화를 주어야 하고, 하드웨어 자원측면에서 두 배의 오버 헤드 비용이 필요하다. 삼중 모듈 방식(TMR)은 구현상의 변화없이 같은 모듈을 세 번 반복하여 사용할 수는 있으나, 하드웨어 오버헤드 비용이 높고, 각 모듈의 동일한 출력을 비교하기 위해 voting 회로가 필요하다. 추가적인 하드웨어 오버헤드를 피하기위하여, 피연산자를 회전 또는 이동하여 연산을 다시 한 후, 오류를 검출하기도 하는데, 이 경우 연산에 소요되는 시간이 추가되어 이 또한 시스템의

시간적인 오버헤드가 된다. 연산 중 오류검출 (CED) 방법은 산술 코드, 버저 코드, 패리티 코드 등을 사용하여 인코딩을 한 상태에서 연산을 수행한다. CED 기술의 효율적인 구현을 위해, 발생할 가능성이 높은 결함유형을 고려하는 것이 중요하다. VLSI 회로에서 고장은 일반적으로 단일방향 오류인 것으로 알려졌다. 단일방향 오류는 동시에 0 에서 1 또는 1 에서 0 으로 오류가 발생되지 않으며, 오류의 방향이 단일방향인 오류이다. 단일방향 오류 감지 (AUED) 기술은 하드웨어 자원 오버 헤드를 고려할 때, 오류 감지의 효율을 높인다. 본 논문에서는 여러개의 단일방향 오류를 탐지할 수 있는 확장 가능한 오류탐지 코딩 (SEDC)을 활용한다. SEDC 방식은 입력 데이터를 4-비트 이하로 분할하고, 분할 된 데이터를 동시에 인코딩하여 오류탐지코드를 할당한다. 따라서, 기존의 단일방향 오류탐지 방식과 달리, 입력데이터의 크기 'n'이 증가하더라도, 오류탐지코드 생성에 소요되는 시간은 증가되지 않으며, 기본 모듈들의 활용이 가능하여 구현의 복잡성도 증가되지 않는다.

본 논문에서는 SEDC 방식을 사용하여, 단일 이벤트 오류발생에 대해 높은 내결함성을 가진 ALU 구조를 제안한다. 기존 관련연구에 비하여 제안된 SEDC 코드 기반 ALU 는 하드웨어 오버헤드와 지연시간 측면에서 더 나은 결과를 보인다. SEDC 기반 32 비트 ALU 의 ASIC 구현은 버저 코드 예측 ALU 에 비하여 하드웨어 자원을 34% 절약했다 [5]. SEDC 기반 16 비트 ALU 의 FPGA 구현은 버저 코드 예측 ALU 에 비하여 하드웨어 자원을 39% 감소시켰다. 이와 더불어, SEDC 기반 확장가능한 자가완전검증회로(TSC)가 하드웨어 자원과 지연시간을 고려할 때 매우 효율적임을 보였다. 32 비트인 경우, 버저코드 기반 TSC 에 비하여 SEDC 기반 TSC 을 사용하였을 때, 하드웨어 자원 사용량이 67% 감소하였고, 지연시간은 81% 향상되었음을 보였다. 추가적으로, 하드 오류를 완화하기 위해 FPGA 의 재구성 기능을 활용할 수 있음도 보였다.

I. Introduction

A. Research Motivation

System reliability has become a major concern as the transistor size decreases [1]. The consequence of increasing complexity in the functionality of applications accelerated the demand of more reliable system. On the other hand, people don't want to return back to less sophisticated systems due to the grown dependence on luxuries automated systems.

Initially, reliable computing was limited to military, industrial, aerospace, and communications applications in which the outcome of computer failure had major economic impact or even loss of life. Nowadays, even commercial and day to day life applications require high reliability as we move towards the era of wired money transfer and automated life-style. Reliability is of vital importance in situations where a computer failure could have disastrous results [2].

Errors can be classified as either soft errors or hard errors. Soft errors are caused by transient or intermittent faults, while hard errors are caused by permanent faults. Permanent faults remain for indefinite periods until corrective action is taken. Studies show that the majority of errors are caused by transient faults.

Studies shows that most of the errors originate from Arithmetic logic unit (ALU) of a microprocessor based system [3], which is used in almost every automation application. From space applications to a simple money transfer, error could cause disaster. For example, a permanent fault in ALU calculation for navigational data may result in the lost of a ten billion dollar shuttle in space for ever. During a money transfer, a single flip of bit in the transferring amount can cause huge deficit to an individual.

Hence a fault tolerant ALU has become the most important part of such applications.

B. Research Objectives

Many error detection techniques have been proposed so far. Usually, a tradeoff is made between the performance of a processor and the area and power required for error detection. For efficient implementation of CED techniques, it is important to consider the relevant types of faults that are supposed to be more probable to occur. 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 [4]. Unidirectional errors can alter the node logic from zero to one or from one to zero, but not both at the same time. So an All Unidirectional Error Detection (AUED) technique provides optimal fault coverage with reduced area overhead.

Our main objective is to reduce the area penalty with 100% fault coverage against unidirectional errors. Delay between occurrence and detection of fault can also play a vital role. If a fault is detected after it is being propagated, then the overall system might fail, hence delay is also one of the key objectives of our research. We will also take in to account the power distribution so as to avoid the hot spots in the design. Lastly, our focus will be on reducing the complexity of overall system when scaling the circuits for higher input data lengths.

C. Thesis Contributions

The main aim of this research is to come up with a fault tolerant ALU using newly developed AUED technique named Scalable Error Detecting (SEDC) scheme with a better hardware and delay overhead as compared to the previous CED techniques. In order to achieve this, we will formulate and design the SEDC encoded ALU for predicting the SEDC code word of a particular ALU output word. As the name employs, SEDC is scalable with respect to input data length, while the latency of the circuit remains constant. SEDC splits the input data into smaller segments (2-, 3- and 4-bits) and encodes them in parallel (using $SEDC_2$, $SEDC_3$ and $SEDC_4$ coding schemes respectively). This inherited parallelism makes our scheme faster. Moreover, the scaling requires few modifications in the basic circuit resulting in less complex structure.

SEDC scheme uses four basic coding schemes, namely $SEDC_2$, $SEDC_3$ and $SEDC_4$. For bigger input data length, multiple copies of these basic coding schemes are used. Hence, the power distribution of overall circuit is very uniform.

We will also present the design and implementation of Totally Self-checking (TSC) checker for SEDC scheme. This checker also exploits the parallelism concept like the code generating part, hence the SEDC checker also exhibit constant latency, no matter how long the input data length is.

The prototype of a complete 8-bit error detecting and reconfigurable ALU system will also be illustrated on FPGA.

Result shows that SEDC based ALU outperforms other coding schemes in terms of delay, while it takes less area than Berger Code Prediction ALU [5] which is known to be the only coding scheme used for implementing fault secure ALU [6].

D. Thesis Organization

The rest of this thesis is organized as follows: the overview of previous work related to this topic has been presented in chapter II. Theoretical background of SEDC scheme is given in chapter III and the overall block diagram of SEDC based self-checking ALU is discussed in chapter IV. With the help of logic equations, first the method of encoding 2-, 3- and 4-bit Boolean Logic, Shift/Rotate and Add/Subtract operation units using SEDC scheme is illustrated in chapter V and then, the scaling of ALU for any input bit length 'n' is elaborated in chapter VI. The design details of TSC SEDC checker is covered in chapter VII. We will discuss the fault coverage of SEDC based self-checking ALU and the TSC SEDC checker in chapter VIII & IX respectively. Chapter X is dedicated for comparing the area, delay and complexity of SEDC scheme with existing self-checking ALU techniques. The FPGA implementation of SEDC based error detecting ALU is discussed and evaluated in Chapter XI. Finally, the conclusion with future goals are discussed in last chapter.

II. Overview and Related Work

Self-checking ALU provides concurrent error detection (CED) capability that can be used to design a fault tolerant computer system. A duplex structure (or double modular redundancy, i.e., DMR) provides high fault security but require twice the area overhead as compared to the area of a simple ALU. Checker in DMR system accommodate exactly twice the number of check bits which increases the area overhead of the checker as well. For example, if a 64-bit fault tolerant ALU is implemented using DMR technique then the checker for DMR system must encompass 128-bit (or more). For system to be TSC, the checker must be TSC as well. A 128-bit TSC checker must contain tree structure of two rail checkers that not only increases the overall area of the system, but also the delay as well. Hence, systems with larger inputs are proposed not to be protected by DMR technique [7].

Fault tolerance by shifted and rotated operands in TMR [8] is proposed for high fault security, but the technique requires three copies of same ALU. Moreover, the shifting and rotating operations slow down the whole process, and hence limits the overall speed of the microprocessor. The outputs of the three ALU modules have to be checked using a voter circuitry, which produces erroneous outputs when two copies of any module in the system fail. The voter circuitry has to cover 3 times the numbers of outputs than the outputs of a single ALU. For a 128-bit TMR ALU, the voter circuit has to accommodate 384-bit inputs which tremendously increases the overall area overhead of the system.

In [9], hardware and time redundancy are combined to achieve fault detection, diagnosis as well as isolation of the faulty module with 75% more area overhead than a normal ALU. The technique is based on the fact that a 32-bit ALU can be implemented using two 16-bit ALUs. If one of the ALU struck by a permanent fault, the other 16-bit ALU can compute the 32-bit result with some degradation in system performance. For the system to work as TSC, the system must employ TSC multiplexers (i.e., differential multiplexers) for multiplexing the outputs and inputs of the smaller ALUs, which adds to area overhead. The control unit must also exhibit self-testing properties, otherwise it will become the single point of failure to the system.

For efficient implementation of CED techniques, it is important to consider the relevant types of faults that are more likely to occur. 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 an unidirectional manner [10]. Unidirectional errors can alter the node logic from zero to one or from one to zero, but not both at the same time [4],[10],[11]. All Unidirectional Error Detection (AUED) techniques provide optimal fault coverage with reduced area overhead as compared to simple duplication (DMR) or triplication (TMR).

Several error detection schemes have been proposed to detect unidirectional errors in computer hardware. Berger Code scheme is the most popular AUED scheme. In [5], Berger Check Prediction (BCP) circuit is proposed for detecting unidirectional errors in the ALU circuit. The BCP circuit generates the check symbol for the n-bit ALU result using the zero's counts of the operands as well as the internally generated carries. Although the use of internally generated carries for computation of check bits makes the scheme strongly fault secure (SFS), but this also increases the latency of the overall system. Latency of Berger code checker also adds into this delay. Results in [12] show that FPGA implementation of BCP ALU requires 45% more area overhead than the area occupied by normal ALU.

Arithmetic codes like Residual codes are efficient for checking arithmetic units because these codes reside under most arithmetic operations [13]. Arithmetic codes can ensure fault secureness for most arithmetic operators [14], but arithmetic code checking has some drawbacks. Logic and shift operations do not preserve arithmetic codes. Therefore, using such codes in ALU and shifter requires the implementation of complex circuitry. Also, arithmetic codes don't provide 100% fault coverage against all unidirectional errors. An efficient fault tolerant ALU using residual codes [15] for whole data path is proposed, but if only ALU is considered then this technique occupies the same area as DMR. The circuit also uses internal carry vectors for CED which makes it slower than the DMR technique.

For applications where only t-unidirectional errors are required to be detected, [16] proposes a modification of BCP ALU using Bose-Lin codes, with less area overhead than [5]. Similar to BCP circuitry , the Bose-Lin Check Prediction circuitry also uses input operands and internally generated carries from normal ALU to generate the check symbols. Hence, the worst case time response of [5] and [16] is almost the same.

No scaling scheme is given in [5] while Bose-Lin based Check Prediction circuitry is not easily scalable. As the check bits increase above three bits, the arithmetic circuits require complete change in some parts of the circuitry.

In [17] error correcting codes are proposed for designing 32-bit fault tolerant ALU. The scheme uses BCH codes for detecting and correcting 5-bit error in any position of its 32-bits input register. Each encoder and decoder takes 63 clock cycles to compute the result that introduces more delay to the circuit. Faults within the ALU cannot be detected by this scheme.

Area and time efficient self-checking adders have been proposed recently using two rail codes [18], but the circuits can't be used in ALU because the shifter and logic unit have to be encoded using different codes, and thus require different checkers as well. This not only increases the complexity of the system, but also the cost as well.

In this thesis we present an AUED method for detecting errors in ALU using Scalable Error Detecting Codes (SEDC) [19]. Unlike BCP [5], this scheme generates the check bits without using the internally generated carries which is the main reason why this scheme is more area and delay efficient than the BCP scheme. With 100% fault coverage against all unidirectional errors, SEDC scheme also provides 82% fault coverage against all other errors that emerge due to single faults in the ALU. SEDC is easily scalable for any number of input bits 'n' while latency of SEDC encoded ALU remains constant. Unlike residual codes, here we use only one type of error detection scheme i.e., SEDC scheme for entire design of ALU, that also simplifies the overall chip design. The area and delay efficient, scalable TSC checker for SEDC scheme is also presented in this paper which further reduce the area of overall system. Although, our coding scheme have bigger code length than Berger codes, but the area efficient TSC SEDC checker requires less area than the TSC Berger checker. The prototype SEDC based ALU system is implemented on FPGA platform which exploits its reconfiguration feature to mitigate permanent errors.

III. Scalable Error Detection Coding Scheme

Scalable Error Detection Coding scheme [19] is formulated and designed in such a way that only area is scaled, while latency depends on a small portion of the input data (explained later).

