August 2014
Master's Degree Thesis

A Self-Repairing Adder with Fault Localization

Graduate School of Chosun University

Department of Computer Engineering

Muhammad Ali Akbar

A Self-Repairing Adder with Fault Localization

오류의 국지화를 이용한 효율적인 자가오류회복 가산기 설계

August 25, 2014

Graduate School of Chosun University

Department of Computer Engineering

Muhammad Ali Akbar

A Self-Repairing Adder with Fault Localization

Advisor: Prof. Jeong-A Lee, PhD

A thesis submitted in partial fulfillment of the requirements for a Master's degree

April 2014

Graduate School of Chosun University

Department of Computer Engineering

Muhammad Ali Akbar

아크바르 무하마드 알리
석사학위논문을 인준함

위원장 조선대학교 교수 신석주 (인)

위 원 조선대학교 교수 최광석 (인)

위 원 조선대학교 교수 이정아 (인)

2014 년 5 월

조선대학교 대학원

Table of Contents

List of Figures

List of Tables

ABSTRACT

A Self-Repairing Adder Design using Fault Localization

Muhammad Ali Akbar
Advisor: Prof. Jeong-A Lee
Department of Computer Engineering
Graduate School of Chosun University

Advanced microelectronics technologies caused the current digital systems to become more vulnerable to faults. It has been observed that the problem of single-event upset in digital systems has become more prominent with the increasing complexity of system on a chip, along with decreasing clock cycles to obtain high operating frequency. Moreover, the increasing complexity of digital system also increases the cost of repairing. Therefore, the existence of fault detection without recovery cannot fulfill the demand of reliable execution in current digital system. Thus, the built-in self-repair is becoming the need of current semiconductor technology.

In this thesis, we propose a self-repairing adder that can repair multiple faults and identify the particular faulty full adder. Fault detection and recovery has been carried out using self-checking full adders that can diagnose the fault based on internal functionality, independent of a fault propagated through carry. The idea was motivated by the common design problem of fault propagation due to carry in various approaches by self-checking adders. Such a fault can create problems in detecting the particular faulty full adder, and we need to replace the entire adder when an error is detected. We apply our self-checking full adder to a carry-select adder (CSeA) and show that the resulting self-

checking CSeA consumes 15% less area compared to the previously proposed self-checking CSeA approach without fault localization. After observing fault localization with reduced area overhead, we utilize the self-checking full adder in constructing a self-repairing adder. It has been observed that our proposed self-repairing 16-bit adder can handle up to four faults effectively, with an 80% probability of error recovery compared to triple modular redundancy, which can handle only a single fault at a time.

한 글 요 약

오류의 국지화를 이용한 효율적인 자가오류회복 가산기 설계

아크바르 무하마드 알리
지도 교수: 이 정 아
컴퓨터공학과
대학원, 조선대학교

향상된 마이크로 전자 기술은 현재의 디지털 시스템을 오류에 더 취약하게 만들었다. 감소 된 클럭사이클과 함께 시스템의 복잡도가 증가하여 시스템이 오류발생에 취약하게 되었다. 일상생활에서 활용되는 디지털 시스템이 증가됨에 따라 하드웨어가 안정적으로 작동되는 것이 요구된다. 위성, 수술시스템과 자동차의 자동 브레이크 시스템 등에서 디지털 시스템이 오류를 발생할 시 경제와 안전에 치명적인 영향을 끼칠 수 있다. 따라서 오류복구가 없이 오류검출만 하는 것은 현재의 디지털 시스템의 안정적인 실행이라는 요구사항을 충족시킬 수 없다. 그러므로 현대의 반도체기술은자가오류회복기술을 내장할 필요가 있다.

본 논문에서는, 우리는 다수의 오류를 복구하거나 특정한 오류를 검출할 수 있는 '자가 복구 전가산기'를 제안한다. 오류의 검출과 복구는 내부 함수에 기초한 '자가 복구 전가산기'를 사용하여 수행되지만, 이러한 방법은 가산기의 캐리전달에 의한 오류와는 독립적으로 수행된다. 이 아이디어는 자가 복구 가산기에 의한 다양한 접근법들 중, 캐리 전달에 의해 발생하는 오류 전송의 공통된 설계 문제 해결에 기인한다. 캐리전달에 따른 오류탐지는 특정한 오류가 있는 전가산기를 찾는 데에 문제를 야기할 수 있으며, 에러가 발생할 경우 모든 전가산기를 교체해야만 한다. 우리는 우리의 '자가 복구 전가산기'를 Carry-Select Adder(CSeA)에 적용하고, 그 결과 오류가 발생한 가산기를 찾을 수 없는 이전의 CSeA 에 비해 15%의 오버헤드 공간 감소 효과가 있음을 확인하였다. 오류의 국지화를 통하여 면적 오버헤드 감소가 가능하였고, 본 논문에서는 '자가 검사 전가산기'를 '자가 복구 가산기'를 설계하는 데 활용하였으며, 우리가 제안한 16bit 의 자가 복구 가산기는 하나의 오류만을 복구할 수 있었던 Triple modular redundancy 방식에 비해 80% 확률로 4 개의 오류까지 복구할 수 있음을 확인하였다.

I. Introduction

A. **Research Overview**

Advanced microelectronics technologies caused the current digital systems to become more vulnerable to faults. It has been observed that the problem of single-event upset in digital systems has become more prominent with the increasing complexity of system on a chip, along with decreasing clock cycles to obtain high operating frequency [1, 2]. The design of a compact circuit on a chip is advantageous in terms of noise but creates various problems in terms of reliability [3, 4]. Researchers agree that reduction in hardware size will increase hardware failures in future processors [5]. Thermal cycling, dielectric behavior and biasing of digital integrated circuits have also caused many transient and permanent faults [6].

In order to deal with the above-mentioned problems, the concepts of self-checking and fault tolerance have been introduced. A system will be fault secure if it remains unaffected by a fault or if it indicates a fault as soon as it occurs [7]. A system will be self-testing if it produces a non-coded output in response to every generated fault [8]. A system will be totally self-checking (TSC) if it is both faults secure and self-testing [7]. The concept of TSC is used in different applications, like self-healing networks [9], self-checking arithmetic logic units (ALUs) [10], etc.

However, the increasing complexity of digital system also increases the cost of repairing [20]. Therefore, the existence of fault detection without recovery cannot fulfill the demand of current digital system. Thus, the built-in self-repair (BISR) is becoming the need of current semiconductor technology [21].

Adder is one of the essential elements present in almost all digital devices [22, 15]. Therefore, the introduction of BISR in adder can play a vital role in digital industry. Many approaches have already been proposed for self-checking adder design. However, very few work has been found related to the self-repairing with reduced hardware overhead. Most of the self-checking approaches require re-execution of instructions for fault recovery [11]. However, the re-execution process affects system performance because all operations connected directly or indirectly to the faulty module should be re-executed. Furthermore, re-execution cannot guarantee fault recovery, especially if there is a permanent fault or wear-out problem. Therefore, on-line detection and self–repairing is required in a highly dependable system architecture [12].

B. **Research Motivation**

The common design problem in various approaches for self-checking adders is fault propagation due to carry. Such a fault can misdirect the system, preventing it from detecting the particular faulty region in a module because a propagated error can make the correct region of a module appear to be faulty, as well.

A duplex system is shown in Fig. 1, in which each module has three full adders. Let us assume that an error occurs in the carry generation portion of the first full adder. The propagated error will indicate the second and third full adders as faulty modules, whereas the first full adder will be deemed a fault-free module. Therefore, in order to achieve recovery, we will repair the second and third modules, and hence, the faulty module is not recovered. Similarly, if we detect that a module having five full adders generates faulty output, then by most of the current approaches we cannot detect which of the five full adders is faulty. Hence, if a fault is indicated in any region of a large module, we need to

replace the whole module in order to accomplish the recovery process. This kind of replacement approach will waste resources because the properly functioning block has also been replaced.

We observed that most design approaches deal with error detection, but little work has currently been done on the self-repair side. The general consideration for designing a self-checking adder is that the recovery process can be feasible after detection of faults. However, the cost for self-repair is too high to be adoptable for most self-checking approaches. If we consider DMR and TMR, then instead of replacing a single module, we have to replace the whole system to achieve recovery. This will introduce a huge area overhead of more than 300% to the actual system, as well as an increase in power consumption. Our proposed design in this thesis considers both the facts of self-checking and the recovery. Because of the fault localization property present in our design, the system is only required to replace the faulty region, without disturbing the whole system.

