Received 2 December 2021; revised 27 February 2022; accepted 14 March 2022. Date of publication 23 March 2022; date of current version 4 April 2022.

Digital Object Identifier 10.1109/OJCAS.2022.3161873

Self-Repairing Carry-Lookahead Adder With Hot-Standby Topology Using Fault-Localization and Partial Reconfiguration

MUHAMMAD ALI AKBAR, BO WANG (Member, IEEE), AND AMINE BERMAK (Fellow, IEEE)

Division of Information and Computing Technology, College of Science and Engineering, Hamad Bin Khalifa University, Doha, Qatar

This article was recommended by Associate Editor J. Viraraghavan.

CORRESPONDING AUTHOR: M. A. AKBAR (e-mail: maakbar@mail.hbku.edu.qa)

This work was supported by the National Priorities Research Program (NPRP) from the Qatar National Research Fund (a member of Qatar Foundation) under Grant NPRP13S-0212-200345.

ABSTRACT In this paper, a self-checking and -repairing carry-lookahead adder (CLA) is proposed with distributed fault detection ability. The presented design with self-checking and fault localization ability requires an area overhead of 69.6% as compared to the conventional CLA. It can handle multiple faults simultaneously without affecting the delay of conventional CLA, with the condition that each module has a single fault at a time. The repairing operation utilizes the hot-standby approach with partial reconfiguration in which the faulty module would be replaced by an accurately functioning module at run-time. The proposed self-repairing adder with high fault coverage requires 161.5% area overhead as compared to conventional CLA design which is 35.3% less as compared to the state-of-the-art partial self-repairing CLA. Moreover, the delay of the proposed 64-bit self-repairing CLA is 40.7% more efficient as compared to conventional ripple carry adder.

INDEX TERMS Self-repairing, fault localization, self-checking adder.

I. INTRODUCTION

THE REALIZATION of digital circuit in the deep submicron process comes at the risk of losing reliability, due to complexity, thermal-cycling, and decreased power supply [1]–[3]. This is in addition to the effect of different environmental conditions that further increase the vulnerability of digital systems [4]. Therefore, the need for self-checking hardware design has become inevitable. To facilitate fault detection, the concept of totally self-checking was introduced [5], based on which different approaches have been reported in the literature. These approaches mainly rely on hardware- or time-based redundancy. In the case of hardware redundancy, more than one concurrently working hardware produces either the same, complemented or encoded output. These outputs will be compared to identify potential faults. For time redundancy, single hardware is used to produce correlated outputs in different intervals; the comparison of these delayed outputs will be used to identify potential faults. In both cases, multiple copies of output are used to determine the fault. It has been argued that the self-checking design should also involve fault localization to minimize the cost of fault repairing [6]. The presence of fault detection without recovery cannot fulfill the demand for reliable execution in current digital systems. Thus, built-in self-repair is becoming particularly pertinent to current semiconductor technology [7]. The adoption of this approach will ensure that all the system's key components are reliable and error-free. One of such key components in digital systems is the adder, which often appears in critical signal paths [8]. Therefore, the introduction of adders with built-in self-repair capability can significantly improve the reliability of digital systems.

Among the fastest adders used in digital systems is the Carry-Lookahead Adder (CLA). For CLA, the summation circuitry for each bit can "lookahead" for their respective incoming carry bit. It means that each full adder in the cascade can run independently without waiting for the carry out of the preceding adder. The speed is therefore significantly improved, at the expense of hardware overhead. Therefore, traditional self-checking approaches like double modular or triple modular redundancy are not feasible for CLA due to their area overhead. The most common approach for designing self-checking CLA is the parity prediction scheme that can detect faults in either even or odd number of bits.

In [9], a parity prediction approach was adopted along with a two-rail code for self-checking CLA. The two-rail code tests the carry block, whereas the parity prediction checks the summation outputs. A similar approach of using parity prediction with duplicated summation block was also reported in [10]. Besides their drawbacks like area overhead, fault coverage, etc., these approaches can only detect faults without recovery or self-repairing.

This is why, a partial triple modular redundancy with parity prediction approach was adopted for self-repairing CLA [11] and [12]. An improved design was further reported in [13], that constitutes two different architectures of the voter circuit. This design approach provides self-repairing for the carry generation block using triple modular redundancy, while a parity prediction-based self-checking approach was employed for the summation bits. To reduce the area overhead of triple modular redundancy, a shared logic was utilized to generate multiple copies of the carry bit. As stated, the advantage of CLA is that each of its carry bit is generated independently to reduce the computation time. However, in [13], each carry block needs to produce copies of the following two carry bits; for example, the block responsible for producing the i^{th} carry bit (C_i) will also produce redundant of C_{i+1} and C_{i+2}. Therefore, each carry bit has multiple copies, and the final carry is generated based on the majority decision using a voter. It should be noted that the voter produces dual carry bits; the first one is for the addition, whereas the second is for computing the parity of the intermediate carry bits. The adoption of the parity prediction approach further limits the fault coverage of their design to an even or odd number of faulty bits. That means its reliability and fault detection was only ensured partially.