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

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

Satisfying the condition for parameter 'a', the maximum possible value for parameter 'b' is selected. The length of SEDC code C represented as ($C_{m-1}, \ldots, C_j, \ldots, C_2, C_1, C_0$) with $C_j \in \{0, 1\}$ for $0 \leq j \leq m-1$, is then computed as per (2).

$$m = \lceil log_2(n + 1 - 3a) \rceil + 2a \qquad (2)$$

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 2-, 3-, and 4-bit data and accordingly referred to as $SEDC_2$, $SEDC_3$ and $SEDC_4$ scheme, respectively. It is then extended for any integer values of n, as shown in Fig. 3.1.

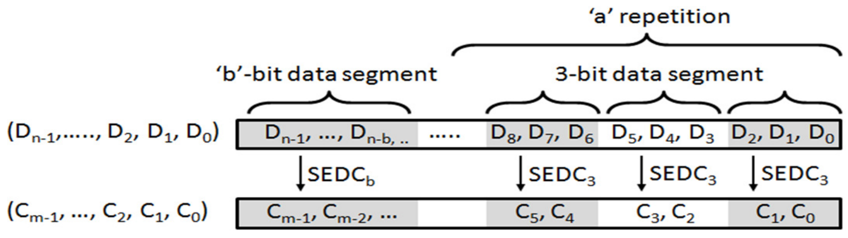


Figure 3.1 Data partitioning and encoding using SEDC scheme for given data word

Next, we will discuss the mathematical foundations of $SEDC_2$, $SEDC_3$ and $SEDC_4$ schemes with logical explanations about their error detecting capabilities.

A. $SEDC_2$ code

Fig. 3.2 gives a 2-D square illustration of $SEDC_2$ scheme where nodes represent data words and their corresponding code words are written in brackets.

The SEDC coding scheme assigns code words to different data words with a unique criteria. Whenever there is a change of bit (or bits) in data word from '1' → '0' (shown with bold arrow in Fig. 3.2), the change is reflected on code word in opposite way, i.e., the code changes from '0' → '1'(shown with dashed arrow in Fig. 3.2), and vice versa. In general, when the weight of data word increases, the weight of its SEDC code word decreases and vice versa. Equation (3) is used to assign 2-bit code words 'C_1C_0' to the 2-bit data words D_1D_0. C_1C_0 can also be used interchangeably; this results in another variant of $SEDC_2$ code.

$$[C_1 : C_0] = [\text{NAND}(D_1, D_0) : \text{XNOR}(D_1, D_0)] \qquad (3)$$

$SEDC_2$ is the basic coding scheme and is embedded in $SEDC_3$ and $SEDC_4$ to detect all unidirectional errors in 3-bit and 4-bit data, as shown later. This ability of scaling codes is not present in any other coding scheme.

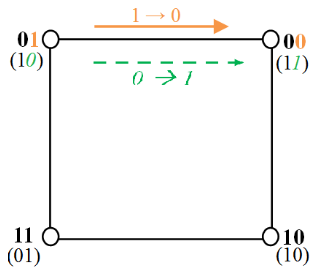


Figure 3.2 2D illustration of $SEDC_2$ scheme

B. $SEDC_3$ code

$SEDC_3$ code for 3-bit data is computed as per (4).

$$(C_1, C_0) = \begin{cases} SEDC_2(D_1, D_0), & \text{if } D_2 = 0 \\ \text{1's complement}\left(SEDC_2\left(\overline{D_1}, \overline{D_0}\right)\right), & \text{if } D_2 = 1 \end{cases} \qquad (4)$$

Fig. 3.3.(a) shows a 3-D cube illustrating the unidirectional error detection mechanism of $SEDC_3$ codes. Same notations are used in Fig. 3.3.(a) as in Fig. 3.2. The dashed side of the cube represents 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 changes in the opposite direction (MSB of the code changes from '1' to '0').

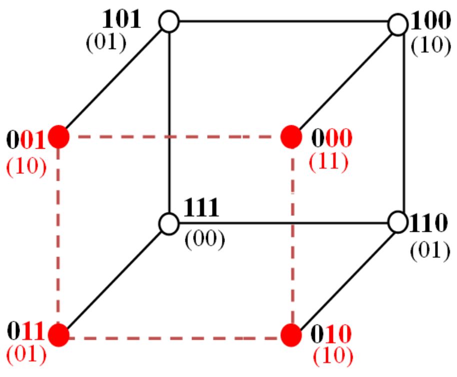
The first four code words for $SEDC_3$ are same as $SEDC_2$ as shown in Fig. 3. 3(a) (with dashed side). The remaining four code words are generated by the following steps:

1) Invert all the 3-bit data bits (we take '100'→'011').

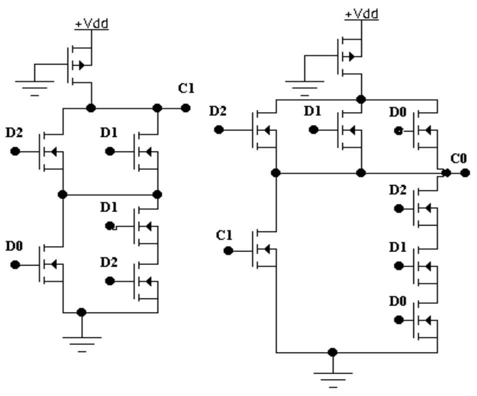
2) Find the $SEDC_2$ code word corresponding to the inverted data resulting from step 1 ('011'→'01').

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.



(a)



(b)

Figure 3.3 (a) 3D illustration of $SEDC_3$ scheme (b) $SEDC_3$ circuit

Any two data words D_A and D_B such that D_B can be converted to D_A by just changing the 1's → 0's or 0's → 1's, are assigned unique code word by above shown steps, hence making it possible to detect all unidirectional errors.

C. $SEDC_4$ code

$SEDC_4$ code for 4-bit data is formulated as per (5).

$$[C_2 : (C_1, C_0)] = [NOT\ (D_3) : SEDC_3 (D_2, D_1, D_0)] \qquad (5)$$

MSB of the code word is completely dependent upon MSB of the data word for $SEDC_4$; hence any change in the MSB of the data word is detected. While the rest of the three data bits are encoded using same $SEDC_3$ scheme.

In general, for $SEDC_n$, the n-bit binary data is grouped into one ‘b’-bit segment and ‘a’ number of 3-bit segments, and then these segments are encoded using $SEDC_b$ and ‘a’ number of $SEDC_3$ modules in parallel, as shown in Fig. 3.1. Small code words are produced from 3-bit and b-bit data segments for error detection. It is noteworthy that each group of data segment and corresponding code segment is independent to each other. This independency makes our scheme scalable.

We could partition to get Berger codes in parallel; however it requires more area than SEDC scheme. For instance, a 3-bit input SEDC circuit shown in Fig. 3.3.(b) can be implemented with 14 MOS transistors, while 3-input one's counter is implemented with a full adder that contains 28 MOS transistors [20].

IV. Introduction to Overall System

The general block diagram of SEDC based error detecting ALU is depicted in Fig. 4.1. [19]. SEDC is a concurrent error detection (CED) scheme which requires a separate encoded module, like the one proposed in [21]. Inputs are simultaneously applied to the functional unit (in our case ALU) and output characteristic predictor circuit (in our case SEDC encoded ALU or SEDC check bits predictor). TSC SEDC checker validates the output of both ALU and SEDC encoded ALU.

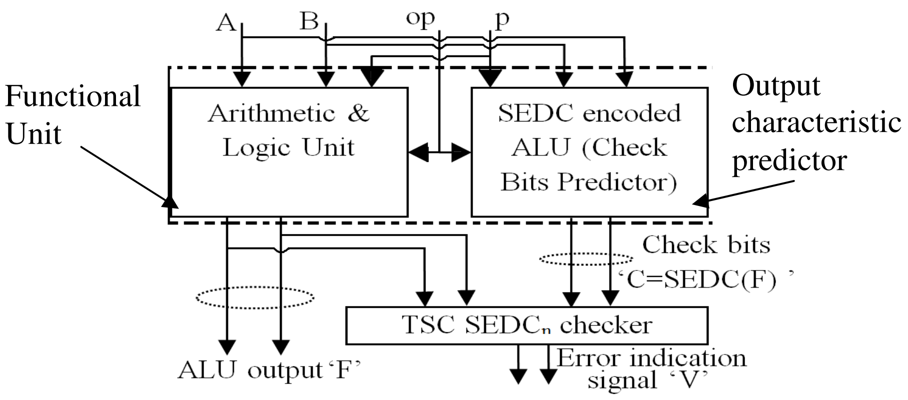


Figure 4.1 General model for SEDC based Self-checking ALU

A. Arithmetic and Logic Unit

The ALU generates the normal output 'F(A, B, op, p)' by taking the inputs 'A', 'B', 'op' and 'p'. The ALU circuit is capable of performing four basic operations, namely Boolean Operation of two n-bit operands, n-bit Shift/Rotate Operation, n-bit Add/Subtract Operation and n-bit Compare Operation (abbreviated as BO, SRO, ASO and CO respectively throughout the thesis).

Input 'op' is designated to specify particular Arithmetic or Logic Operation (like Add/Sub, Shift or any Boolean Operation). To provide the input shift bit 's_i' for shift operation, or carry input 'c_{in}' for Add/Subtract operation, 1-bit input 'p' is used.

B. SEDC encoded ALU

SEDC encoded ALU or SEDC check bits predictor is designed in such a way that it generates the corresponding SEDC check bits ‘C = SEDC(F)’ using the same inputs ‘A’, ‘B’, ‘p’ and ‘op’, in parallel. For designing this SEDC encoded ALU, we generated truth tables with inputs A, B, op and p, while the output equals to ‘C = SEDC (F)’, using Logic Friday software.

Similar to the ALU, SEDC encoded ALU comprises of SEDC encoded BO Unit, SEDC encoded SRO Unit, SEDC encoded ASO Unit and SEDC encoded Compare Unit. To switch between different operations, input ‘op’ is used. One can observe from Fig. 4.1, that unlike BCP ALU [5], SEDC encoded ALU does not use the internally generated carries.

C. TSC SEDC checker

A totally self-checking (TSC) SEDC checker is used to validate the ALU output ‘F’ with its corresponding SEDC encoded ALU output ‘C’. A 2-bit error signal ‘V’ is generated if ‘F’ and ‘C’ do not correspond to each other (i.e., $C \neq SEDC(F)$). As the checker is TSC, hence it has the ability to detect or safely hide its own error. The checker also exhibits scalability, i.e., the checker can be designed for any number of input bits with little increment in hardware while the latency remains constant.

V. Formulation of SEDC encoded ALU

In this chapter we will discuss the formulation of each SEDC encoded ALU operation (Boolean, shift/rotate and add/subtract operation, compare operation with some examples).

A. 2-, 3- and 4-bit SEDC encoded Boolean Operation Unit

In Table 5-1 [19], we list the logic equations to implement 2-bit SEDC encoded BO Unit. The symbol ‘opBO’ is equivalent to ‘op’ in Fig. 4.1. A total of 16 Boolean logic operations can be performed by changing input ‘opBO’.

Table 5-1. Logic for 2-bit SEDC encoded BO Unit

Operations	opBO	2-bit SEDC Encoded Boolean Operations	A_3/ A_2	B_3/ B_2	S_9/ S_{10}
		$(C_{1(2BIT)}, C_{0(2BIT)}) = SEDC_2\ (A, B, \text{OPBO})$			
Logic 0	0	$SEDC_2(00)$	X	X	0
A NOR B	1	$SEDC_2(\bar{A}) + SEDC_2(\bar{B})$ - $SEDC_2(\bar{A}$ OR $\bar{B})$	0	0	1
$\bar{A}$ AND B	2	$SEDC_2(\bar{A}) + SEDC_2(B)$ - $SEDC_2(\bar{A}$ OR B)	0	1	1
$\bar{A}$	3	$SEDC_2(\bar{A})$	0	X	1
A AND $\bar{B}$	4	$SEDC_2(A) + SEDC_2(\bar{B})$ - $SEDC_2(\text{AOR}\bar{B})$	1	0	1
$\bar{B}$	5	$SEDC_2(\bar{B})$	X	0	1
A XOR B	6	$SEDC_2(A) + SEDC_2(B) - (2\times SEDC_2(A.B)) + 3$	0 1	1 0	1 1
A NAND B	7	$SEDC_2(\bar{A}) + SEDC_2(\bar{B}) - SEDC_2(\bar{A}.\bar{B})$	1	1	0
A AND B	8	$SEDC_2(A) + SEDC_2(B)$ - $SEDC_2$(A OR B)	1	1	1
A XNOR B	9	$SEDC_2(\bar{A}) + SEDC_2(B) - (2 \times SEDC_2(\bar{A}\ .\ B)) + 3$	0 1	0 1	1 1
B	10	$SEDC_2(B)$	X	1	1
$\bar{A}$ OR B	11	$SEDC_2(\bar{A}) + SEDC_2(B) - SEDC_2(\bar{A}\ .\ B)$	1	0	0
A	12	$SEDC_2(A)$	1	X	1
A OR $\bar{B}$	13	$SEDC_2(A) + SEDC_2(\bar{B}) - SEDC_2(A\ .\ \bar{B})$	0	1	0
A OR B	14	$SEDC_2(A) + SEDC_2(B) - SEDC_2(A\ .\ B)$	0	0	0
Logic 1	15	$SEDC_2(11)$	X	X	1

The ‘$SEDC_2$’ symbol mentioned in Table 5-1 generate the $SEDC_2$ check bits ‘$C_{1(2bit)}C_{0(2bit)}$’ of the 2-bit input operands ‘A’ and ‘B’. ‘2 x $SEDC_2$’ denote the one bit left shifted result after taking $SEDC_2$ of the operands, with ‘0’ as the input shift bit. Other symbols like ‘+’ and ‘-’ represent the normal add and subtract operations. “$\bar{A}$” shows the inverted A, while ‘AND’, ‘OR’, ‘XOR’, ‘NAND’ and ‘NOR’ are the normal 2 operand logic gate operations. For example, if the value of ‘opBO’ is 6, then the output of the ALU is normal XOR value of ‘A’ and ‘B’ while output of the SEDC encoded ALU is computed by the corresponding equation from Table 5-1. Unlike BCP ALU [5], Table 5-1 don’t contain any signal (Carry out, internally generated carries etc) that is being generated by normal ALU. This makes SEDC encoded ALU faster than BCP ALU. Following example illustrates the 2-bit SEDC encoded XOR operation.