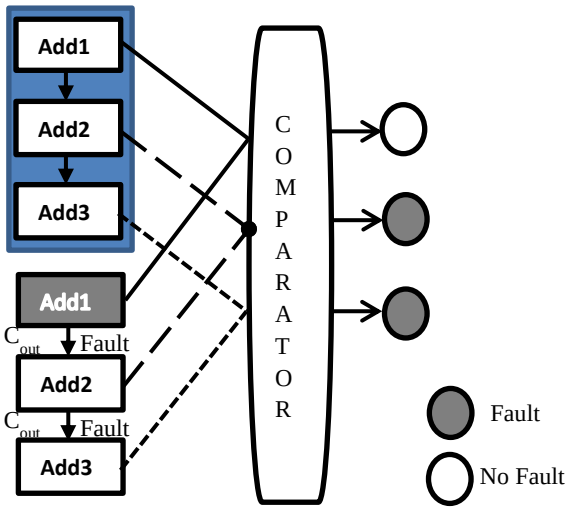


Figure 1: Fault propagation through carry

C. **Contribution**

In this research, a self-checking full adder has been proposed which can detect the fault based on its internal functionality and independent of the propagated carry. Even the error propagated through carry will not create a fault indication in subsequent full adders because all full adders are self-checking with respect to their individual functionality. We then proposed a self-repairing adder by using the fault localization property present in our proposed self-checking full adder. The designed self-repairing adder can provide reliable output for more than one fault, with reduced area overhead. This is because, instead of replacing the whole system of adders, we replace the particular faulty full adder only. The main contributions of this thesis are as follows:

1) All full-adders present in adder module are self-checking with respect to their individual functionality.
2) Self-checking Carry Select Adder (CSeA) with fault localization whose area overhead almost similar to the conventional CSeA without self-checking.
3) A reliable carry propagation chain has been designed using dedicated self-checking adder for replacements
4) Possibility of recovering multiple fault at run-time
5) Number of faults recovered is dependent on the adder size

D. **Thesis Organization**

The thesis is organized as follows. Chapter 2 discusses the previous work and common design problem (i.e., the fault propagation problem due to carry). In chapter 3 we discuss the basic principle, fault coverage of our proposed design approach for a self-checking full adder and its application to CSeA. In chapter 4 we present the concept and comparison of our proposed self-repairing adder

approach. The comparison in terms of area and reliability covers in chapter 5. Our research work is concluded in chapter 6.

II. RELATED WORKS AND LIMITATIONS

Self-checking systems require redundancy, which can be broadly classified as time- and hardware-based redundancy. We will discuss both of them briefly, along with related works, in this section.

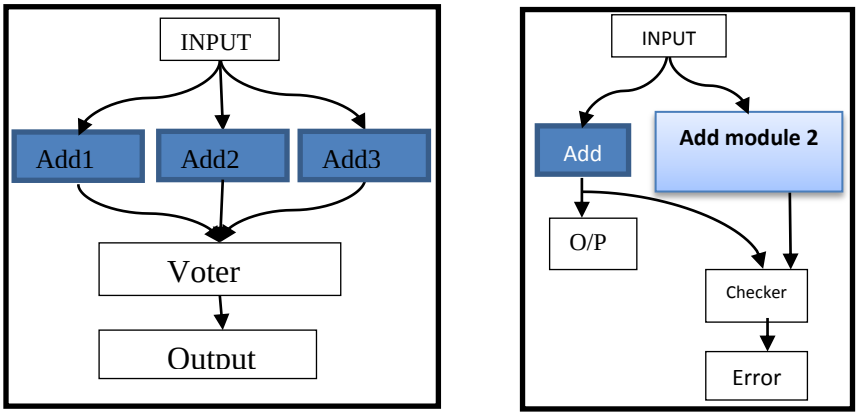
A. Hardware Redundancy

The basic idea for this approach is to use more than one hardware to produce either the same, inverted or coded output. The comparison between the outputs of the actual and the redundant hardware will be used to indicate the fault. This approach can be further classified into duplication of the existing hardware and output predictor block.

1. Duplication of Hardware

In the duplication of hardware scheme, the two most commonly used approaches are double modular redundancy (DMR) and triple modular redundancy (TMR) shown in Fig. 2. In a duplex system (i.e., DMR) the same hardware is repeated once, whereas TMR requires three modules with which to vote to determine the final output.

The major problem with this concept is that instead of detecting faults in individual modules, it can detect the overall fault of a system. This kind of checking creates many complexities in terms of fault recovery, because we cannot detect which individual block is faulty. The other drawback of this approach is that straightforward implementation will create an area overhead of more than 100% [3, 13]. Secondly, it cannot detect a common mode failure in which both modules experience the same set of errors [14].



(a) (b)

Figure 2: Hardware Redundancy: (a) TMR (b) DMR

The advantage of TMR over DMR is the probability of obtaining reliable output. It can easily be recognized that by using the TMR technique, the area overhead will increase up to 300% compared to the original design without self-checking. The second problem with this kind of design approach becomes apparent when two modules have the same faulty result at the same time, and then the correct module is treated as faulty because the output is dependent on a voter circuit that decides on the basis of majority [10].

In a CSeA design, two adders are working in parallel for a complemented value of initial carry-input, C_{in}, and the final output will depends on the actual value of C_{in}. The benefit of having two adders at a time was realized by Vasudevan et al. [15], and they introduced a fault-tolerant capability without excessive hardware overhead. However, the problems of common mode failure and a high area overhead requirement for fault recovery still remained unsolved. The fault in carry generation is allowed to propagate and cannot be detected with its first occurrence. Hence, this approach also suffers in terms of determining the defective full adder modules.

2. Concurrent Error Detecting Codes

This approach also requires an extra functioning module and is similar to DMR. However, in contrast to duplicated hardware approach, the extra module will generate encoded output with encoded inputs, as shown in Fig. 3. The resulting output from normal module will be encoded further and compared with the encoded output of the extra module. The advantage of this approach is diversity which can encounter the problem of common mode failure.

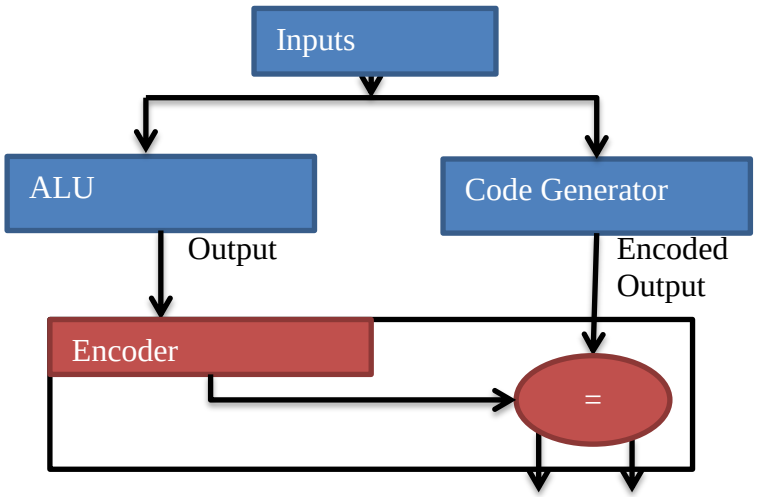


Figure 3: Concurrent Error detection scheme

B. **Time Redundancy**

The basic concept of time redundancy is that the same hardware will perform a single operation in different intervals of time, and we can detect an error by comparing the two outputs obtained at different time instances.

Khedhiri et al. [13] designed a self-checking full adder circuit based on the concept of time variation. The single full adder will work such that it is used twice to perform the same computation by following different logical paths, and the comparison of the final results will indicate the presence of a fault. The

key features of the authors' design are diversity and a decrease in area overhead by using different paths for a single operation in each time interval. Moreover, in order for system performance or speed to remain unaffected, it has been argued that the result will be propagated after the first computation. Hence, if the first computed result is faulty, then before detecting the fault, that result will be used for other computations. The propagated fault will also cause faulty results in other modules.

III. SELF-CHECKING ADDER DESIGN

A. **Basic Principle**

The presence of carry propagation chain in adder cannot be ignored during self-checking and fault localization. In order to avoid the problem of fault propagation due to carry one possible approach is to adapt carry-free adder. However, the complexity and area overhead of carry free adder without self-checking is almost equal to the duplex system. Another approach is to make self-checking independent of the propagated carry. We utilize the following relations to achieve the goal of fault localization:

1) The Sum and Carry-out bit will be equal to each other when all three inputs are equal.
2) The Sum and Carry-out bit will be complemented when any of the three inputs is different.