In this paper, we propose a self-checking and -repairing CLA with distributed fault detection ability. The proposed design can detect and locate multiple faults simultaneously, with the condition that each module should have only one fault at a time. The fault recovery is achieved with a hot-standby approach in which a spare module replaces the faulty one. The replacement process is conducted with a novel partial reconfiguration concept in which the modified input values update the functionality of the circuits generating the internal carry bits. The proposed design with self-checking requires 69.6% area overhead compared to the conventional CLA without self-checking whereas, the required overhead is increased to 161.5% for proposed self-repairing CLA with a single spare module. Notably, the proposed design can achieve 99.1% reliability for a 64-bit CLA with 16-blocks where each block has a pair of spare modules and can perform 4-bit computation. Compared to ripple carry adder, the delay efficiency of the proposed self-checking CLA remains the same as the conventional CLA design, which is 50.9%. While the delay efficiency of 64-bit self-repairing CLA is 40.7% higher than ripple carry adder.

The remainder of this paper is organized as follows. Section II describes the proposed self-checking and -repairing CLA design. Evaluation and comparison with previous approaches is presented in Section III, followed by a conclusion in Section IV.

II. PROPOSED SELF-REPAIRING CARRY LOOK-AHEAD ADDER DESIGN

In this section, the operation of CLA is introduced followed by the proposed self-checking and -repairing designs.

A. CLA TOPOLOGY AND OPERATION

In CLA, all the internal carry bits are pre-computed in parallel to facilitate its operation [12]. Typically, a CLA consists of two main blocks. The first block is the carry block (CBL), which generates the internal carry bits using carry generator (CG) modules. The second block is the summation block (SBL) which is responsible for generating the sum-bits using the sum generator (SG) modules, as shown in Fig. 1(a).

The CBL is designed using the basic concept of carry propagation and generation. The carry bit will be generated if both inputs are high (i.e., $G_i = a_i \cdot b_i$), whereas the carry will be propagated if either one or both input bits are high (i.e., $P_i = a_i \oplus b_i$ or $P_i = a_i + b_i$). By combining these two operations, the i^{th} carry bit can be computed by (1). As inherent to the CLA, each carry bit should be generated in parallel using independent circuitry. Therefore, (1) is modified such that each carry bit only depends on the initial carry-in C_{in}. For example, the first three carry bits of a 4-bit CLA can be generated independently using (2) to (4). The generalized Boolean expression for the i^{th} carry bit is shown in (5). Note that in conventional non-self-checking CLA, the generate and propagate bits are computed once for all carry bits and then shared between them. This logic sharing is further extended to compute the sum-bits, which are equal to $P_i \oplus C_{i-1}$ [13]–[14].

$$C_i = G_i + P_iC_{i-1} \tag{1}$$

$$C_0 = G_0 + P_0C_{in} \tag{2}$$

$$C_1 = G_1 + P_1C_0 = G_1 + P_1G_0 + P_1P_0C_{in} \tag{3}$$

$$C_2 = G_2 + P_2C_1 = G_2 + P_2G_1 + P_2P_1G_0 + P_2P_1P_0C_{in} \tag{4}$$

$$C_i = G_i + P_iG_{i-1} + \cdots + P_0 \ldots P_{i-1}P_iC_{in}. \tag{5}$$

Since each carry-bit is generated using an independent circuitry, the CBL is the most area-hungry and complex part of a CLA. Its area overhead and complexity becomes extremely high as the size of the adder increases. To address this issue, the block architecture of CLA is widely adopted in which multiple small-size CLA blocks are repeated to construct an adder for large input bit-width [13]. As a result, the number of carry bits generated by each CBL is equal to the block