Example 1: Let A = 01, B = 10, opBO = 0110 (XOR)

$\Rightarrow$ A AND B = 00; $SEDC_2$(A) = 10, $SEDC_2$(B) = 10, $SEDC_2$(A AND B) = 11;

$\Rightarrow$ 2x$SEDC_2$(A AND B) = 110;

$\Rightarrow$ $SEDC_2$(A) + $SEDC_2$(B) - {2x$SEDC_2$(A AND B)} + 3= 001

Discarding the MSB, the remaining two LSBs ‘01’ are the $SEDC_2$ check bits for XOR operation between ‘A’ and ‘B’, which can be verified from (3).

For scaling 2-bit SEDC encoded BO unit to 3-bit, ‘opBO’, ‘A_2’ and ‘B_2’ signals are encoded to generate ‘S_9’, using Table 5-1 (where ‘A_2’ and ‘B_2’ are the MSB’s of the 3-bit input operands ‘A’ and ‘B’ respectively). Replacing ‘S_Q’ with ‘S_9’ in (6), will yield the $SEDC_3$ check bits $C_{1(3bit)}C_{0(3bit)}$.

$$(C_{1(3bit)}, C_{0(3bit)}) = \begin{cases} (C_{1(2bit)}, C_{0(2bit)}) - 1 & \text{if } S_Q = 1 \\ (C_{1(2bit)}, C_{0(2bit)}) & \text{otherwise} \end{cases} \quad (6)$$

Similarly, (7) & (8) are used for scaling 3-bit SEDC encoded BO unit to 4-bit unit. Here signals ‘opBO’, ‘A_3’ and ‘B_3’ are encoded to generate ‘S_{10}’ using Table 5-1, and depending upon ‘S_{10}’, the output $SEDC_4$ code $C_{2(4bit)}C_{1(4bit)}C_{0(4bit)}$ is obtained.

$$C_{2(4bit)} = \begin{cases} 1 & \text{if } S_{10} = 0 \\ 0 & \text{otherwise} \end{cases} \quad (7)$$

$$(C_{1(4bit)}, C_{0(4bit)}) = SEDC_3(A_{(3bit)}, B_{(3bit)}, opBO) \quad (8)$$

B. 2-, 3- and 4-bit SEDC encoded Shift/Rotate Unit

The SEDC check of the result of a shift/rotate operation is simply the SEDC check of the operand, since no information bit is discarded except for their position. The logic shift operation involves the 's_i' bit (equivalent to 'p' in Fig. 4.1) as input shift bit. Table 5-2 tabulates the logic of 2-bit SRO unit [19].

Table 5-2. Logic for 2-bit SEDC encoded SRO Unit

Operations	opSR	2-bit SEDC Shift/Rotate Operations
		$(C_{1(2bit)},C_{0(2bit)}) = SEDC_2(A(A_1, A_0), s_i, opSR)$
Shift Left A	00	$SEDC_2([A_0\ s_i])$
Shift Right A	01	$SEDC_2([s_i\ A_1])$
Rotate Left A	10	$SEDC_2(A)$
Rotate Right A	11	$SEDC_2(A)$

The circuit is able to perform same operations given in [5] that are, shift left, shift right, rotate left and rotate right on any 2-bit operand 'A' and generate its 2-bit SEDC code. Following is the example to illustrate the 2-bit SEDC encoded shift/rotate operation.

Example 2: Let A = 01, s_i = 1, opSR = 01 (shift right);

$\Rightarrow$ $[s_i\ A_1] = 10$; (normal shift output)

$\Rightarrow$ $SEDC_2([s_i\ A_1]) = 10$ (SEDC encoded output)

For scaling 2-bit SEDC encoded SRO unit to 3-bit unit, we replaced S_Q with S_1 in (6), while for scaling 3-bit SRO unit to 4-bit unit we used (9) & (10). Signal 'sibit' and S_1 are generated using primary inputs opSR, A and si, whose logic equations are given in Table 5-2.

$$(C_{1(4bit)}, C_{0(4bit)}) = SEDC_3(A(A_2,A_1,A_0), \text{sibit}, opSR) \quad (9)$$

$$C_{2(4bit)} = \begin{cases} \text{NOT } A_2, \text{if opSR} = 0 \\ \text{NOT si}, \text{if opSR} = 1 \\ \text{NOT } A_2, \text{if opSR} = 2 \\ \text{NOT } A_0, \text{if opSR} = 3 \end{cases} \quad (10)$$

C. 2-, 3- and 4-bit SEDC encoded Add/Subtract Unit

The SEDC check of the result of an add/subtract operation is simply the SEDC check of the normal add/subtract result, as formulated in (11) & (12) [19].

$$C_{1(2bit)}, C_{0(2bit)} = SEDC\ \{A \oplus (B \oplus opAS) \oplus (C_{in} \oplus opAS)\} \quad (11)$$

$$C_{c(2bit)} = SEDC\{A \oplus (B \oplus opAS)\}(C_{in} \oplus opAS) + A(B \oplus opAS) \quad (12)$$

The ‘opAS’ here corresponds to the input ‘op’, and ‘C_{in}’ corresponds to ‘p’ in Fig. 4.1. $C_{c(2bit)}$ and $C_{1(2bit)}C_{0(2bit)}$ are SEDC check bits for output carry and sum, respectively. Example 3 shows the SEDC check bit generation for 2-bit add operation.

Example 3: Let A = 01, B = 10, C_{in} = 1, opAS = 0;

$\Rightarrow$ C_{out} = 1, A + B + C_{in} = Sum = 00;

$\Rightarrow$ *$SEDC_2$* C_{out} = 0, *$SEDC_2$* Sum = 11;

Here again, for scaling 2-bit SEDC encoded Add unit to 3-bit unit, (6) is used to generate check bits ‘$C_{1(3bit)}C_{0(3bit)}$’ corresponding to the output *sum* of the ALU, by replacing S_Q with S_{15}. The logic equations that generate check bit for carry out signal ‘$C_{c(3bit)}$’ and S_{15} signal are given in Table 5-3.

Similarly, (13) & (14) are used to generate the check bits ‘$C_{2(4bit)}C_{1(4bit)}C_{0(4bit)}$’ corresponding to output *sum* of the 4-bit SEDC encoded ALU, while the logic equations for carry out signal ‘$C_{c(4bit)}$’ and S_{17} signal are given in Table 5-3.

$$C_{2(4bit)} = \begin{cases} \text{NOT } C_{\text{c(2bit)}}, & \text{if } S_{17} = 1 \\ C_{\text{c(2bit)}}, & \text{otherwise} \end{cases} \tag{13}$$

$$(C_{1(4bit)}, C_{0(4bit)}) = (A_{(3bit)}, B_{(3bit)}, C_{in,} \text{opAS}) \tag{14}$$

The three SEDC encoded operation units compute the SEDC check bits independent of the internally generated carries from the normal ALU, which reduce the overall latency of the system.

Table 5-3. Logic equations for S_1, sibit, S_{14}-S_{15} signals

$S_1 = (opSR_1')(opSR_0)(A_2) + (opSR_1')(opSR_0')(A_1)$
$sibit = (opSR_0)(A_3) + (opSR_0')(s_i)$
$S_{14} = C_{c(3bit)} = (B_2')(C_{c(2bit)}) + (A_2')(B_2')(C_{c(2bit)}') + (A_2')(B_2)(C_{c(2bit)})$
$S_{15} = (A_2)(B_2)(C_{c(2bit)}') + (A_2)(B_2')(C_{c(2bit)}) + (A_2')(B_2')(C_{c(2bit)}') + (A_2')(B_2)(C_{c(2bit)})$
$S_{16} = C_{c(4bit)} = (A_3')(B_3') + (A_3')(C_{c(3bit)}) + (B_3')(C_{c(3bit)})$
$S_{17} = A_3' B_3 + A_3 B_3'$

Equations in Table 5-3 [19] are formulated by inputting the logic tables in Logic Friday software and then minimizing them to generate logic equations. All the logic equations uses primary inputs ‘A’, ‘B’, ‘p’ and ‘op’ to compute the SEDC check bits of a particular Arithmetic and Logic operation.

All the circuits are implemented using combinational logic. These circuits can also work for 1-bit input data. The only change to made is; take the LSB of the SEDC Code (C_0) rather than taking both C_1C_0 bits.

D. 2-, 3- and 4-bit SEDC encoded Compare Unit

The Compare unit takes in only two operands A & B, and can perform three operations, that are, A greater than B, A is equal to B, and A is smaller than B. Particular operation can be selected using the input op. Table 5-4 enlists the $SEDC_2$ code of the two input compare operation.

Table 5-4. Logic for 2-bit SEDC encoded Compare Unit

Operations	**opCC**	**2-bit SEDC Compare Operations**
		$(C_{1(2bit)}, C_{0(2bit)}) = SEDC_2(A(A_1, A_0), B(B_1, B_0), opCC)$
$A > B$	00	$SEDC_2(00)$
$A == B$	01	$SEDC_2(01)$
$A < B$	11	$SEDC_2(11)$

For scaling 2-bit unit to 3- (e = 2) and 4-bit (e = 3) unit, equation (15) is generally used.

$$[C_{1(e+1\text{-bit})}, C_{0(e+1\text{-bit})}] = [\{(A_e + C_{1(e\text{-bit})}).\overline{B_e} + A_e.C_{1(e\text{-bit})}\}, \{(A_e \oplus B_e) + C_{0(e\text{-bit})}\}] \quad (15)$$

VI. Scaling SEDC encoded ALU for n-bit Input

As discussed in chapter II, SEDC scheme is made for 2-, 3- and 4-bit inputs. For n-bit input, combination of these 2-, 3- and 4-bit input schemes are used, denoted as $SEDC_2$, $SEDC_3$ and $SEDC_4$ respectively. Following we give logic and examples for designing n-bit SEDC encoded BO, SRO and ASO units using their respective 2-, 3- and 4-bit modules.

A. n-bit SEDC encoded Add/Subtract Unit

The $SEDC_n$ check bits (C_m) have two parts; $\{C_{m-1}, C_{m-2},\ldots,C_{m-L}\}$ generated by b-bit SEDC encoded BO with input operands $\{A_{n-1}, A_{n-2},.., A_{n-b}\}$, $\{B_{n-1}, B_{n-2},.., B_{n-b}\}$, opBO, and $\{C_{m-(2xk)-L-1}, C_{m-(2xk)-L-2}\}$ generated by sets of 3-bit SEDC encoded BO on inputs $\{A_{nn-1}, A_{nn-2}, A_{nn-3}\}$, $\{B_{nn-1}, B_{nn-2}, B_{nn-3}\}$, opBO. For calculating values of nn, k and L, (16)-(18) [19] can be used.

$$nn = n - (3k) - b \qquad (16)$$

$$k = 0\ to\ (a - 1) \qquad (17)$$

$$L = [log_2\{n + 1 - (\,3a)\}] \qquad (18)$$

Example 5: Let A and B contains 8-bit each;

$\Rightarrow$ b = 2, a = 2 and m = 6 {from (1) & (2)}

$\Rightarrow$ k = 0 to 1, nn = 6, 5 and L = 2. {from (16)-(18)}

So we generate (C_5, C_4) using one 2-bit SEDC encoded BO unit ($SEDC_2$) with inputs (A_7, A_6), (B_7, B_6) while (C_3, C_2), (C_2, C_0) are generated using two 3-bit SEDC encoded BO units ($SEDC_3$) with inputs (A_5, A_4, A_3), (B_5, B_4, B_3) & (A_2, A_1, A_0), (B_2, B_1, B_0) respectively. Input 'opBO' being the common input to all of the three SEDC encoded modules.

B. n-bit SEDC encoded Add/Subtract Unit

Similar to n-bit BO operation unit, the logic to implement n-bit SEDC encoded SRO unit is given in Table 6-1 [19]. Again the modules are split into b-bit and 3-bit modules for generating the SEDC code. For 8-bit SEDC encoded SRO unit, values of a, b, m, k and L remains same as in Example 5. Check bits (C_5, C_4) are generated by 2-bit SEDC encoded SRO unit using (A_7, A_6), (B_7, B_6) and $sibit_1$ as inputs while (C_3, C_2) and (C_1, C_0) are generated by two 3-bit SEDC encoded SRO units with inputs (A_5, A_4, A_3), (B_5, B_4, B_3), $sibit_2$ and (A_2, A_1, A_0), (B_2, B_1, B_0), sibit$_2$' respectively. The sibit$_2$' is computed the same way as sibit$_2$ but with changed values of 'k'. The logic to generate sibit$_1$ and sibit$_2$ signals is also given in Table 6-1.