By using the above two relations, a full adder can be self-checked with only the expense of an equivalence tester (E_{qt}), as shown in Fig. 4. The purpose of the equivalence tester is to check the equivalence of all inputs. Hence, apart from the Sum and Carry bits calculation, we have to compute logic for the equivalence tester, and the expression for that purpose is in Eq. (3.3). The functional block E_{qt} has been designed to produce an active low output so that we can use the fault-secure Exclusive-NOR logical function (XNOR) gate for comparison.

$$\text{Sum} = \text{A} \oplus \text{B} \oplus C_{in} \tag{3.1}$$

$$\text{C}_{out} = A \bullet B + C_{in} \bullet (A + B) \tag{3.2}$$

$$Equivalence\ Tester\ (E_{qt}) = \overline{(\bar{A}\,\bar{B}\overline{C_{in}} + ABC_{in})} \tag{3.3}$$

$$Error\ (E_f) = Sum \odot C_{out} \odot E_{qt} \tag{3.4}$$

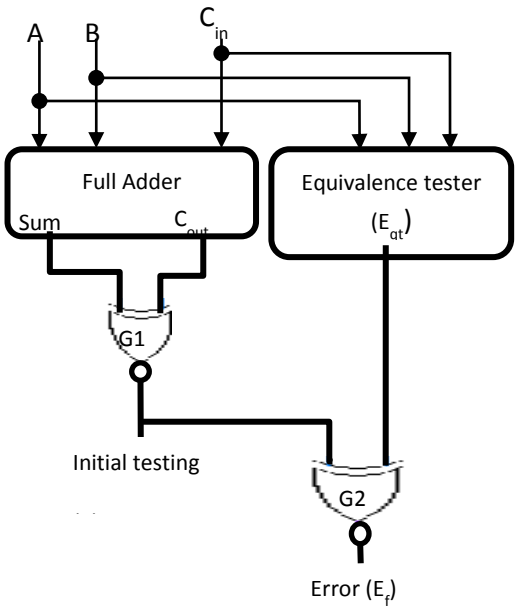


Figure 4: Proposed self-checking full adder

The final fault is computed by using two XNOR gates. The purpose of the first XNOR gate (G1) is to check whether the Sum and C_{out} bits are equal or complemented. We need a second XNOR gate (G2) because of the previously mentioned observation that Sum and C_{out} will always complement each other except when all inputs are equal. Thus, the output of G1 will indicate the equality or difference of Sum and C_{out}, and G2 will verify the output of G1 by comparing it with an equivalence tester, and thus generate the final error indication. When E_{qt} is zero, the output of G1 and G2 should be logic 1 and 0, respectively. On the other hand, if E_{qt} indicates logic 1, then both XNOR gates should generate logic 0 (i.e., I_t =0 and E_f =0), and in any other case, a fault will be indicated.

B. **Self-Checking Carry Select Adder (CSeA)**

1. Proposed Approach

Carry Select Adder (CSeA) is one of the most common types of adder which pre-compute Sum bits using two parallel blocks of Ripple Carry Adder (RCA) with complemented values of initial C_{in} and the final Sum bit will be generated after receiving the actual value of C_{in}. The presence of two parallel RCA is advantageous for introducing Self-checking ability with the reduced hardware overhead. The self-checking CSeA shown in Fig. 5 was proposed by Vasudevan et al. [15], where two adders and a 2-pair-2-rail checker, along with some combinational logic, are used for self-checking.

We design a self-checking CSeA with single adder, assuming the initial C_{in} to be zero and where the individual full adder has been replaced by our proposed self-checking full adders. We then generate Sum and Carry output for the initial C_{in} equal to one using the following relation, where S_j^i and C_j^i represent the j^{th} position of the Sum and Carry bits, and i is either 0 or 1, indicating the value of the initial C_{in}:

1) S_0^1 is always a complement of S_0^0, i.e. $S_0^1 = \overline{S_0^0}$.
2) S_1^1 depends on S_0^0, S_1^0:

   If (S_0^0=0) **then** $S_1^1 = S_1^0$;

   If (S_0^0=1) **then** $S_1^1 = \overline{S_1^0}$.

3) C_1^1 depends on C_1^0, S_0^0 and S_1^0:

   If (C_1^0=1 **OR** $S_0^0 \bullet S_1^0$=1) **then** C_1^1= 1;

The above relationships can be summarized in Eq. (3.5), Eq. (3.6) and Eq. (3.7). An exclusive OR (XOR) operation is performed in Eq. (3.6) to generate S_1^1 because S_1^1 is not always a complement to S_1^0. Moreover, since both C_1^0 *and* $S_0^0 \bullet S_1^0$ cannot equal 1 at the same time, we used the XOR gate in Eq. (3.7) because a self-checking XOR gate was presented by Belgacem et al. [16]. The two-bit proposed self-checking CSeA is shown in Fig. 6. The design consumes less area compared to the self-checking CSeA approach of Vasudevan et al. [15], shown in Fig. 5.

$$S_0^1 = \overline{S_0^0} \tag{3.5}$$

$$S_1^1 = S_0^0 \oplus S_1^0 \tag{3.6}$$

$$C_1^1 = C_1^0 \oplus (S_0^0 \bullet S_1^0) \tag{3.7}$$

Thus, for arbitrary number of bits we found that except for the least significant bit (LSB), S_j^0 and S_j^1 has the following relation:

Except for the LSB, the Sum bit computed when carry-in equals 0 will be a complement to the corresponding Sum bit with carry-in equal to 1, only when all the lower Sum bits are equal to logic 1.

In general we can say that:

$$\boldsymbol{If}\ (S_1^0 \bullet S_2^0 \bullet S_3^0 \ldots\ldots S_{(i-1)}^0 = 1)$$

$$\boldsymbol{Then}\ (S_j^1 == \overline{S_j^0});$$

$$\boldsymbol{Else}\ (S_j^1 == S_j^0)\ ;$$

where ***j*** indicates the bit position. Thus, in order to design an n-bit CSeA, the generalized Boolean equations have been shown in Eq. (3.8), Eq. (3.9) and Eq. (3.10). The design module shown in Fig. 7 will be used to extend the 2-bit CSeA design to an n-bit CSeA design. All the intermediate Sum bits between the LSB and final C_{out} can be generated using the module shown in Fig. 7. The LSB and module for final C_{out} (MOFC) design in Fig. 6 can be used to complete the n-bit self-checking CSeA.

$$S_0^1 = \overline{S_0^0} \tag{3.8}$$

$$S_n^1 = S_n^0 \oplus \left(S_1^0 \bullet S_2^0 \bullet S_3^0 \ldots\ldots S_{(n-1)}^0\right) \tag{3.9}$$

$$C_n^1 = C_n^0 \oplus \left(S_1^0 \bullet S_2^0 \bullet S_3^0 \ldots\ldots S_{(n-1)}^0\right) \tag{3.10}$$

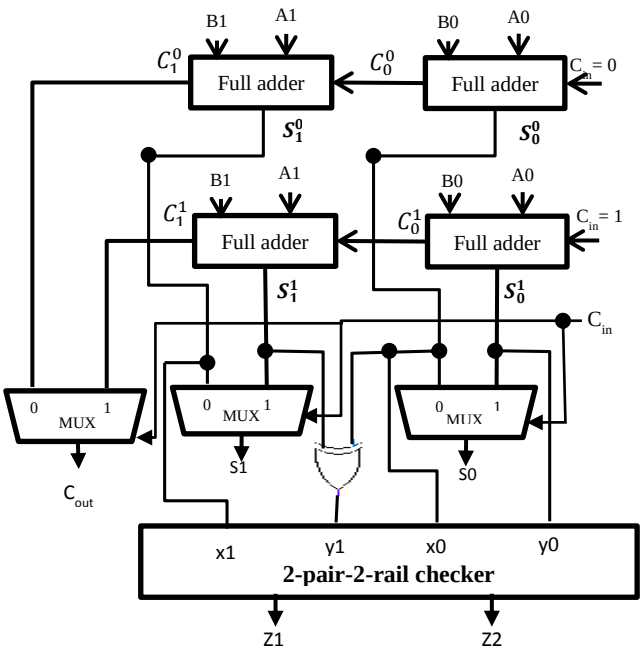


Figure 5: Self-testing carry-select adder [15]

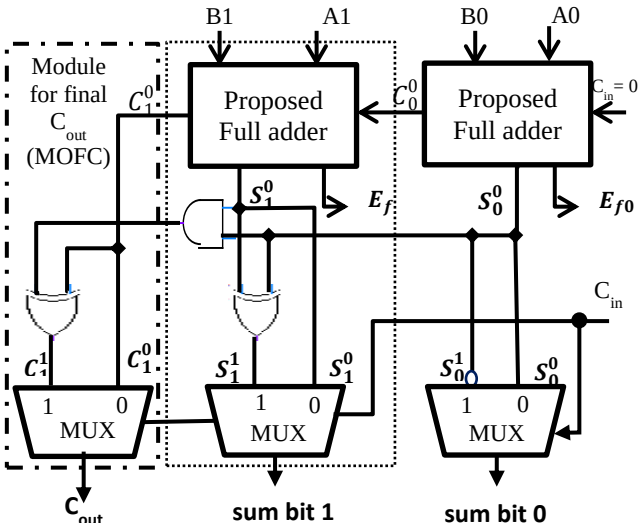


Figure 6: Proposed design of two-bit self-testing carry-select adder

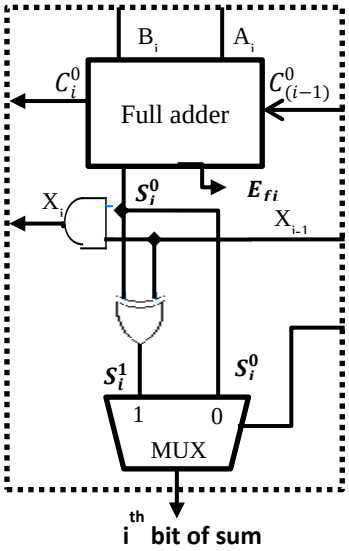


Figure 7: Designed module required for n-bit extension

2. Correction of Previous Approach

Vasudevan et al. in [15] presented one Self-checking carry-select adder (CSeA) with reduced area overhead scheme. The proposed design seemed to be promising for self-checking CSeA, however, we find that the claim for self-

checking is only valid for 2-bit CSeA, and the 6-bit CSeA shown in the paper cannot provide self-checking. We argue the failure of their design model and present a correct self-checking CSeA design based on two-rail encoding. We show that the transistor overhead of the correct model is higher than the one claimed by Vasudevan et al. [15].