Table 6-1. Logic for scaling n-bit SEDC encoded SRO unit

b-bit SEDC encoded SRO unit with inputs ((A_{n-1}, A_{n-2}, ..A_{n-b}), sibit$_1$, opSR)			
Operations		***opSR***	***sibit$_1$***
Shift/Rotate Left A		X0	A_{n-b-1}
Shift Right A		01	s_i
Rotate Right A		11	A_0
3-bit SEDC encoded SRO unit with inputs ((A_{nn-1}, A_{nn-2}, A_{nn-3}), sibit$_2$, opSR)			
Operations	***opSR***	***k***	***sibit$_2$***
Shift Left A	00	= (a-1)	s_i
		≠ (a-1)	$A_{n-b-(3x(k+1)-1}$
Shift/Rotate Right A	01	X	$A_{n-b-(3xk)}$
Rotate Left A	10	= (a-1)	A_{msb}
		≠ (a-1)	$A_{n-b-(3x(k+1))-1}$

C. n-bit SEDC encoded Add/Subtract Unit

Table 6-2 [19] list the logic to design n-bit SEDC encoded ASO unit. Care must be taken to connect carry in and carry out signals between a b-bit and 3-bit SEDC encoded ASO units. Also there is a requirement of two extra XOR gates for inverting B and C_{in} bits (for subtraction operation).

Table 6-2. Logic for scaling n-bit SEDC encoded ASO unit

b-bit SEDC encoded ASO unit Inputs = ((A_{n-1}, A_{n-2}, …., A_{n-b}), (B_{n-1}, B_{n-2}, …., B_{n-b}), C_{in1}, opAS)				
Operations	***opAS***	***a***	***C_{in1}***	***B***
$A + B + C_{in}$	0	= 0	C_{in}	B
		≠ 0	C_k	
$A – B - C_{in}$	1	0	$\overline{Cin}$	$\bar{B}$
		≠ 0	$\overline{Ck}$	
(C_K, C_{Sm-L-1}, …., $C_{Sm-(2xL)-L-2}$ = 3-bit SEDC encoded ASO unit Inputs = ((A_{nn-1}, A_{nn-2}, …, A_0), (B_{nn-1}, B_{nn-2}, …, B_0), C_{in2}, opAS)				
Operations	***opAS***	***k***	***C_{in2}***	***B***
$A + B + C_{in}$	0	= (a-1)	C_{in}	B
		≠ (a-1)	C_{k+1}	
$A – B - C_{in}$	1	= (a-1)	$\overline{Cin}$	$\bar{B}$
		≠ (a-1)	$\overline{C_k + 1}$	

C. n-bit SEDC encoded Compare Unit

Equation (16)-(18) can be used in similar way to partition the data as described earlier. For designing n-bit SEDC encoded Compare unit, Equation (15) can be utilized. The value of ‘e’ should be equal to ‘n-1’.

VII. TSC SEDC checker

In this chapter we will discuss the basic difference between a SEDC code checker and a Berger code checker. With introduction to '10' coding, we will explain the logic equations and MOS level circuit diagrams to implement TSC SEDC checker. We will also show the scaling of TSC SEDC checker to any number of input bits ‘n’ using the basic 2-, 3- and 4-bit TSC SEDC checkers.

A. Difference between SEDC and Berger code checker

Close inspection of Fig. 7.1.(a) and Fig.7.1.(b) reveals that Berger code checker differ SEDC checker by two modules namely; the check bit complement generator which generates the bit by bit complement of the check bits and a tree of two rail checker [7]. As the complement generator generates the bit by bit complement, in other words, Berger checker encodes the n-bit ‘F’ and m-bit pre-computed check bits into two rail codes. The two rail codes are then checked using tree structure [22]. If we partition the Berger codes in a similar way as we do with SEDC, then this two rail code length will increase, causing the depth of two rail checker tree to increase.

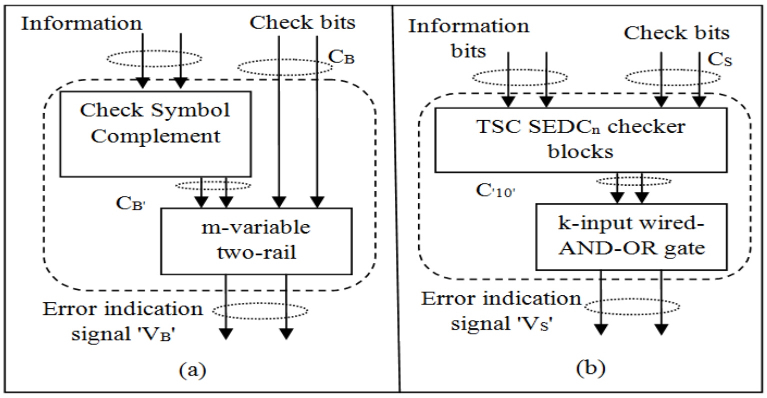


Figure 7.1 Architectures of TSC (a) Berger checker (b) SEDC checker

On the other hand, SEDC checker has distinct modules as shown in Fig.7.1.(b): the SEDC$_n$ checker and the wired AND-OR logic block. The SEDC$_n$ checker encodes the n-bit ‘F’ and m-bit ‘C’ into ‘10’ codes. Table 7-1 [23]shows the ‘10’ coding scheme for 1-information bit ‘F_0’

and 1-check bit ‘C_0’, according to which the correct output code space is V_1V_0 = {10} only (that is why we named this encoding scheme as ‘10’ encoding), while in two rail encoding the correct code space is V_1V_0 = {01, 10}. The benefit of ‘10’ codes is that they can be checked using wired AND-OR circuits in parallel, while the two rail codes can only be checked using a tree of two rail checkers, which increases the overall delay if the length of check bits increases.

Table 7-1. ‘10’ Codes Table for one bit input ($SEDC_1$)

F_0	C_0	V_1	V_0
0	0	1	1
0	1	1	0
1	0	1	0
1	1	0	0

B. Logic and circuits for TSC $SEDC_1$, $SEDC_2$, $SEDC_3$, $SEDC_4$ and $SEDC_n$ checkers

The SEDC checker is also composed of one b-bit TSC SEDC checker and a-sets of 3-bit TSC SEDC checkers. These small checkers encodes ‘F’ and ‘C’ into ‘10’ codes, rather than two rail codes. In the case of 1-, 2- and 3-bit TSC SEDC checkers, the output can be directly used as an error indication signal, but for n > 3, the outputs of smaller TSC SEDC checkers are converted to a 2-bit error signal using one level of wired-AND-OR gate. Subsections discuss the logic and circuit diagrams for primitive TSC SEDC checkers ($SEDC_1$, $SEDC_2$, $SEDC_3$ and $SEDC_4$ checkers) which can be used to scale the TSC SEDC checker to a n-bit TSC SEDC checker.

1. TSC $SEDC_1$ checker

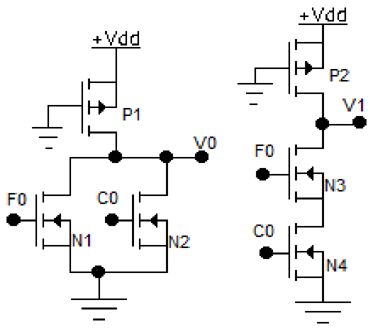


Figure 7. 2 MOS circuit for TSC $SEDC_1$ Checker

Fig. 7.2 shows the pseudo nMOS logic implementation of TSC $SEDC_1$ checker.

Table 7-1 shows the logic for 1-bit TSC SEDC (TSC SEDC$_1$ checker). Highlighted cases indicate the valid input code word (i.e., 10, 01) and the valid output code word (10). F_0 denotes the 1-bit information word which is the output of ISG 'F' and C_0 denotes 1-bit SEDC check bit generated by SEDC Check Symbol Generator (SCSG) 'C'. V_1V_0 is the 2-bit error indication signal of TSC SEDC checker 'V' [23].

2. TSC SEDC$_2$ checker

Equations (19) & (20) [23] are used to implement TSC SEDC$_2$ checker that generates the 2-bit error signal V_1V_0 (here "." & "+" denotes the normal AND & OR operations respectively). Again the correct output code space is {10}. Now C_1C_0 denotes the SEDC check bits and F_1F_0 are the information bits.

$$V_0 = \overline{(F_1 + F_0 + C_0)(C_1 + F_1F_0C_0)} \tag{19}$$

$$V_1 = \overline{C_1.(F_1 + F_0)(C_0 + F_1F_0)} \tag{20}$$

The CMOS level design of TSC SEDC$_2$ checker requires only 9 extra MOS transistors to convert 1-bit SEDC checker circuit to 2-bit SEDC checker as shown in Fig. 7.3.

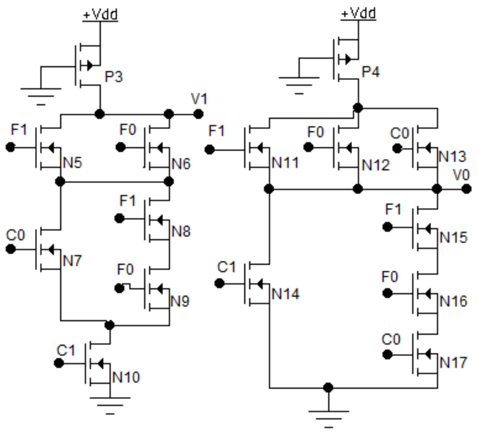


Figure 7. 3 MOS circuit for TSC SEDC$_2$ Checker

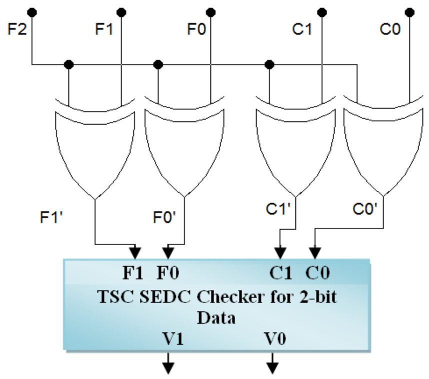


Figure 7. 4 Block diagram of 3-bit TSC SEDC checker

3. TSC SEDC$_3$ checker

Fig. 7.4 [23] shows the block diagram and the logic for 3-bit TSC SEDC checker. Three bit data $F_2F_1F_0$ from ISG and 2-bit SEDC check bits C_1C_0 from SCSG are first converted to $F_1'F_0'$

and $C_1'C_0'$ respectively and then they are checked using the same 2-bit TSC SEDC checker as shown in Fig. 7.4. When F_2 bit is '1', the F_1F_0 and C_1C_0 are inverted, while if F_2 is '0' then F_1F_0 and C_1C_0 remain same. As the outputs of XOR gates are fed to TSC $SEDC_2$ checker, hence any error in XOR gates is detected. This makes the overall 3-bit SEDC checker TSC.

4. TSC $SEDC_4$ checker

A 4-bit TSC SEDC checker consists of one TSC $SEDC_1$ checker and one TSC $SEDC_3$ checker as shown in Fig. 7.5 [23]. Both $SEDC_1$ and $SEDC_3$ checkers generate 2-bit output V_1V_0. As the valid code word is {10}, hence to make sure that both the checker units generate {10} output during error free operation, we 'AND' V_1 output-bit of TSC $SEDC_1$ checker with V_1 output-bit of TSC $SEDC_3$ checker. Also, we 'OR' V_0 output-bits of both TSC SEDC checkers using wired logic gates. We checked and confirmed by fault simulation that wired-AND and wired-OR gates are also TSC for single faults (stuck-at-0, stuck-at-1, transistor-stuck-on and transistor-stuck-off).

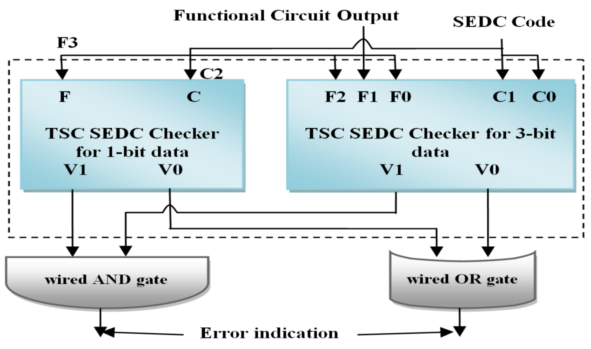

Functional Circuit Output
SEDC Code
F3
C2
F
C
TSC SEDC Checker for 1-bit data
V1
V0
F2 F1 F0
C1 C0
TSC SEDC Checker for 3-bit data
V1
V0
wired AND gate
wired OR gate
Error indication

Figure 7. 5 Block diagram of TSC $SEDC_4$ Checker

As compared to TSC Berger checker, SEDC checkers don't require tree of TSC two-rail checkers for comparison of check bits C_B with the predicted code bits C_B', as shown in Fig. 7.1. The wired-AND and wired-OR circuitry show constant latency for any number of input bits, unlike the TRC tree.

5. TSC $SEDC_n$ checker

Similar to the SEDC code generator, TSC SEDC checker requires copies of 1-, 2- and 3-bit SEDC checkers depending upon the value of "a" and "b" (evaluated from (1)). For example, if n=8 bits, then from (1), a = 2 and b = 2. This requires a TSC $SEDC_2$ checker and couple of TSC $SEDC_3$ checkers to realize an 8-bit TSC SEDC checker.

Circuit for wired-AND and wired-OR gates will also expand as “n” increases. For n = 8 bits, there will be total 3 TSC SEDC checkers with 2-bit output each. So a 3-input wired-AND and 3-input wired-OR gate is required to compare all the V_1 and V_0 bits. Fig. 7.6 [23] shows the block diagram of n-bit TSC SEDC checker.

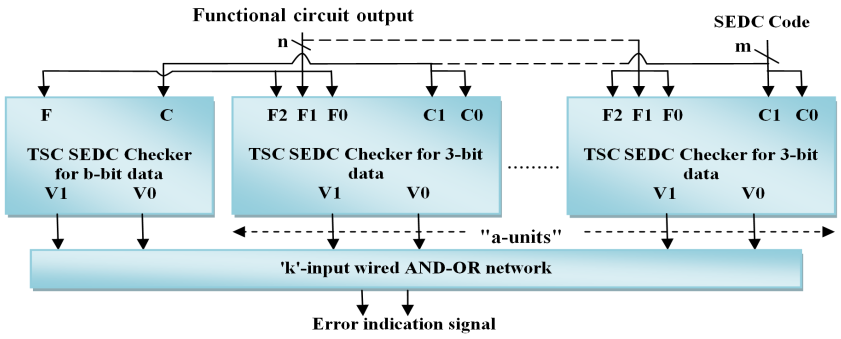

Functional circuit output
n
SEDC Code
m
F
C
TSC SEDC Checker for b-bit data
V1
V0
F2 F1 F0
C1 C0
TSC SEDC Checker for 3-bit data
V1
V0
....
F2 F1 F0
C1 C0
TSC SEDC Checker for 3-bit data
V1
V0
"a-units"
'k'-input wired AND-OR network
Error indication signal

Figure 7. 6 Block diagram of TSC $SEDC_n$ Checker

In general, for “n” bit input, there are “a+1” TSC SEDC checkers with 2-bit output each. So we require d = {2x(a+1)}-input wired-AND and wired-OR gates. With each increasing input to the wired-AND and wired-OR gates, one extra transistor is required by each of the wired-gates. Consequently, the circuit expands in width-wise fashion, and hence the latency of the wired logic remains constant for any value of “n”.

Size of the load transistor driving these wired-AND and OR gates will also increase with increasing inputs. So we consider the maximum fan in of one gate equal to 4. For d > 4, an extra load transistor is connected in parallel. If “d” denotes the total number of inputs to the TSC checker then we require $r = \lceil (k/4) \rceil$ load transistors. Total “d + r” number of transistors are required to implement the “d”-input wired AND-OR network with a constant latency of 1 transistor.

VIII. Fault Coverage of SEDC encoded ALU

A. Fault model

As discussed in Chapter III, SEDC scheme is an AUED scheme hence we assume that any single fault in ALU or SEDC encoded ALU causing a unidirectional change at their respective outputs, are detected by TSC SEDC checker. But there are some errors in ALU that result in bidirectional error at the output of ALU, and hence are not be detected by SEDC scheme. First we will explain the types of errors that can cause bi-directional change at the output of the ALU and then we will show that how much fault coverage SEDC scheme can provide against such errors.

1. *Type 1 error*

In typical ALU design, there is no distinction between the circuit that handles the arithmetic operations and that which handles the logic operations. Consequently, a single fault in carry propagation circuit may induce multiple random errors during the logic operations [5].

2. *Type 2 error*

In a normal ripple carry adder, the carries and sum bits are not computed by independent circuits (i.e., the half-sum signals are used to compute the sum bits and also to propagate the carries). Thus an error to half-sum signal can modify both a carry c_i and an output sum s_i resulting in a bidirectional error.

3. *Type 3 error*

The error in carry may propagate to other adder blocks, causing bidirectional errors at the output of ALU. As our scheme encodes 2-bit, 3-bit or 4-bit data segments separately, so we applied test vectors on a 2-bit, 3-bit and 4-bit carry ripple adder separately. We call each 2-bit, 3-bit and 4-bit carry ripple adder a "block". We applied stuck at '0' and stuck at '1' faults on 1st (C_1), 2nd (C_2) and 3rd (C_3) carry of ripple carry adder, one at a time, as shown in Fig. 8.1. We found the percentage of faults which cause bi-directional change at the output sum bits (S) of the 2-bit, 3-bit and 4-bit carry ripple adder respectively.

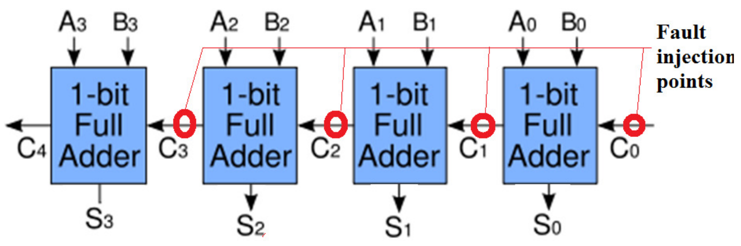


Figure 8.1 Fault injection points in a ripple carry adder

a) Type 3-A error:

For 4-bit data, we consider that primary input carry (C_0) is error free. A 4-bit adder can have $2^9 = 512$ input combinations (test vectors). Results show that if the stuck at '0' or at '1' error occurs on C_1 then 64 input combinations out of 512 possible combinations (12.5%) produce bidirectional errors at the output sum bits ($S_3S_2S_1S_0$). The stuck at errors on 2nd and 3rd propagating carry don't produce bi-directional error at the output sum at all.

b) Type 3-B error:

For a 3-bit carry ripple adder, stuck at '0' or '1' error on 2nd propagating carry (C_1) results in 16 out of 128 cases (12.5 %) in which the output sum ($S_2S_1S_0$) is bi-directionally corrupted by the error. The stuck at error on 2nd propagating carry (C_2) don't produce bi-directional errors at the output sum at all.

c) Type 3-C error:

For a 2-bit carry ripple adder, the stuck at error on 1st propagating carry (C_1) does not produce bi-directional error at the output sum at all (S_1S_0).

4. Type 4 error

Final carry bit of 2-bit (C_2), 3-bit (C_3) or 4-bit adder (C_4) can also be faulty, which may be propagated to next stages of carry ripple adder. Again we run the fault simulations and found that if input carry of a 2-bit, 3-bit or 4-bit adder is faulty, then 25 % of the input combinations can cause bidirectional error at the output sum bits.

5. *Type 5 error*

Final carry bit of 2-bit, 3-bit or 4-bit adder can also be propagated to the other adder groups in the chain. Fig. 8.3 shows an example of 8-bit ALU output and its 6-bit SEDC code. The 8-bit adder is implemented using one 2-bit and two 3-bit carry ripple adders. We consider that the erroneous carry is generated by the left most 3-bit adder and this error is also propagated to the next 3-bit adder as well as the 2-bit adder. We found that subset of the Type 4 errors (almost 25% of the type 4 errors) are the cause of Type 5 errors. These errors can be detected by our scheme.

B. Fault secureness

1. *Against type 1 error*

In our proposed SEDC encoded ALU design, Logic Unit and Arithmetic Unit inside the ALU are separated, hence we eliminate the probability of multiple random errors of Type 1.

2. *Against type 2 error*

To cope with this problem, we adapt the method described in [24], i.e., to implement independent circuits for propagate (or transmit) and half-sum signals. This introduces an overhead of one XOR gate per bit slice. We took this extra overhead into account when calculating the overall area overhead of our proposed scheme.

3. *Against type 3 error*

This type of error is not detectable by SEDC scheme. To cope with this error, we can add redundant carry out circuitry. As our scheme partitions the data into more numbers of 3-bit modules in which only type 3-B errors are present. If we generate two carry bits (i.e., C_1 & C_1') as the second propagating carry as shown in Fig. 8.2, and calculate the next Sum bit (S_1) and carry bit (C_2) by C_1 and C_1' respectively, then we can decrease the chances of bidirectional error due to type 3-B error.

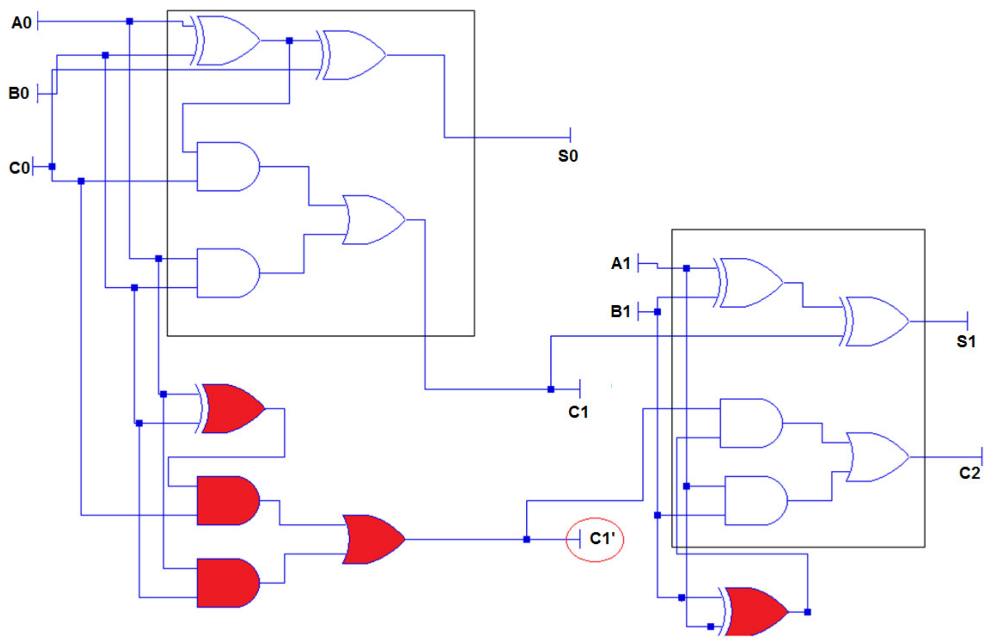


Figure 8.2 Adding redundant carry generating circuit to cope with Type 3-B errors

4. *Against type 4 error*

As we told, 25% of the type 4 errors result in type 5 errors, which are detected by our scheme (explained next), so we can say that only 18% of the type 4 errors cause bidirectional error at the output sum bits of the ripple carry adder, which is undetected by our scheme.

5. *Against type 5 error*

As our scheme encodes the segmented data (2-bit, 3-bit and 4-bit data), hence each pair of check bits are generated independently. If the output carry of one adder block is affecting the outputs of other adder block, then each pair of check bits will not correspond to the n-bit output of adder, and hence these errors can be detected. Fig. 8.3 shows the example of how a type 4 error resulted in a type 5 error, and is detected by SEDC check bits.

If Sum_2 denotes the normal output of the ALU, and Sum_2' symbolizes the erroneous output due to the presence of Type 5 error, then SEDC code of Sum_2' (i.e., SEDC(Sum_2')) will not be equal to the SEDC check bits generated by the SEDC encoded ALU (i.e., SEDC(Sum_2)). Consequently, the Type 5 error is detected.

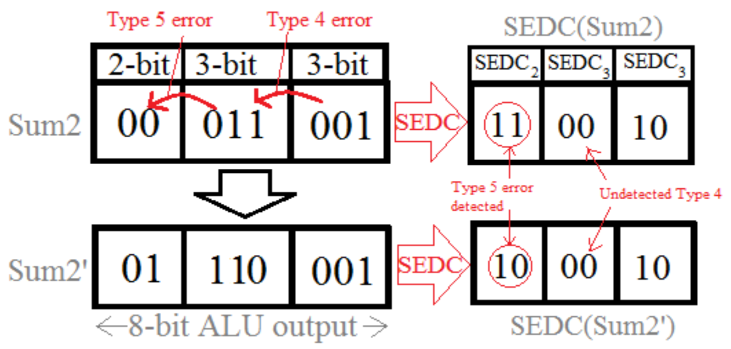


Figure 8.3 SEDC ALU organization that eliminates Type 5 errors

C. Overall Fault Coverage

Due to single fault, either Type 3 or Type 4 errors remain undetected by SEDC scheme. Among Type 3 and Type 4 errors, a subset of Type 4 errors (called Type 5 errors) are detected by SEDC scheme, so in worst case only 18% bi-directional faults stay undetected, while 100% unidirectional errors can still be detected.

The fault coverage of SEDC scheme can further be increased by putting redundant circuitry to cop against Type 3 and Type 4 errors. The redundant circuitry could specifically be added to certain locations of the normal adder by fully understanding the nature of the faults. For example, Type 3 error occurs when fault is present in first propagating carry, i.e., C_1. If we duplicate the circuit for C_1 only, then this type of error can completely be covered.

IX. Results of Fault Testing on TSC SEDC checker

The following definitions can be used to describe a Totally Self-Checking (TSC) system [4], [7], [10], [11], [22].

- ***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 code word, 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 code word for at least one valid input.

- ***Definition 3:*** A circuit is code disjoint iff it maps the input code space to output code space, and the non-input code space to non-output code space.

- ***Definition 4:*** A circuit is totally self-checking if it is *self-testing* and *fault-secure*.

A Circuit is TSC if it obeys all above definitions. TSC $SEDC_1$, $SEDC_2$, $SEDC_3$ and $SEDC_4$ circuits given earlier in this paper are tested for single stuck-at-0, stuck-at-1 and transistor-stuck-open and transistor-stuck-short faults. We assume that fault occurs one at a time and there is enough time between detection of first fault to the occurrence of other fault. Following is the analysis of SEDC checker circuit for satisfying all three properties of totally self-checking checker. We apply one fault at a time in the circuit of Fig. 7.2 and observe the output. Table 9-1 shows that under single fault operation, the circuit never produce any incorrect code word (hence its fault secure), generates error indication signal for at least one valid input code word (hence it is self-testing) and remains code disjoint (here {10} is the correct output code space).