The property used by Vasudevan et al. [15] for a pair of full adders can be generalized as follows:

The relation between sum bits calculated with identical inputs is only dependent on the carry-input, and for complemented values of carry-input, we will obtain complemented sum bits (keeping other pairs of input bits identical).

For example, if the sum bits S_i^1 and S_i^0 are generated by using intermediate carry C_i^1 and C_i^0, respectively, then according to the above-mentioned statements:

$$\boldsymbol{If}\ (C_i^1 == \overline{C_i^0});$$

$$\boldsymbol{Then}\ S_i^1\ is\ equal\ to\ \overline{S_i^0}$$

Where, i indicates the bit position, while 1 and 0 in the superscript represent the initial value of C_{in}. If we analyze the CSeA design, then the above statements are only valid for the least significant sum bit of a particular CSeA block. This is because the first full-adder in every CSeA block is the only one that will get the complemented values of the carry-input. The remaining full adders will depend on the propagated carry, which may or may not be complementary to each other. Therefore, we cannot say whether the generated sum bits, other than the first full adder, will be inverted to each other or not.

The possibility of having equal values of propagated carry by the two corresponding adders in a CSeA was neglected by Vasudevan et al. Therefore, the approach in [15] fails for CSeA with more than two bits, as shown in Fig. 9 [15]. Note that the initial carries, C_1 and C_2, always complementary to each other because of the design requirement of CSeA, while the intermediate carries, C_a and C_b, may or may not be complementary to each other, depending on the conditions of carry propagation. Thus, S_a and S_b will not always be complementary to each other. Therefore, comparing S_a and S_b directly using 2-pair-2-rail-checker (TTRC) will give the wrong indication of faults. Even if there is no fault, the TTRC will indicate a fault. Since the problem in their approach starts from C_a and C_b, we do not discuss the intermediate carries C_x and C_y. Let us consider the binary addition of 3-bit numbers as illustrated in Fig.8. It can easily be seen from Fig. 8(a) and (b) that the most significant bits may or may not be complementary to each other.

Figure 8: Examples of 3-bit addition (a) $S_2^0 = S_2^1$ (b) $S_2^0 = \overline{S_2^1}$

A carry-select adder pre-computes sum bits using two parallel ripple-carry adders (RCAs), with complemented values of the initial C_{in}, and the actual value of the C_{in} will be used to determine the final sum bit. Vasudevan et al. utilized both RCAs to obtain the complementary behavior of the corresponding sum bits. However, it is possible to perform a logical operation such that one of the RCA blocks should always provide inverted sum bits with respect to the

opponent block for checking purposes only. This will provide a more simplified and systematic design, which can be extended easily.

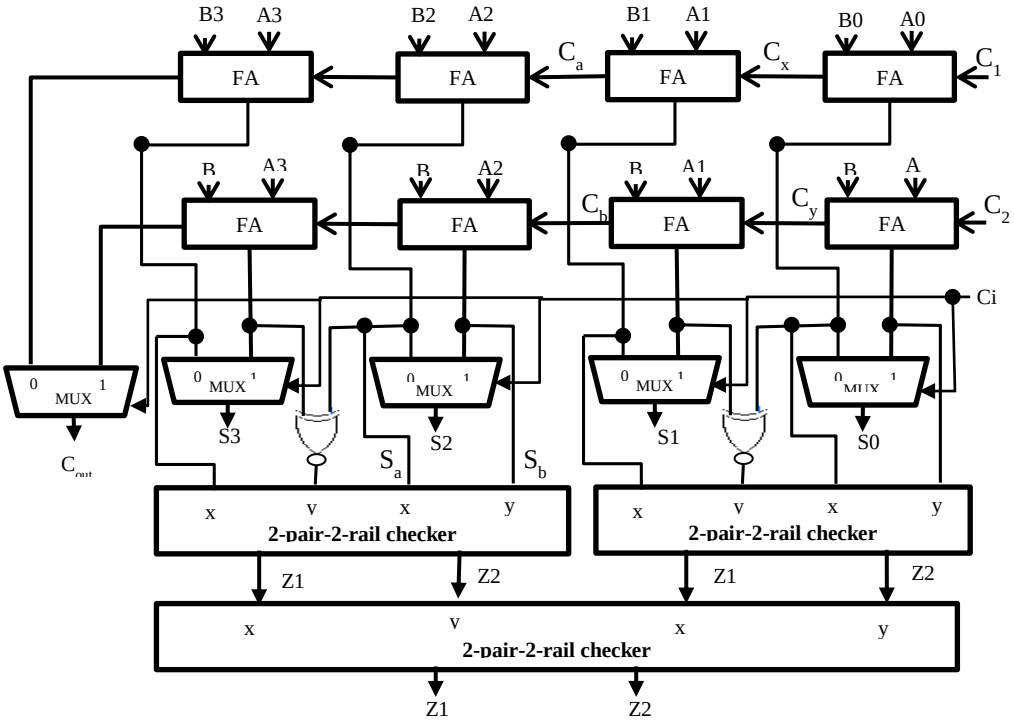


Figure 9: Faulty design of 4-bit self-testing carry-select adder [15]

In this theis, we will discuss only one possible way in which the sum bits calculated at initial C_{in} = 0 are altered, such that they become complementary to the sum bits calculated at an initial C_{in} = 1 for comparison.

After close observation, we found that:

Except for the least significant bit, the sum bit computed when initial carry-in equals 0 will be complementary to the corresponding sum bit with an initial carry-in equal to 1 only when all the lower sum bits are equal to logic-1.

In general, we can say that:

$$\boldsymbol{If}\ \left(S_1^0 \bullet S_2^0 \bullet S_3^0 \ldots\ldots S_{(i-1)}^0 = 1\right)$$

$$\boldsymbol{Then}\ S_i^0\ \textit{is equal to}\ \overline{S_i^1};$$

$$\boldsymbol{Else}\ \ S_i^0\ \textit{is equal to}\ S_i^1\ ;$$

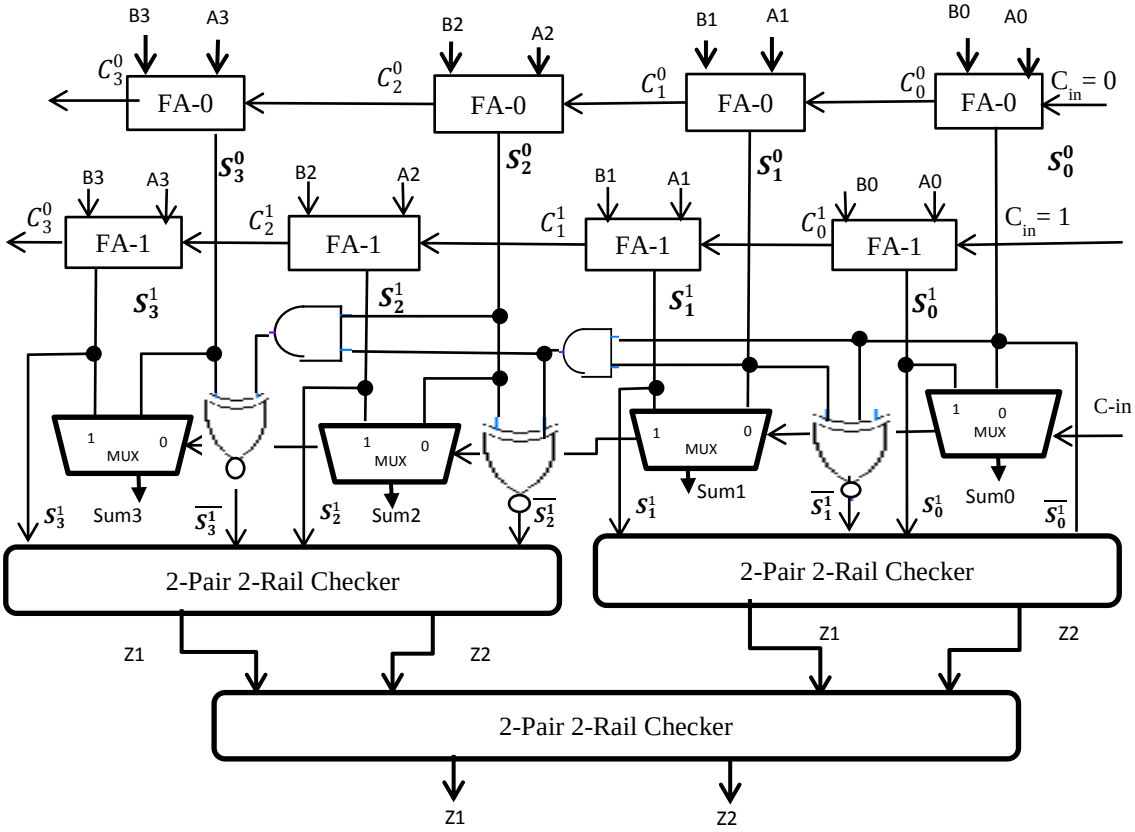


Figure 10: Corrected design of self-testing carry-select adder with two rail encoding