- **Case 1- Transistor stuck ON:** In Table 9-1, we show all 6 cases of transistor stuck ON faults (one at a time). For the case of N3 or N4 stuck ON, the circuit shows fault detection by one input code combination (highlighted with dark), and hence the circuit is self-testing, while other cases show that the circuit is fault secure as well as code disjoint.

- **Case 2-Transistor stuck OFF:** In Table 9-1, all 6 cases for transistor stuck OFF fault are shown. In case of N1 or N2 stuck OFF, the circuit shows self-testing property (highlighted with dark) and for rest of the cases, the circuit is fault-secure.

- **Case 3- Input stuck at 0:** When input F_0 or C_0 is stuck at 0, the circuit shows self-testing property, otherwise it remains fault secure.

- **Case 4- Input stuck at 1:** When input F_0 or C_0 is stuck at 1, the circuit shows self-testing property, otherwise it remains fault secure.

Table 9-1. Results of single faults on TSC $SEDC_1$ checker

F_0	C_0	V_1	V_0	F_0	C_0	V_1	V_0	F_0	C_0	V_1	V_0
Transistor P1 is stuck ON				***Transistor P1 is stuck OFF***				***Input C_0 stuck at zero***			
0	1	1	0	0	1	1	0	0	0	1	1
1	0	1	0	1	0	1	0	1	0	1	0
Transistor P2 is stuck ON				***Transistor P2 is stuck OFF***				***Input F_0 stuck at zero***			
0	1	1	0	0	1	Floating	0	0	0	1	1
1	0	1	0	1	0	Floating	0	0	1	1	0
Transistor N1 is stuck ON				***Transistor N1 is stuck OFF***				***Input C_0 stuck at 1***			
0	1	1	0	0	1	1	0	0	1	1	0
1	0	1	0	1	0	1	1	1	1	0	0
Transistor N2 is stuck ON				***Transistor N2 is stuck OFF***				***Input F_0 stuck at 1***			
0	1	1	0	0	1	1	1	1	0	1	0
1	0	1	0	1	0	1	0	1	1	0	0
Transistor N3 is stuck ON				***Transistor N3 is stuck OFF***				-	-	-	-
0	1	0	0	0	1	1	0	-	-	-	-
1	0	1	0	1	0	1	0	-	-	-	-
Transistor N4 is stuck ON				***Transistor N4 is stuck OFF***				-	-	-	-
0	1	1	0	0	1	1	0	-	-	-	-
1	0	0	0	1	0	1	0	-	-	-	-

There are two cases where the output becomes floating (i.e., P2 stuck OFF). In either case, if we consider the floating voltage as logic high, then the circuit is fault secure, and if we consider the floating voltage as logic low, then the circuit is self-testing. Hence, we can say that the circuit in Fig. 7.2, which is a 1-bit SEDC checker, is TSC checker because it satisfies all the three axioms of being TSC. Similar analysis was carried out for testing 2-, 3- and 4-bit SEDC checkers and found that all these checkers are TSC.

X. Area, Delay, Complexity and Power Comparison

A. SEDC encoded ALU

1. Area overhead

Fig. 10.1 shows the gate count for normal, SEDC encoded and BCP ALUs having 8-, 16- & 32-bit inputs. Data for a 16-bit Residue ALU (ALU+Controller) [15] is also presented for comparison. For the gate count, all the logic tables (Table 5-1 - 6-2) and equations are logically minimized and synthesized using Logic Friday logic minimize software [25]. All the circuits for SEDC encoded ALU are functionally tested using Modelsim software with Verilog language, while the post fitting simulations are carried out using Quartus II software. Altera's Cyclone III device was used for post-layout simulations.

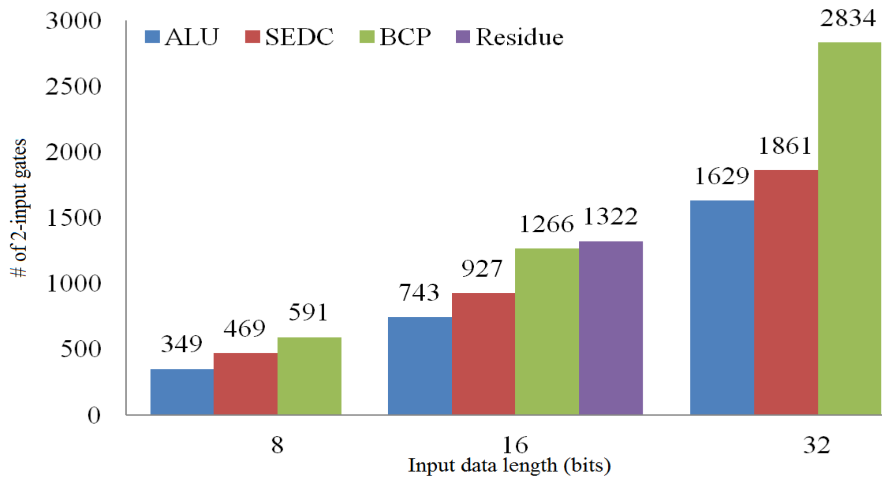


Figure 10.1 Area utilization chart for SEDC encoded, BCP and simple ALU

The gate count includes all the circuitry inside SEDC encoded ALU shown in Fig. 4.1 except the Compare unit. The extra XOR gates that cope with Type 2 errors are also taken into account while calculating data for Fig. 10.1. The multiplexer that selects the particular operation is also considered.

The data for BCP based ALU is taken from [5], [16], [26] & [27] with few changes: the data in [10] does not include the area overhead due to zero's counter for operands (i.e, X_c and Y_c), shown in Fig. 10.2 [5]. The Multiple Carry Save Adder (MCSA) contains chains of full and half adders and their exact gate count is estimated from [26]. Circuits in [27] are implemented with multiple input gates, so we translated all the circuits to 2-input gates, for uniformity.

The increasing trend in BCP hardware is due to the presence of three n-bit counters; one right after the 3x1 input MUX and two for calculating 'X_c' and 'Y_c' respectively, as shown in Fig. 10.2. Although 'X_c' and 'Y_c' are re-utilized for whole data path, still they add to overall area. P. K. Lala [27] proposed the most area optimized realization of the zeroes counter as compared to previous implementation. The area required to implement these zero counters becomes more than double when the number of input bits to the zero counter is doubled.

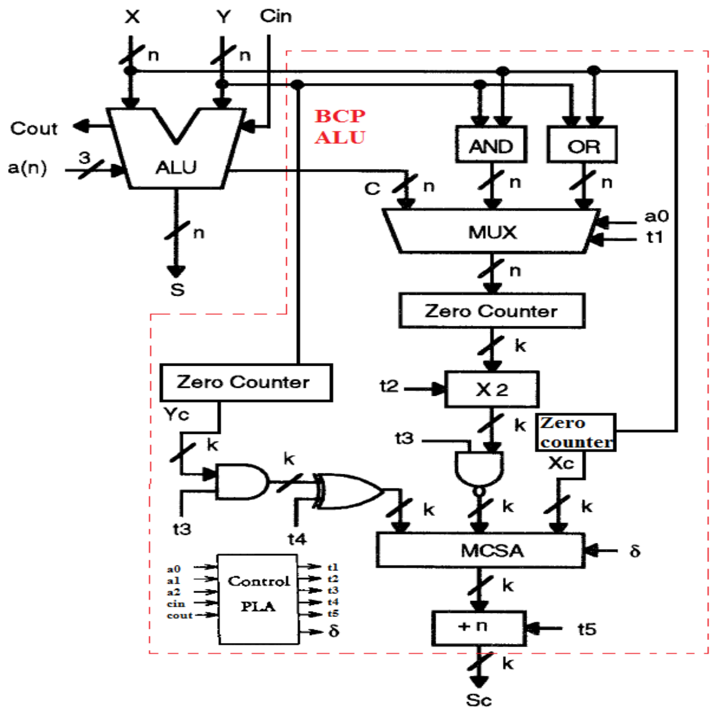


Figure 10.2 Berger Check Prediction Arithmetic and Logic Unit

The size of MCSA block also increases with increasing number of inputs (2 full adders and 1 half adder for each increasing 'k'-bit). The number of 'AND' & 'OR' gates increases linearly with 'n' (one 2-bit gate with each increasing input) while XOR and NAND gates increases linearly with increasing 'k', in Fig. 10.2. 'MUX' (NOT-AND-OR based implementation) takes twice the gate counts while the size of 'X2' block (n-bit 2x1 mux) increases slowly when the number of inputs are doubled. The ‘+n’ block has very negligible effect on overall circuitry, so its gate count is not taken into account.

In short, every single unit occupies twice or more than twice area if input data length is doubled, except ‘k’ input gates and ‘X2’ block which have little increment in their hardware. But, more than doubling effects of three zeroes counters has the dominant effect on overall gate requirement. The resultant effect is that, the overall gate count becomes twice if the input data length is doubled.

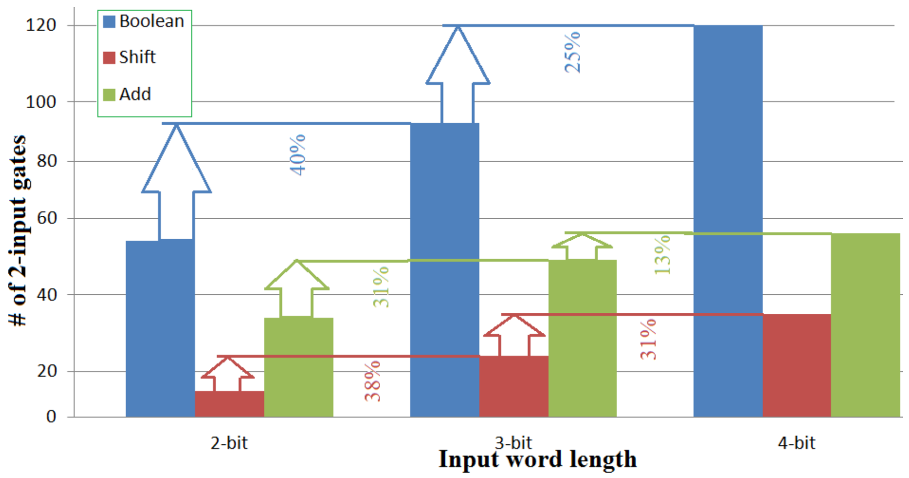


Figure 10.3 Increasing trend in SEDC encoded ALU hardware

On the other hand, SEDC uses set of pre-defined 2-, 3- and 4-bit modules to implement n-bit SEDC encoded ALU. By looking at Fig. 10.3, one can figure out that the increasing trend in SEDC encoded ALU hardware is slower than that of BCP. It is also evident from Fig. 10.1 that SEDC encoded ALU takes less than twice the hardware, when input data length is doubled, which concludes that SEDC encoded ALU occupies less area than BCP circuitry for any number of input bits ‘n’.

2. *Delay*

The PLA block in Fig. 10.1 generates the control signals (t1, t2, t3, t4, t5) after the arrival of C_{out} from the normal ALU. The MCSA unit is made up of full and half adder trees. As stated previously, the number of full and half adders increase with the increasing inputs so the delay in generation of check symbol is also increased. On the other hand, SEDC encoded ALU does not wait for internally generated carries of normal ALU for computation of check bits. Moreover, SEDC uses parallel modules of 2-, 3- and 4-bit SEDC encoded ALUs, so the maximum delay incurred is equal to the latency of a 4-bit input SEDC module, hence maximum delay remains same for any number of input bits '$n \geq 4$'.

3. *Power and Complexity in Scaling*

The scaling of SEDC encoded ALU is also very simple. Scaling method of BCP ALU is not presented in [5] but more than doubling of the circuit is required to scale the BCP ALU for doubling the input bits which increases the overall complexity. Bose-Lin scheme in [16] is not as simple when more than three check bits are required. Some changes in MCSA block need to be made. In case of scaling SEDC, the scaled circuit is the same un-scaled circuit plus some extra gates. For more than 4-input bits, the circuit requires replication of 2-, 3- and 4- bit SEDC encoded ALU modules. Due to this replication, the power distribution of the overall circuitry is very uniform.

For scaling Bose-Lin scheme with more than 4 check bits, X2 block is completely changed (Fig. 10.2). While for SEDC, only one inverter is required to convert $SEDC_3$ to $SEDC_4$.

B. TSC SEDC checker

In this section, area consumed by TSC SEDC checker and wired AND-OR network is compared with TSC Berger one's counter and two rail checker. As the code rate of SEDC scheme is more than that of Berger scheme, hence for fairness in comparison, we also consider the area utilized to store the check bits.

1. *Area Overhead*

The TSC $SEDC_n$ checker block shown in Fig. 4.1 requires fewer gates (whose circuit diagrams are given in chapter VII), that are implemented with $[15 + (a \times 39)]$ MOS transistors if 'b' value is 2, $[39 + (a \times 39)]$ MOS transistors if 'b' value is 3 or $[45 + (a \times 39)]$ MOS transistors if 'b' value is 4. The wired AND-OR logic network is implemented with '$d + r$'

MOS transistors where d = {2×(a+1)}and r = ⌈d/4⌉. As far as the latency of overall TSC SEDC checker is concerned, it is observed constant for n > 3 as shown in Table 10-2. SEDC circuits are first designed by Logic Friday software using logic equations in chapter VII. These circuits are then implemented with Verilog HDL using Modelsim software, and then finally synthesized by Altera's Quartus II.

We took data for combinational implementation of one's counter from [27] in which gate level circuit diagrams for up to 32-bit one's counter are given. For translating gate level circuits to transistor level circuits we use data given in [20].

Although the SEDC scheme has a bigger code size than Berger coding scheme, if we consider the overall area, it is observed that TSC SEDC checker takes less area than TSC Berger checker. Table 10-1 enlists the area (in terms of # of transistors) for implementing the TSC SEDC and Berger checkers for up to 32-bit information word. It can be seen from Table 10-1 that with the increase in binary data length area increases if we consider the combinational implementation of Berger code [27]. SEDC scheme takes almost 67% less area than latency optimized version of Berger scheme [27] even after taking into account the storage area for check bit.

Table 10-1. Area comparison between SEDC and Berger checker

Data Bit	**Berger Code**				**SEDC**			
	Code storage Area	***1's counter Area***	***TRC Area***	***Total Area***	***Code storage Area***	***Checker Area***	***AND-OR Network***	***Total Area***
2	24	18	4	**46**	24	15	0	**39**
3	24	46	8	**78**	24	39	0	**63**
4	36	123	12	**171**	36	45	6	**87**
5	36	114	16	**166**	48	54	6	**108**
7	36	208	24	**268**	60	84	8	**152**
8	48	246	28	**322**	72	93	8	**173**
9	48	355	32	**435**	72	117	8	**197**
15	48	686	56	**790**	120	195	14	**329**
16	60	879	60	**999**	132	321	16	**469**
30	60	1640	116	**1816**	240	390	26	**656**
32	72	1939	120	**2131**	264	405	28	**697**

2. *Delay*

As far as the delay is concerned, again it is observed to be constant for SEDC scheme. The maximum latency of SEDC checker is limited to 4 equivalent MOS transistor levels, which does not affect the overall performance. The reason behind the constant latency being the use of wired-AND-OR circuitry as the equivalency tester, rather than two rail checker. The n-bit TSC SEDC checker itself consists of small parallel checkers, that are, $SEDC_1$, $SEDC_2$, $SEDC_3$ or $SEDC_4$ checkers.

On the other hand, Berger checker provides delay, both due to ones counters as well as the tree structure of two rail checkers. Moreover, this delay keeps on increasing as the data length increases, which is undesirable for delay optimized reconfigurable embedded architectures, like FPGA.

Table 10-2. Critical Path (C.P) delay comparison of TSC Berger and SEDC checker (unit = nanoseconds)

Data Bits	**Berger**			**SEDC**		
	C.P of 1's counter	***C.P due to TRC***	***Total C.P***	***C.P of SEDC checker***	***C.P of AND-OR network***	***Total C.P***
2	20.4	11.6	**32**	35.6	0	**35.6**
3	22.1	23.2	**45.3**	42.2	0	**42.2**
4	35.9	23.2	**59.1**	42.2	10.8	**53**
5	59.5	23.2	**82.7**	42.2	10.8	**53**
7	59.5	23.2	**82.7**	42.2	10.8	**53**
8	93.5	23.2	**116.7**	42.2	10.8	**53**
15	138.6	34.8	**173.4**	42.2	10.8	**53**
16	151.1	34.8	**185.9**	42.2	10.8	**53**
30	207.2	46.4	**253.6**	42.2	10.8	**53**
32	244.2	46.4	**290.6**	42.2	10.8	**53**

XI. Prototyping the SEDC based error detecting reconfigurable ALU architecture on FPGA

In this chapter we discuss the FPGA implementation of overall system whose block diagram was presented in chapter IV. We also present the method of autonomous reconfiguration of FPGA upon detection of error in an 8-bit ALU. The comparison of area overhead of FPGA implementation of SEDC, DMR, BCP and TMR schemes are also evaluated in this chapter.

A. Overall FPGA based system design

The overall block diagram of FPGA and PC based fault tolerant ALU system is shown in Fig. 11.1. The SEDC based error detecting and reconfiguring ALU architecture consists of an 8-bit normal ALU, an 8-bit SEDC encoded ALU and an 8-bit TSC SEDC checker. This system is implemented on FPGA along with the logic to send error signal to PC via PS2 port. If the error is transient, the program counter of the ALU halts and current instruction is recomputed until the fault disappears. If the error prevails for more than specified amount of clock cycles, then the error signal is sent to PC via the PS2 port. The PC containing Labview software receives the error signal and reconfigures the FPGA with fresh bit stream of the system without running the complete design flow of FPGA design implementation, as discussed later. The USB JTAG port carries this fresh bit stream for reconfiguring FPGA.

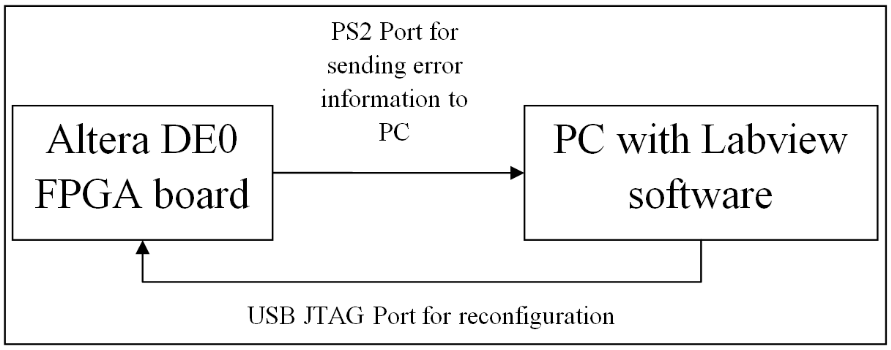

PS2 Port for sending error information to PC
Altera DE0 FPGA board
PC with Labview software
USB JTAG Port for reconfiguration

Figure 11.1 Overall prototype diagram for fault tolerant ALU

In the subsection, we discuss the implementation details of each and every module.

1. Altera DE0 FPGA board

Fig. 11.2 shows the detail view of Altera's DE0 FPGA board. The board contains Cyclone III EP3C16F484 FPGA chip. This FPGA contains 16k Logic Elements (LE) and 484 usable I/O pins. The board consists of many peripherals out of which the following listed peripherals are used in our project.

- **10 Toggle :** Due to the limited number of switches, we multiplexed the data using two switches SW_1 and SW_0. The remaining switches are used to input the operands 'A', 'B', 'p' and operation selecting input 'op'. Toggle switches SW_1 and SW_0 act as selector switches for multiplexing the inputs A, B, p and op whose detailed function is enlisted in Table 11-1.

Table 11-1. Function of selection switches & seven segments

S.No	SW_1SW_0	Mode	SW9-SW2	Seg_2-Seg_0
1	00	Input first operand A	8-bit Operand 'A'	Operand A
2	01	Input second operand B	8-bit Operand 'B'	Operand B

3	10	Input Cin (in case of Add/Subtract operation) or
		
		Input si (in case of shift/rotate operation)
			SW9-SW3 = don't care
			
			SW2 = 'p'
				1-bit 'P'

4	11	Select the operation to be performed	8-bit 'op'	Result S

- **3 Push Buttons :** Push Button 2 is designated as the system reset while Button 0 is used to manually clock the circuit. Button 1 is used to force the outputs of ALU (S) to some faulty condition, and hence this situation is simulated as a fault.

- **4 Seven Segment Displays :** The four seven segment displays are used to show the status of inputs as well as the outputs. Table 11.1 also lists the status of Segments at every possible position of SW_1 and SW_0.

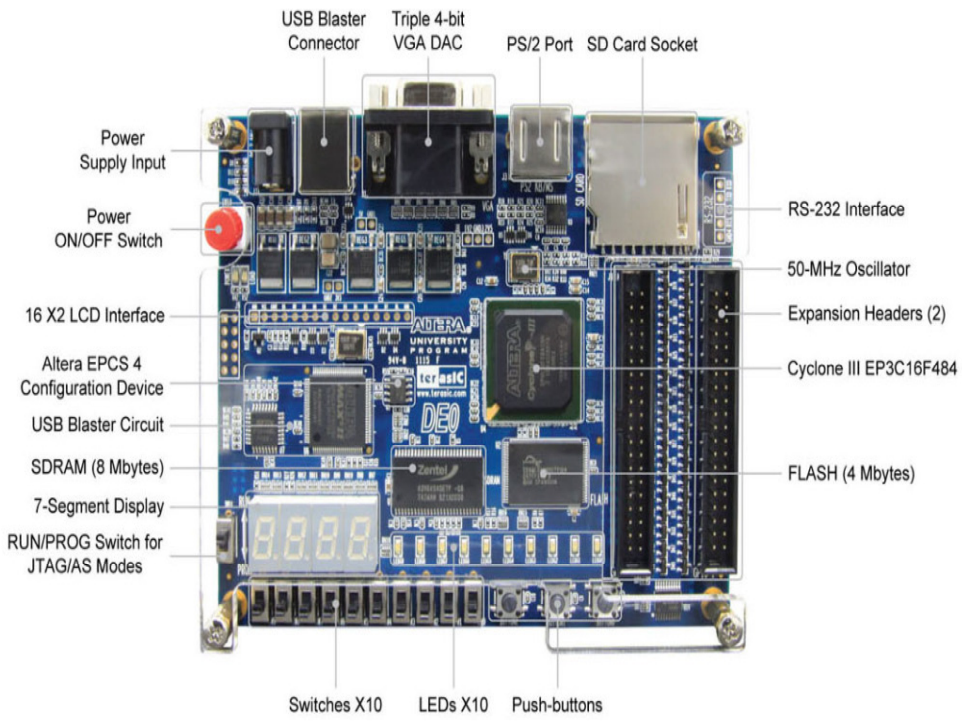


Figure 11.2 DE0 Board

- **PS/2 Port** PS/2 Port is used to send the interrupt to the PC upon detection of error. The ASCII code of ESCAPE key is used to send to PC via PS2 port, which is recognized by the Labview's virtual instrument as error.
- **9 Green LEDs** LEDG9-LEDG8 are used to indicate the status of 2-bit output of the TSC SEDC checker, while the output of SEDC encoded ALU (i.e., the SEDC check bits) is shown on LEDG5-LEDG0.

2. *NI Labview Software*

In this project, we used NI Labview 9.0 for developing a keystroke logger whose job is to receive the error signal from FPGA and then reprogram the FPGA. The block diagram consists of a keyboard stroke capture module and a while loop. The keyboard stroke capture module fires the while loop when a particular keystroke is observed from the PS2 port of the PC. The while loop then executes a batch file that contains required instructions for provoking the Programmer which then reprograms the FPGA with given bit stream file via USB JTAG port.

The reconfiguration also resets the FPGA circuitry, and hence there is a high probability that the permanent faults are removed (except manufacturing defects). Rather than going through all the steps of FPGA design, the programming file (.cdf) for the complete system is once generated and reutilized for reprogramming. The typical reconfiguration time taken by Cyclone III chip is 20ms, and it varies with the size of .cdf file.

B. Implementation Results & Area Comparison

Fig. 11.3 shows the area comparison between normal ALU, SEDC ALU and BCP ALU. These ALUs are implemented on Cyclone III FPGA EP3C16F484 chip and their area in terms of Logic Element counts is presented. Result shows that the FPGA implementation of SEDC encoded ALU takes 16% more area than simple ALU while BCP ALU takes 90% more area. Less area consumption of SEDC is due to the fact that SEDC circuits are less complex and can easily be synthesized by the synthesizing software. A 16-bit SEDC ALU is composed of smaller 2-, 3- and 4-bit ALUs, which require less area when synthesized by the software. SEDC encoded ALU don’t require internally generated carries from the normal ALU, which cause longer latencies as well as large area consumption after synthesizing on FPGA.