Thus, in order to apply TTRC for n-bit CSeA, we need to have S_n^1 along with its complement, and S_n^0 will not always equal to $\overline{S_n^1}$. If all the (n-1)th sum bits at C_{in}=0 are equal to logic-1, then the value S_n^0 is equal to the complement of S_n^1.

In other cases, if any of the (n-1)th sum bits at C_{in}=0 are equal to logic-0, then we take the inverse of S_n^0, so it equals to the complement of S_n^1. Therefore, in Eq. (3.11) we performed an XNOR operation between S_n^0 and the product of all lower sum bits computed at the initial C_{in}=0, such that the resultant K will always be equal to $\overline{S_n^1}$. The design module shown in Fig. 10 will be used to implement the 4-bit self-checking CSeA.

$$K = \overline{S_n^0 \oplus \left(S_{(n-1)}^0 \ldots \ldots S_3^0 \bullet S_2^0 \bullet S_1^0\right)} \tag{3.11}$$

We applied the same technology and implementation used by Vasudevan et al. [15] for comparison. A standard complementary metal-oxide semiconductor–based AND gate with 6 transistors was used for area computation and the transistor count for full-adder, multiplexer (MUX), XNOR gate and TTRC was taken from Vasudevan et al. [15], as given below:

- Full adder – 28 transistors;
- MUX – 12 transistors;
- XNOR – 10 transistors;
- TTRC – 8 transistors.

For an n-bit self-checking CSeA, we required (n-2) number of AND gates, (n-1) number of XNOR gates, (n+1) number of MUX, (2n) number of full adders and (n-1) number of TTRC, respectively. We can see from Table 1 that the difference in transistor overhead for 4- to 64-bit self-checking CSeAs varies from 22 to 682, compared to the faulty self-checking CSeA design by Vasudevan et al. [15]. Moreover, the transistor overhead of the corrected self-checking CSeA, as compared to CSeA without self-checking, was found to be

23.2% to 34.5%, whereas in the faulty approach presented by Vasudevan et al. [15], the overhead was 15.49% to 18.84%.

Table 1: Comparison of CSeA with Self-Checking CSeA Before and After correction of Vasudevan et al. [15]

No. of bits	CSeA without self-checking	Vasudevan et al. faulty design [15]			Corrected self-checking CSeA		
	Transistor required [15]	Trans. required	Trans. overhead	% Trans. overhead	Trans. required	Trans. overhead	% Trans. overhead
4-bit	284	328	44	15.49%	350	66	23.2%
6-bit	420	490	70	16.66%	534	114	27.14%
8-bit	556	652	96	17.26%	718	162	29.14%
16-bit	1100	1300	200	18.18%	1454	354	32.18%
32-bit	2188	2596	408	18.65%	2926	738	33.73%
64-bit	4364	5188	824	18.88%	5870	1506	34.5%

C. **Fault Coverage**

The condition for fault coverage of our design has been summarized in Table 2. The logic-high of the final error (E_f) will indicate a fault. We can easily see that the designed approach guarantees self-checking only when any one of the Sum, Cout or E_{qt} lines becomes faulty. If two of them have a fault at the same time, then that fault cannot be detected. The assumption of having a single fault at one time becomes valid because the fault-secure property assumes that between two consecutive faults, we have enough time to detect the error at its first occurrence [16]. Moreover, unlike other techniques, like DMR and TMR, etc., we are testing a very small module of a single full adder. The probability of having two faults at one time in a small module is much lower than with a large module.

For example: A=1, B=0 and C_{in}=1 are three respective inputs in our proposed full adder. The corresponding outputs for these inputs without fault are: Sum=0, Carry=1 and E_{qt} =1. Let us assume that the Sum bit flips from 0 to 1, and the outputs, after the fault, will thus become Sum=1, Carry=1 and E_{qt} =1. This condition has been indicated as a fault in Table 2.

Table 2 Conditions For Fault Coverage

CONDITIONS	STATUS
IF $((E_{qt} == 0)$ AND $(\mathbf{Sum} == C_{out}))$	NO FAULT
IF $((E_{qt} == 1)$ AND $(\mathbf{Sum} == \overline{C_{out}}))$	NO FAULT
IF $((E_{qt} == 0)$ AND $(\mathbf{Sum} \neq C_{out}))$	FAULT
IF $((E_{qt} == 1)$ AND $(\mathbf{Sum} \neq \overline{C_{out}}))$	FAULT

With our proposed approach for a self-checking adder, a fault generated in any full adder can be detected individually. Even if the generated carry has a fault, it will not create a fault indication in subsequent full adders because all the full adders are self-checking with respect to their individual functionality and are independent on their carry input. Hence, the faulty propagated carry will not create a problem in self-checking in any subsequent full adder. The benefit of this approach is that if we want to recover from the fault, then instead of replacing all full adders present in the complete module, we replace only particular full adders.

Moreover, Sum and C_{out} of a full adder are not sharing any logic, so the problem of transient faults can be overcome. For example, consider a single full adder in which both the final Sum and C_{out} share the Sum bit of the first half-adder. Let a, b and C_{in} be the three input bits in the full adder, with values equal to 0, 0 and 1, respectively. With these inputs the correct outputs for the final Sum and C_{out} are 1 and 0. Assume that the Sum bit of the first half adder ($a \oplus b$)

flips from 0 to 1. Then this bit flipping can generate a fault in both Sum and C_{out} because of logic sharing and can thus create a problem in detection. Therefore, in our proposed idea, the Sum and C_{out} modules work without logic sharing, as shown in Fig. 11, and any fault occurring within the internal logic of an individual module will make that particular module faulty, and thus can be detected easily on comparison.