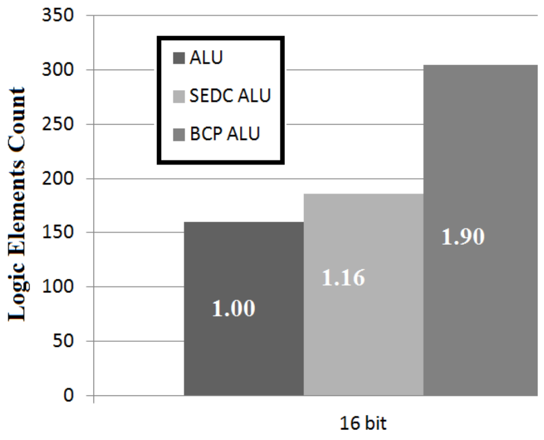


Figure 11.3 Area comparison of FPGA implementation of 16-bit normal ALU, SEDC ALU and BCP ALU

BCP ALU is implemented using the equation (20)-(37) [5]. The symbols ‘Xc’, ‘Yc’, ‘Cc’ and ‘Sc’ denote the zeroes count value of input operands ‘X’, ‘Y’, internally generated carries ‘C’ and output of ALU ‘S’, respectively. To convert X, Y and C to Xc, Yc and Cc, we require 16-bit, 16-bit and 15-bit zeroes counters respectively. Another zeroes counter is required to convert the output of Logic operation unit to its equivalent zeroes count (represented as $(X \vee Y)_c$ in equation (22)). These bulky zero counter circuits constitute much of the Logic Elements when implemented on FPGA. Equations (20) and (21) contains multiple add operations, hence they require multiple levels of adder circuits which further requires more area on FPGA.

$$S_c = X_c + Y_c - C_c - c_{\text{in}} + c_{\text{out}} \tag{20}$$

$$S_c = X_c - Y_c + N(C) - \bar{c}_{\text{in}} + c_{\text{out}}. \tag{21}$$

$$S = X \wedge Y \Rightarrow S_c = X_c + Y_c - (X \vee Y)_c \tag{22}$$

$$S = \overline{X} \wedge Y \Rightarrow S_c = -X_c + Y_c + (\overline{X} \vee Y)_c \tag{23}$$

$$S = X \wedge \overline{Y} \Rightarrow S_c = X_c - Y_c + (X \vee \overline{Y})_c \tag{24}$$

$$S = \overline{X} \wedge \overline{Y} \Rightarrow S_c = -X_c - Y_c + (X \vee Y)_c + n \tag{25}$$

$$S = X \vee Y \Rightarrow S_c = X_c + Y_c - (X \wedge Y)_c \tag{26}$$

$$S = \overline{X} \vee Y \Rightarrow S_c = -X_c + Y_c + N(\overline{X} \wedge Y) \tag{27}$$

$$S = X \vee \overline{Y} \Rightarrow S_c = X_c - Y_c + N(X \wedge \overline{Y}) \tag{28}$$

$$S = \overline{X} \vee \overline{Y} \Rightarrow S_c = -X_c - Y_c + (X \vee Y)_c + n \tag{29}$$

$$S = X \oplus Y \Rightarrow S_c = X_c + Y_c - 2(X \wedge Y)_c + n \tag{30}$$

$$S = X \odot Y \Rightarrow S_c = -X_c - Y + 2N(\overline{X} \wedge Y) \tag{31}$$

$$S = X \Rightarrow S_c = X_c \tag{32}$$

$$S = Y \Rightarrow S_c = Y_c \tag{33}$$

$$S = \overline{X} \Rightarrow S_c = n - X_c \tag{34}$$

$$S = \overline{Y} \Rightarrow S_c = n - Y_c \tag{35}$$

$$S = 0 \Rightarrow S_c = n \tag{36}$$

$$S = 1 \Rightarrow S_c = 0. \tag{37}$$

XII. Conclusions and Future Enhancements

In this thesis we presented a new architecture for detecting all unidirectional errors in ALU using SEDC scheme. We presented theory and design technique of an SEDC encoded ALU. We also discussed the method of scaling SEDC encoded ALU for n-bit input data length with simple addition in hardware, which does not affect the overall latency of the system. We show that the complexity in scaling the SEDC circuit is less than Bose-Lin implementation [16], while the SEDC encoded ALU is faster than both [5] & [16]. We implemented the circuits of BCP ALU and SEDC encoded ALU using 2-input logic gates for uniformity in calculation of gate counts and found that BCP ALU hardware requires more area than SEDC encoded ALU for same input data length. We also show that SEDC provides 82% fault coverage against all errors that result due to single faults in ALU. Hence, for delay and area sensitive applications, n-bit SEDC encoded ALU performs better in terms of speed & complexity than [5] and [16] with optimum fault coverage.

We also presented a totally self-checking checker for scalable error detecting codes. We introduced how this SEDC scheme utilizes the parallelism concept by partitioning of input data bits “D” into 2-, 3- and 4- bit data segments and then encode those using $SEDC_2$, $SEDC_3$ and $SEDC_4$ schemes. Then for checker part, SEDC scheme again partitions the functional circuitry outputs into 2-, 3- and 4-bit segments and use TSC $SEDC_2$, $SEDC_3$ and $SEDC_4$ checker blocks to detect all unidirectional errors. The proposed SEDC checker is tested and shown to be totally self-checking by fault testing method. We also show that due to specific 2-bit error indication signal by each TSC checker module, overall SEDC TSC checker employs TSC wired-AND and wired-OR logic gates as an equality tester of code bits rather than TSC two rail checkers tree, which makes SEDC TSC checker more area and delay efficient than Berger TSC checker. Also the inherited parallelism and independency between modules within n-bit checker makes the checker faster.

Lastly, we implement an 8-bit SEDC based error detecting ALU system on Altera’s FPGA. We designed a Labview VI that receives error signal from FPGA and reconfigures it. We again compare the area utilization by SEDC and BCP schemes, this time on FPGA, and found that SEDC requires less area as compared to BCP.

In future, SEDC based error detecting ALU can be more refined for locating errors. This will enable us to use partially reconfiguring in only those parts of ALU that are effected by error. Separate PC for reconfiguring the FPGA can also be replaced with a microcontroller to increase the portability of the system. The results in our research also motivate to design error secure data path of the complete microprocessor.

Bibliography

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

[2] V. S. Veeravalli, “Diagnosis and error correction for a fault-tolerant Arithmetic and Logic Unit for medical microprocessors,” M.S. thesis, Dept. of Elec. & Comput. Eng., NBR Univ., New Jercy, 2008.

[3] A. Rohani, H. R. Zarandi, “An Analysis of fault effects and propagations in AVR Microcontroller ATmega103(L),” *IEEE Int. Conf. on Availability, Reliability & Security*, pp. 166-172, 2009.

[4] M. Al-Ani and Q. Al-Shayea, “Unidirecitonal Error Correcting Codes for Memory Systems: A Comparative Study,” *Int. J. of Comput. Sci. Issues*, vol. 7, Issue 1, no. 3, 2010.

[5] J.-C. Lo, S. Thanawastein, and M. Nicolaidis, "An SFS Berger check prediction ALU and its application to self-checlking processor designs," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 11, pp. 525-540, April 1992.

[6] R. Bickham, “An analysis of error detection techniques for Arithmetic Logic Units,” M.S. thesis, Dept. of Elec. Eng., Vanderbilt Univ., Nasville, Tennessee, 2010.

[7] N. K. Jha and S. J. Wang, “Design and synthesis of self-checking VLSI circuits,” *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 12, pp. 878-887, June 1993.

[8] M. Alderighi, S. D. Angelo, C. Metra, & G. R. Sechi, "Achieving fault-tolerance by shifted and rotated operands in TMR non-diverse ALUs," *Proc. IEEE Int. Symp. on Defect and Fault Tolerance in VLSI Syst.*, pp.155-163, 2000.

[9] S. Hong & S. Kim, "Lizard: Energy-efficient hard fault detection, diagnosis and isolation in the ALU," *IEEE Int. Conf. on Comput. Des.*, pp. 342-349, Oct. 2010.

[10] R. W. Cook et al., “Design of self-checking microprogram control," *IEEE Trans. Comput.*, vol. c-22, pp. 255-262, Mar. 1973.

[11] D. K. Pradhan and J. J. Stiffler, "Error correcting codes and self-checking circuits in fault-tolerant computers," *IEEE Trans. Comput.*, vol. 13, pp 27-37, Mar. 1980.

[12] V. Srinivasan, J. W. Farquharson, W. H. Robinson, and B. L. Bhuva, "Evaluation of Error Detection Strategies for an FPGA-Based Self-Checking Arithmetic and Logic Unit," *Military & Aerospace Programmable Logic Devices Conf.*, Washington, D.C., Sept. 2005.

[13] R. Forsati, K. Faez, F. Moradi and A. Rahbar, "A Fault Tolerant Method for Residue Arithmetic Circuits," *Int. Conf. on Inform. Manage. & Eng.,* pp.59-63, 3-5 April 2009.

[14] M. Nicolaidis, R. O. Duarte, S. Manich and J. Figueras, "Fault-secure parity prediction arithmetic operators," *IEEE Des. & Test,* vol.14, no.2, pp.60-71, Apr-Jun 1997.

[15] M. Medwed, S. Mangard, "Arithmetic logic units with high error detection rates to counteract fault attacks," *Proc. of Europe Conf. on Des., Automation & Test,* pp. 1-6, 14-18 March 2011.

[16] S. S. Gorshe and B. Bose, "A Self-Checking ALU Design with Efficient Codes," *Proc. of 14th VLSI Test Symp.*, pp. 157-161, May. 1996.

[17] V. Khorasani, B. V. Vahdat and M. Mortazavi, "Analyzing Area Penalty of 32-Bit Fault Tolerant ALU Using BCH Code," *14th Euromicro Conf. on Digital Syst. Des.*, pp.409-413, Aug-Sept. 2011.

[18] D. P. Vasudevan, P. K. Lala & J. P. Parkerson, "Self-Checking Carry-Select Adder Design Based on Two-Rail Encoding," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 54, no. 12, pp. 2696-2705, Dec. 2007.

[19] S. Natarajan and J. A. Lee, "Self-checking programmable arithmetic & logic unit (PALU) having the scalable error detection code (SEDC) generator and method of scalable error detection code," *Korean Patent* Application no. 10-2011-XXXXXXX.

[20] M. A. Smith, "Equivalent gate counts," 2013, Available : http://en.wikipedia.org/wiki/Transistor_count

[21] S. Mitra, E. J. McCluskey, "Which concurrent error detection scheme to choose," *Proc. Int. Test Conf.*, pp. 985-994, 2000.

[22] S. J. Piestrak, D. Bakalis & X. Kavousianos, "On the design of self-testing checkers for modified Berger codes," *IEEE Online Testing workshop*, pp. 153-157, 2001.

[23] S. Natarajan and J. A. Lee, "Scalable totally self-checking checker for self-checking processing unit based on scalable error detection coding (SEDC) algorithm and processing system having the checker," *Korean Patent* Application no. 10-2012-XXXXXXX.

[24] G. G. Langdon Jr. & C. K. Tang, "Concurrent error detection for group lookahead binary adders," *IBM j. Res. Develop.,* pp. 563-573, Sept. 1970.

[25] Logic Friday Software : http://sontrak.com/

[26] J.-C. Lo, S. Thanawastien and T. R. N. Rao, "Concurrent error detection in arithmetic and logical operations using Berger codes," *Proc. of 9th Symp. on Comput. Arithmetic*, pp. 233-240, Sep 1989.

[27] D. A. Pierce, JR. and P. K. Lala, "Modular implementation of efficient self-checking checkers for the Berger code," *J. of Electron. Testing: Theory and Applicat.*, vol. 9, pp. 279-294, 1996.

ABSTRACT

Online Fault Detection and Reconfiguration of ALU using Scalable Error Detection Coding Scheme

Zahid Ali
Advisor: Prof. Jeong-A Lee, Ph. D.
Department of Computer Engineering
Graduate School of Chosun University

Microelectronic circuits and SRAM-based FPGA devices are becoming more vulnerable to faults and errors due to shrinking size and higher packaging densities for the transistors. As a result, error detection becomes a vital concern for system reliability.

Errors can be broadly classified as soft and hard errors. Soft errors are caused by transient or intermittent faults, while hard errors are caused by persistent faults. Studies show that the majority of errors are caused by transient faults.

Techniques to detect soft errors caused by transient faults have been developed and tradeoff is usually made between processor performance and the area and power required for error detection. Diverse duplex circuits can cope against most of the errors including common mode failure (CMF) at the cost of twice the area overhead. Triple Modular Redundancy can be used to reduce the complexity of the system by just copying the same circuit three times and using a voting circuit. Time redundant methods like re-computing with rotated operands (RERO) and with shifted operands (RESO) are employed to save area overhead, but they introduce unavoidable delay to the system. Several concurrent error detection (CED) methods involve encoding the functional circuit using some codes, like arithmetic codes, Berger codes and parity codes. For efficient implementation of CED techniques, it is important to consider the relevant types of faults that are supposed to be more probable to occur. The types of faults within a VLSI circuit have been analyzed and found to be unidirectional errors. Unidirectional errors can alter the node logic from zero to one or from one to zero, but not both at the same

time. Unidirectional Error Detection (AUED) technique provides good fault coverage with reduced area overhead.

In this thesis, we employ an error detection scheme called Scalable Error Detection Coding (SEDC) which is capable of detecting single as well as multiple unidirectional errors. SEDC scheme partition the data into segments and perform parallel encoding for assigning code words. Consequently, SEDC scheme can be scaled for any binary data length 'n' with constant latency and less complexity as compared to other All Unidirectional Error Detection (AUED) schemes.

Using SEDC scheme, we present a fault tolerant ALU architecture that achieves high fault tolerance against single event upsets. The proposed SEDC encoded ALU performs better in terms of area and delay as compared to the previous implementation. Result shows that ASIC implementation of SEDC based error detecting 32-bit ALU saves 34% area while FPGA implementation of 16-bit SEDC encoded ALU saves 39% area as compared to the Berger Code Prediction ALU [5]. We also present an area and delay efficient, scalable, Totally Self-checking (TSC) checker for SEDC scheme. The proposed 32-bit checker achieves 67% reduction in area and 81% improvement in delay as compared to TSC Berger checker. We also utilize the reconfiguration feature of FPGA to mitigate hard errors.

ACKNOWLEDGMENT

I would first of all thank Allah Almighty Who enabled me to carry out this project with full devotion and consistency. It is only because of His blessings that I could find my way up to the completion of this task.

Next, I would like to express my immense regards and honest gratitude to my supervisor, Prof. Jeong-A Lee. Her valuable support, guidance, appreciation and supervision, throughout the course of this novel research, motivationally steered me to accomplish this task successfully.

I would like to thanks Dr. S. Natarajan for his contributions to this work. Also my lab mates Olufemi Adeluyi and Park Hui-Jong for their absolute help.

Finally, I also pay my esteem regards to MKE (Ministry of Knowledge Economy), Korea under the Global IT Talents Program supervised by NIPA (National IT Industry Promotion Agency), for its financial support during the period of my Master studies.