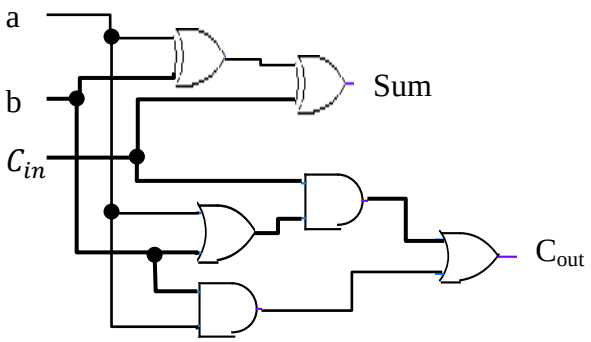
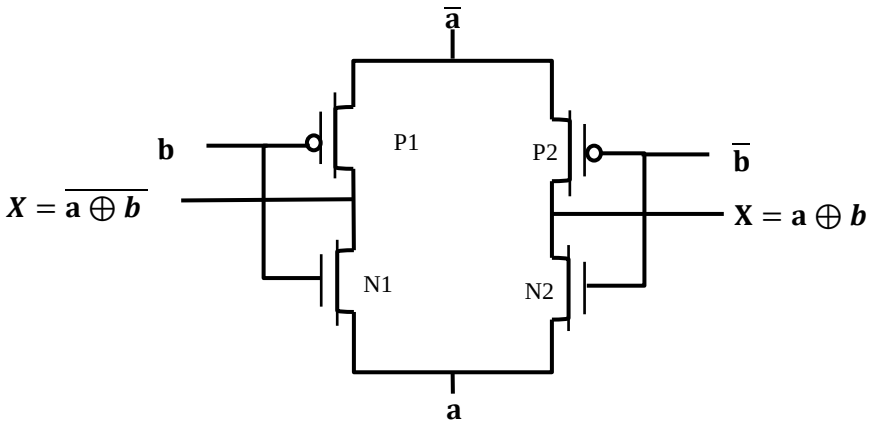
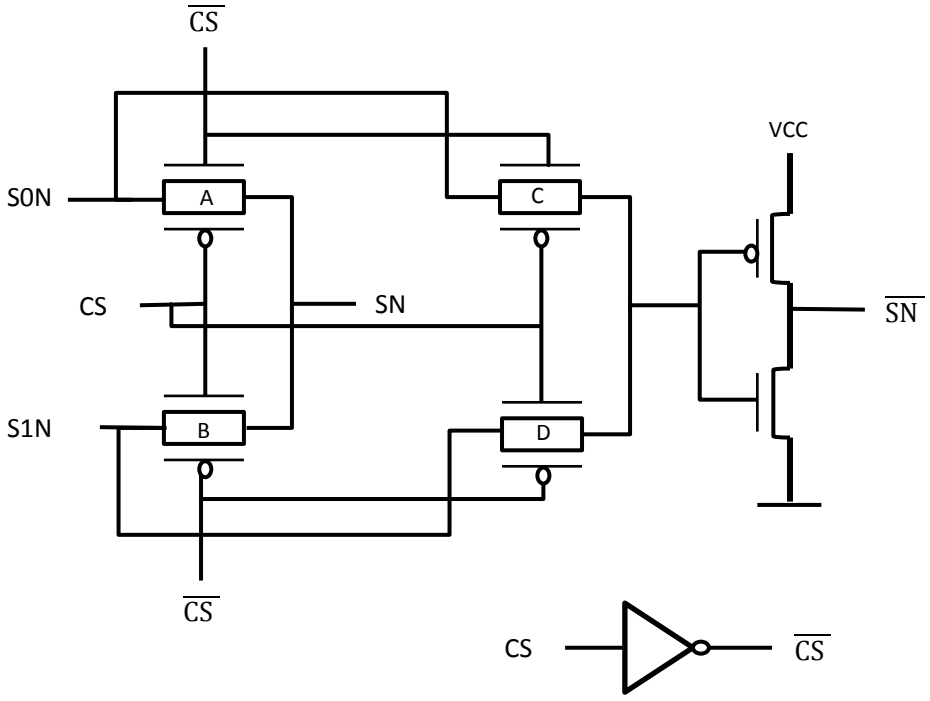


Figure 11: Full-adder without logic sharing

In addition to a self-checking full adder, we need to have a self-checking XOR gate and multiplexer (MUX). The self-checking XOR gate was proposed by Belgacem et al. [16] using a pass-transistor–based differential pair as shown in Fig. 12(a). The self-checking MUX design, however, was presented by Vasudevan et al. [15] in which the complementary outputs SN and $\overline{SN}$ will be used to detect the presence or absence of faults and is shown in Fig. 12 (b).



(a)



(b)

Figure 12: Self-Checking (a) XOR gate [16] (b) MUX [15]

IV. SELF-REPAIRING ADDER DESIGN

A. **Previous Design Approaches**

TMR is a well-known approach for obtaining reliable output with more than 300% area overhead. However, reliability can only be assured for a single faulty module at a time. This is because the faulty module has not been replaced with a correctly functioning module, and as a result, a triple modular system changes into a duplex system. Koal et al. [18] proposed a self-replacing approach for TMR, but the resulting design required more than 500% area overhead along with at least 5 clock cycles to replace the particular faulty module. Moreover, system complexity due to a switching mechanism between input and output rendered the design far beyond practical implementation.

Fazeli et al. [11] proposed a self-checking and self-reliable adder using the concept of narrow width value (NWV). The overall approach deals with a processor design property whereby most of the data coming into the ALU consists of NWVs. Therefore, the individual arithmetic and logic operations can be split into two parts: one is used to perform normal operations while the unused part is used to provide redundant computation, like a duplex system. Consider the case of a 64-bit adder in which (most of the time) a 32-bit adder will be used for addition while the other 32-bit adder remains unused because of NWV. This unused 32-bit adder has been made equivalent to the other 32-bit portion. Thus, a duplex system with minimum hardware overhead is obtained. Furthermore, the concept of TMR has also been used to provide reliable output. Fazeli et al. [11] proposed that a third 32-bit adder module for TMR can be obtained from the multiplier block, because the module of the multiplier is a combination of add and shift operations.

The problems associated with this approach led us to propose our design. First, the reason for using NWV in processor design is to simplify computational complexity and to reduce energy consumption [11]. However, adaptation of their approach will remove both those benefits. In addition, the problem of common mode failure and fault propagation due to carry has not been considered. Moreover, the possibility of two faulty modules can result in failure of their approach. Furthermore, correct output on a single fault will only be achieved when both the operands are narrow width values. In other cases, the proposed adder can only detect the error using a duplex concept. Thus, beyond hardware complexity and overhead due to the MUXs, the system reliability will only depend on the NWV, and according to their calculation based on the ARM processor, the probability of having NWV in addition is only 56% [11]. Therefore, there is only a 56% probability of obtaining reliable output for a single fault. This will raise the problem of system diagnoses, as to whether the output is reliable or not, along with precautions if the system fails to recover. Also, the MUXs are used without considering their reliability issues.

B. **Design Challenges and Solutions**

Along with the area, power consumption and other limitations for a self-reliable adder, there is one important challenging factor that is also related to carry propagation. Although our proposed self-checking adder is independent of fault propagation due to carry, for fault recovery at run time, the propagated carry is crucial. If a fault is detected in any full adder, then because of fault propagation due to carry, we cannot trust the output of the corresponding full adders. Therefore, if we replace the particular faulty module, we need to re-execute all addition processes to get true and reliable output. However, this whole process will increase processing time.

In our proposed design, we provide a solution for this problem by designing a reliable carry propagation chain using a dedicated self-checking adder for replacements. If both the adders are faulty at the same time, this will be indicated as a critical situation. It is possible to have two faults at a time, like the above-mentioned problem, but there is a low probability of the fault being at the same adder position.

C. **Proposed Self-Repairing Adder**

The design approach requires two proposed self-checking adders working at one time on the same input bits, as shown in Fig. 13(a). The operation of the proposed design is such that one adder behaves as a normal adder (indicated by Adder-0) while the second adder will work for recovery at run time (indicated by Adder-0e). The suffix “e” indicates an “extra” adder used for fault recovery. The MUX will only be dependent on the fault of the normal adder (i.e., Adder-0). If any fault occurs in the normal adder, then the MUX will export the Sum and Carry-out from the extra adder module (i.e., Adder-0e), as shown in Fig. 13(b). If both adders become faulty, this will be indicated as a worst situation. Also, there is a very low possibility of more than one fault in a single full adder because of a negligible area, as opposed to the complete adder module. Therefore, multiple faults in the individual full adder have not been considered in this research.

The final Carry-out obtained from the MUX will be fed to both full adders present in order to compute the next Sum bit, as shown in Fig. 13(a). The MUX in Fig. 13(a) consists of two internal MUXs, as shown in Fig. 13(b). Both MUXs are designed according to the proposed self-checking MUX design by Vasudevan et al. [15].

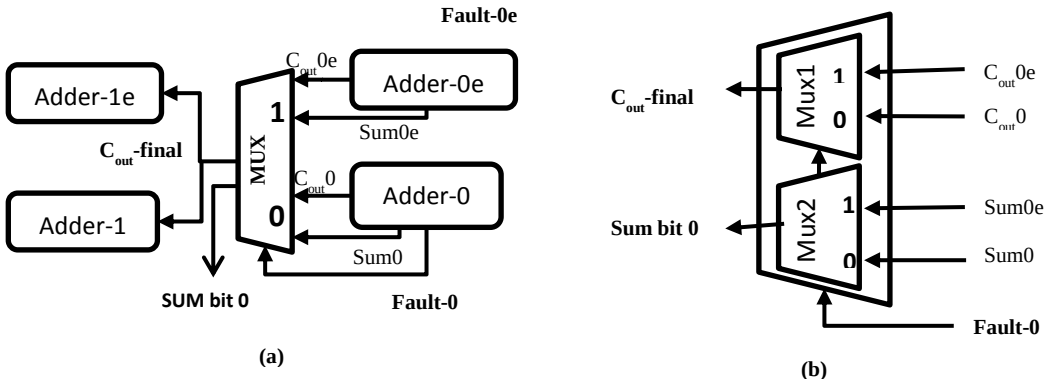


Figure 13: (a) Proposed design for self-repairing adder (b) MUX structure

V. AREA AND RELIABILITY COMPARISON

A. **Area Comparison**

The area overhead can be compared in different ways, such as transistor count, gate count and technology dependence. In this section, we will compare our proposed self-checking full adder with DMR and TMR. Also, our designed self-checking CSeA will be compared with the conventional CSeA and self-checking CSeA from Vasudevan et al. [15]. The area overhead in both cases was computed in terms of transistor count. The transistor count for implementing a full adder and MUX was taken from Vasudevan et al. [15] while the transistor count for the self-checking XOR gate was taken from Belgacem et al. [16]. The final transistor count for individual modules is shown in Table 3.

Table 3: Transistor Count for Individual module Implementation

Modules	Required Transistors for Implementation	
AND	6	
XOR	4	[16]
MUX	12	[15]
ADDER	28	[15]
EQUIVALENCE TESTER (E_{qt}) (Eq. 3)	12	
CHECKER (2-XOR)	8	
MODULE FOR FINAL C_{OUT} (MOFC) (shown in Fig. 6)	16	

1. Area Compared To DMR And TMR

Single-bit adder implementation using DMR required two adders and a single XOR gate for comparison. For TMR, we need three adders and one majority voter for selecting the correct output. In order to compute transistor count for voter circuitry, we used the fault secure voter design proposed by Kshirsagar and Patrikar [17].

The comparison result of our designed self-checking full adder with DMR and TMR is shown in Table 4. We see that our designed self-checking full adder requires an overhead of 20 transistors compared to a full adder without self-checking, whereas DMR and TMR have an overhead of 32 and 96 transistors, respectively. Hence, our designed self-checking full adder requires 25% fewer transistors than DMR and 158% fewer than TMR. Moreover, our self-checking full adder, without creating a time penalty, has 20.8% more area efficiency than a self-checking full adder using the time-redundancy approach of Khedhiri et al. [13].

Table 4: Comparison with Time-Redundancy based Self-Checking Adder, DMR and TMR

	Individual Transistor Count		Total # of Transistors	Transistor overhead
Proposed full-adder	ADDER E_{QT} CHECKER	28 12 8	48	20
DMR	ADDER XOR	56 4	60	32
TMR	ADDER Voter [17]	84 40	124	96
Self-checking adder [13]	-	-	58	30

2. *Area Compared To self-checking CSeA*

The transistor count for our proposed self-checking CSeA with multiple bit data is presented in Table 5. The transistor count for a conventional CSeA with and without self-checking was presented by Vasudevan et al. [15]. It is seen from Table 6 that our designed self-checking CSeA has only negligible area overhead compared to a normal CSeA design and also requires a 15% to 16% lower transistor count compared to the self-checking CSeA proposed by Vasudevan et al. [15].

A graphical representation is shown in Fig. 14. We can see that our proposed self-checking CSeA design shows the same trend compared to a CSeA without self-checking, whereas the percentage increase in area overhead of the self-checking CSeA proposed by Vasudevan et al. [15] shows considerable increment compared to a normal CSeA.

Table 5: Transistor Count for Different Data Size

	4-bit	6-bit	8-bit	16-bit	32-bit	64-bit
AND	3	5	7	15	31	63
XOR	3	5	7	15	31	63
MUX	4	6	8	16	32	64
Full Adder	4	6	8	16	32	64
Equivalence Tester	4	6	8	16	32	64
Checker (Pass Trans. Based)	4	6	8	16	32	64
MOFC	1	1	1	1	1	1
Total Transistor Required	286	426	566	1126	2246	4486

Table 6: Comparison with Other Approaches Using CSeA

Number of bits	CSeA without self-checking	Our proposed CSeA design		CSeA Design proposed by [15]	
	Transistors required [15]	**Transistors required**	**Transistor overhead**	**Transistors required**	**Transistor overhead**
4-bit	284	286	2	328	44
6-bit	420	426	6	490	70
8-bit	556	566	10	652	96
16-bit	1100	1126	26	1300	200
32-bit	2188	2246	58	2596	408
64-bit	4364	4486	122	5188	824

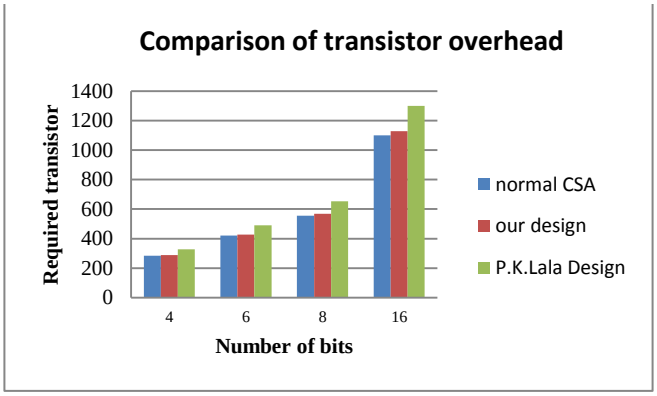


Figure 14: Comparison with CSeA: our proposed design and self-checking CSeA[15]

B. **Reliability Comparison**

The problem associated with TMR reliability is that if two single full adders associated with different modules become faulty one after the other, then the TMR will fail to provide reliable output. This is due to the absence of self-checking and self-replacement in TMR, because the purpose of the voter circuit is to provide reliable output based on majority agreement without indicating the presence of a particular fault in the modules. In order to make TMR self-checking, we need to increase hardware overhead as mentioned by Majumdar et al. [10]. However, the absence of a recovery process will further increase the probability of more than one fault. Thus, TMR is sensitive to a single adder of

each module. To reduce this sensitivity, we propose the concept of dedicated replacement for individual adders, as shown in Fig. 15.

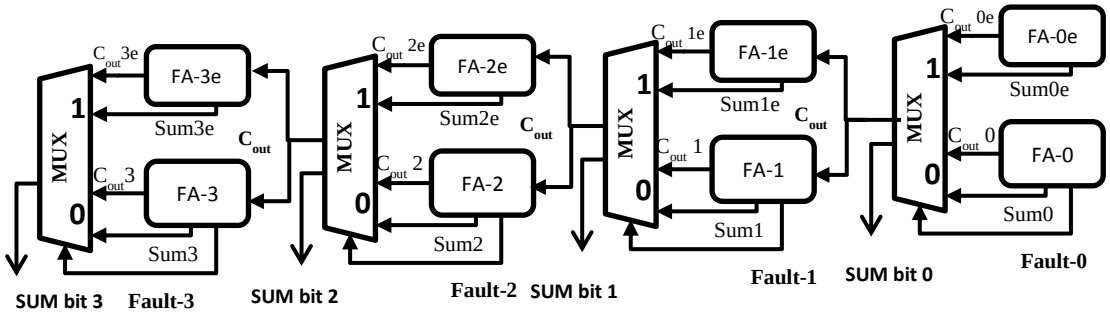


Figure 15: Our proposed 4-bit self-repairing adder

In TMR, all three modules are sensitive with respect to the faults in the individual full adders. Therefore, we can use the concept of simple combinational probability without replacement to compute the probability of fault recovery, as shown in Eq. (5.2). In our proposed design, all full adders are sensitive with respect to their individual replacement full adder modules. If we compute the probability of a critical situation in which two adders at the same position become faulty, then our approach can be the same as two modules having n full adders. Each module has full adders with numbers 1 to n sequentially, such that two full adders of the same number cannot be present in a single module. If we introduced r random faults, then the probability of having faults in at least 2 full adders at the same position without replacement can be computed with Eq. (5.1), where N is the total number of full adders in both modules.

$$\frac{\{\binom{2}{2}\binom{N-2}{r-2}\}*n}{\binom{N}{r}} \tag{5.1}$$

In our proposed design, two full adders at the same position becoming faulty is the worst situation. Therefore, Eq. (5.1) represents the failure condition of our

adder. Hence, the probability of fault recovery in our proposed self-repairing adder can be computed by subtracting Eq. (5.1) from 1 as shown in Eq. (5.3).

$$P_{fault\ recovery\ in\ TMR} = \frac{\binom{n}{r}*3}{\binom{N}{r}} \tag{5.2}$$

$$P_{fault\ recovery\ in\ our\ design} = 1 - \left(\frac{\binom{N-2}{r-2}*n}{\binom{N}{r}}\right) \tag{5.3}$$

where n indicates the number of full adders in a single module, and N shows the total number of full adders in all modules. The total number of faults is represented by the term r.

Considering we have three identical modules of 4-bit adders working in parallel, there is an equal probability of a first fault in any single module. If a second fault occurs, there is a 73% probability that the fault will occur in any one of the remaining modules that were not affected by the first fault. Therefore, if two faults occur in a TMR-based 4-bit adder, there is only a 27% possibility of error recovery. Hence, TMR can efficiently handle only a single fault at a time.

Let us consider the same 4-bit adder with dedicated adders for run-time replacement, as shown in Fig. 15. Suppose that the first fault occurs in Adder-0; then there is only a 14% probability that the next fault will occur in Adder-0e, which is the dedicated repair module for Adder-0. Therefore, if two faults occur in our proposed 4-bit adder, there is about a 86% probability of error recovery. The probability of having a second fault in the dedicated adder for the first infected adder will decrease further with the increase in number of bits. Hence, our designed adder is more reliable against multiple faults, compared to TMR.

It can be seen from Table 7 that the reliability of our proposed design increases with an increase in the number of bits. For a 4-bit adder design, our proposed

approach can sustain up to two faults efficiently. But for a 16-bit adder, our design can handle a minimum of 4 faults effectively, with an 80% probability of error recovery. With TMR, if any two modules go faulty, the whole TMR may fail to provide reliable output. It can also be seen from Table 7 that the reliability of TMR has no significant improvement with respect to the number of bits.

Fazeli et al. [11] adapted the TMR mechanism for fault recovery, which is only possible when input bits are NWV. According to their calculations, based on an ARM processor, there is only a 56% probability of NWV. Therefore, the adder is only 56% reliable for a single possible fault. Furthermore, if input bits are NWV, then the possible error recovery for multiple faults is equal to the TMR.

Moreover, the dominance of our proposed solution, compared to the TMR and NWV can be visualized clearly for 16-bit adder design. The reliability will increase further for higher order adders.

Table 7: Comparison of System Failure in NWVA, TMR and Our Proposed Design Approach

	M. Fazeli et al. [11]	TMR	Proposed Design Approach
Fault Recovery	Fault correction requires three conditions to be fulfilled: 1) Both the operands should be NWV. 2) No more than one module should be faulty. 3) None of the MUXs or other extra circuitry goes faulty, because there is no self-checking in their design.	Fault recovery is possible when no more than one module goes faulty at a time. Totally self-reliable voter circuit is required	Multiple-fault recovery is possible, providing both corresponding adders do not go faulty at one time
Output Reliability on Single fault	56%	100%	100%
4-Bit Adder			
Output Reliability on 2nd Faults	**(Input bits are NWV)** 27%	27%	85.82%
Output Reliability on 3rd Faults	5%	5%	42.8%
16-Bit Adder			
Output Reliability on 2nd Faults	**(Input bits are NWV)** 31%	31%	96.7%
Output Reliability on 3rd Faults	9.7%	9.7%	90.3%
Output Reliability on 4th Faults	2.8%	2.8%	80%
Output Reliability on 5th Faults	0.7%	0.7%	67%

VI. CONCLUSION

With our proposed design of a self-checking full adder, a fault generated in any full adder can be detected individually. Even the error propagated through carry will not create a fault indication in subsequent full adders because all full adders are self-checking with respect to their individual functionality. We then proposed a self-repairing adder by using the fault localization property present in our proposed self-checking full adder. The designed self-repairing adder can provide reliable output for more than one fault, with reduced area overhead. This is because, instead of replacing the whole system of adders, we replace the particular faulty full adder only. Moreover, our proposed 16-bit self-repairing adder can repair up to 4 faults with 80% probability of error recovery.

The self-checking full adder proposed in this thesis is 25% more area efficient than DMR. We applied our self-checking full adder to CSeA to observe the performance and area overhead compared with CSeA without self-checking. It was observed that our designed self-checking carry-select adder has negligible area overhead compared to a normal CSeA without self-checking. Also, our designed self-checking CSeA consumes 15% less area overhead compared to the self-checking CSeA approach without fault localization presented by Vasudevan et al. [15].

BIBLIOGRAPHY

[1] N. Mehdizadeh, M. Shokrolah-Shirazi, and S. G. Miremadi, "Analyzing fault effects in the 32-bit OpenRISC 1200 microprocessor", *Third International Conference on Availability, Reliability and Security*, pp. 648-52, 2008.

[2] A. Meixner, M. E. Bauer, and D. J. Sorin, "Argus: Low-cost, comprehensive error detection in simple cores", *40th Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 210-22, 2007.

[3] M. Nicolaidis, "On-line testing for VLSI: state of the art and trends." *Integration, the VLSI Journal*, vol. 26, no. 1, pp. 197-209, 1998.

[4] H. Psaier and S. Dustdar, "A survey on self-healing systems: approaches and systems", *Computing*, vol. 91, no. 1, pp. 43-73, 2011.

[5] A. Pellegrini, R. Smolinski, L. Chen, X. Fu, S. K. S. Hari, J. Jiang, S. V. Adve, T. Austin, and V. Bertacco, "CrashTest'ing SWAT: Accurate, gate-level evaluation of symptom-based resiliency solutions", *Proceedings of the Conference on Design, Automation and Test in Europe*, pp. 1106-9, 2012.

[6] S. Hong, and S. Kim, "Lizard: Energy-efficient hard fault detection, diagnosis and isolation in the ALU", *IEEE International Conference on Computer Design (ICCD)*, pp. 342-9, 2010.

[7] J. E. Smith and P. Lam , "A theory of totally self-checking system design", *IEEE Trans. Comput.*, vol. C-32, pp.831 -44, 1983.

[8] N. K. Jha and S.-J. Wang, "Design and synthesis of self-checking VLSI circuits", *IEEE Trans. Comput.-Aided Des. Integr. Cicuits Syst.*, vol. 12, no. 6, pp.878 -87, 1993.

[9] T. Angskun, G. Fagg, G. Bosilca, J. Pjesivac-Grbovic, and J. Dongarra. "Self-healing network for scalable fault-tolerant runtime environments", *Future Generation Computer Systems*, vol. 26, no. 3, pp. 479-85, 2010.

[10] A. Majumdar, S. Nayyar, and J. S. Sengar. "Fault Tolerant ALU System", *2012 International Conference on Computing Sciences (ICCS)*, pp. 255-60, 2012.

[11] M. Fazeli, A. Namazi, S.G. Miremadi, and A. Haghdoost, "Operand Width Aware Hardware Reuse: A low cost fault-tolerant approach to ALU design in embedded processors", *Microelectronics Reliability*, vol. 51, no. 12, pp.2374-87, 2011.

[12] T. Koal, D. Scheit, M. Schölzel, H.T. Vierhaus, "On the Feasibility of Built-in Self Repair for Logic Circuits," *IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems*, pp. 316-24, 2011.

[13] C. Khedhiri, M. Karmani, B. Hamdi. "Concurrent Error Detection Adder Based on Two Paths Output Computation", *2011 Ninth IEEE International Symposium on Parallel and Distributed Processing with Applications Workshops (ISPAW)*, pp. 27-32, 2011.

[14] S. Mitra and E. J. McCluskey. "*Which concurrent error detection scheme to choose?*", Proc. International Test Conference, pp. 985-94. IEEE, 2000.

[15] D. P. Vasudevan , P. K. Lala and 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, 2007.

[16] H. Belgacem, K. Chiraz, and T. Rached. "Pass Transistor Based Self-Checking Full Adder", *International Journal of Computer Theory and Engineering*, vol. 3, no.5, 2011.

[17] RV Kshirsagar, RM Patrikar, “Design of a novel fault-tolerant voter circuit for TMR implementation to improve reliability in digital circuits”, *Microelectronics Reliability*, vol. 49, no. 12, pp.1573-77, 2009.

[18] T. Koal, M. Ulbricht and H.T. Vierhaus, “Vitual TMR Scheme Combining Fault Tolerance and Self Repair”, *in 16th Euromicro Conference on Digital System Design(DSD 2013)*, IEEE, pp. 235-42, 2013.

[19] T. Ban and L. Naviner, "A simple fault-tolerant digital voter circuit in tmr nanoarchitectures," in *8th IEEE International Northeast Workshop on Circuits and Systems Conference (NEWCAS 2010)*, pp. 269-72, 2010.

[20] A.G. Ganek and T.A. Corbi, “The dawning of the autonomic computing era”, *IBM Systems Journal* , vol. 42, no. 1, pp. 5-18, 2003.

[21] T. Koal, D. Schiet, H. T. Vierhaus, “A Concept for Logic Self Repair”, *12th Euromicro Conference on Digital System Design, Architectures, Methods and Tools*, pp. 621-624, Aug. 2009.

[22] V. Ocheretnij, D. Marienfeld, E.S. Sogomonyan, M. Gossel, "Self-checking code-disjoint carry-select adder with low area overhead by use of add1-circuits", *10th IEEE International On-Line Testing Symposium*, pp. 31-36, July 2004.

ACKNOWLEDGEMENT

I would first of all thank Allah Almighty Who enabled me to carry out this project with full devotion and consistency. Surely, 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 sincerest gratitude to my supervisor Prof. Jeong-A Lee for her excellent guidance, caring and patience and providing me with comfortable research environment. Her sincere encouragement to pursue my master's degree at CS (Computer System) Lab and continuous support has been a great assistance to achieve a milestone in my career. She is the friendliest, easy going and encouraging advisor that anyone could wish for.

I would also like to show appreciation to all members of the thesis examining committee Prof. Seokjoo Shin and Prof. Kwang-Suk Choi for their valuable advices and insight throughout my research. In addition, I would like to thank Department of Computer Engineering, Chosun University, to provide me the atmosphere to augment my knowledge.

I also pay 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.

Finally I would like to thanks to my whole family for their kind love and prayers which always play a vital role in my success. I dedicate this thesis to my beloved parents Firdous Parveen and Ghulam Akbar Khan and also, to my elder sister Fouzia Akbar, as a reward to their efforts for building my career.