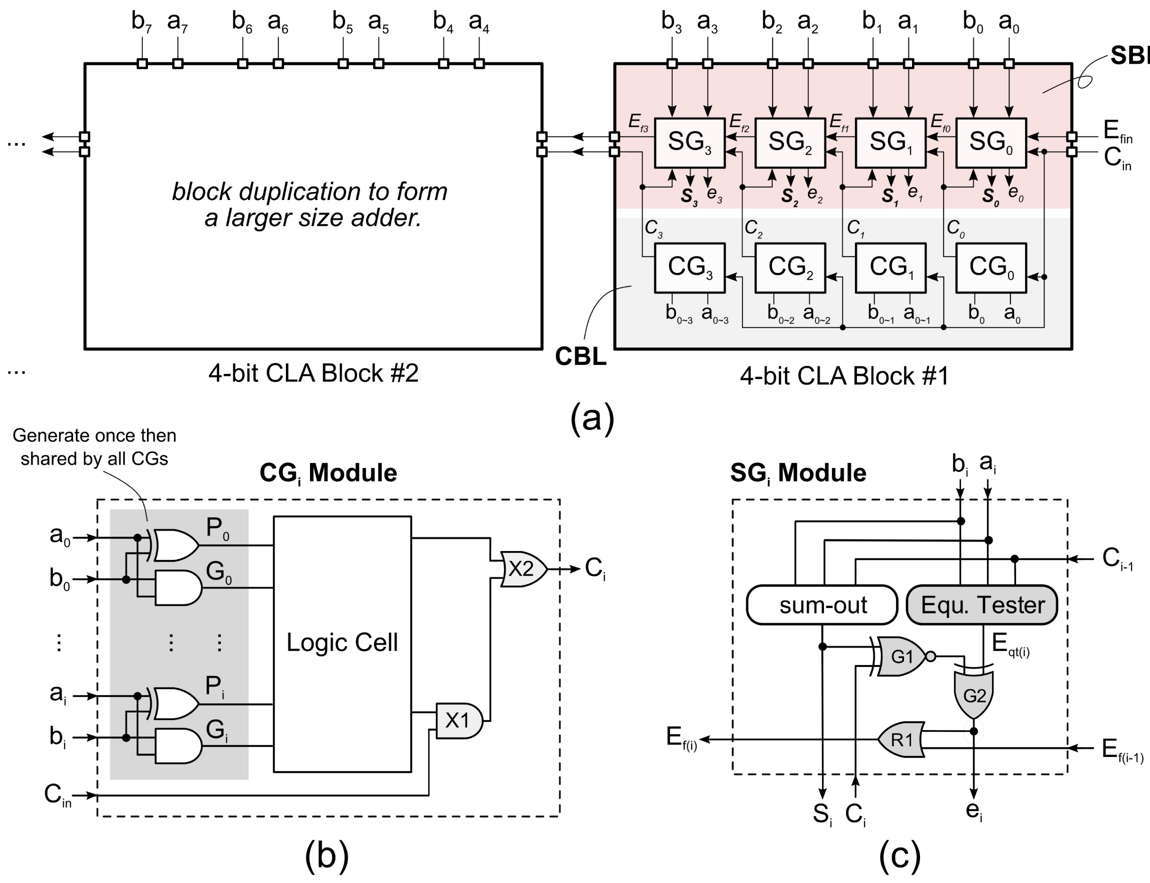


FIGURE 1. **The proposed architecture of (a) Self-checking CLA, (b) along with the module of the carry generator (CG), and (c) sum generator (SG).**

size, as shown in Fig. 1(a). The final carry-out bit C_{out} generated by each CBL will be used as C_{in} for the next CBL. The overall latency will increase because the next block needs to wait for the preceding block's result, but the area overhead and complexity will be significantly reduced. To reduce computational delay, the carry block should be designed such that C_{in} is the last element needed for computation. As soon as C_{in} is received from the previous block, the output could be updated immediately, as seen in Fig. 1(b). The logic cell implementations in CG vary from CG_0 to CG_3 depending on their respective Boolean equations. Meanwhile, except for the first CBL, each consecutive CBL will generate the carry-bits with an additional delay of two logic gates, i.e., X1 and X2 shown in Fig. 1(b).

B. PROPOSED SELF-CHECKING CLA WITH FAULT LOCALIZATION

The area overhead of CBL for generating carry-bits in parallel is so high that any approach using hardware duplication for self-checking will not be fruitful. Furthermore, the presence of logic sharing in non-self-checking CLA is not encouraged in self-checking architectures because any fault in shared logic will quickly get masked and cannot be detected or localized.

To address this issue, we propose a hardware-friendly self-checking and fault localization approach for CLA, in which the i^{th} sum-bit (S_i) and carry-out bit (C_i) respectively generated by the SBL and CBL, are compared with the i^{th} input bits a_i and b_i to determine any potential fault. Its operation can be summarized as: S_i of the SBL and C_i of the CBL will be equal to each other, if and only if the previous carry-bit C_{i-1} of the CBL and the i^{th} input bits are all equal, that is:

$$If (a_i == b_i == C_{i-1})\ then\ S_i = C_i\ otherwise\ S_i \neq C_i.$$

With the above conditional decision, an equality tester is required to check whether a_i, b_i and C_{i-1} are equal and produce a comparison output $E_{qt(i)}$, followed by a checker to determine whether a fault happens. For an error-free adder, if $E_{qt(i)} = 1$, S_i and C_i must be equal; otherwise they must be complementary. The $E_{qt(i)}$ bit can be computed using (6), and the checker can be implemented by (7). The architecture of the proposed SG module is shown in Fig. 1 (c). It should be noted that the sum-out produces the sum-bit while the gates G1 and G2 are responsible for checking the relation of S_i, C_i and $E_{qt(i)}$. An error indicator e_i for each SG will then be produced based on the output of G2. The OR gate (R1) generates the universal error signal E_f, whose value will be high for all SGs proceeding to the faulty one. The distinguishing features of e_i and $E_{f(i)}$ can be visualized in the self-repairing process.

$$E_{qt(i)} = \overline{(a_i \oplus b_i) + (a_i \oplus C_{in})} \qquad (6)$$

$$e_i = S_i \odot C_i \oplus E_{qt(i)}. \qquad (7)$$

As each output bit is tested separately in the proposed design, the detected fault will easily be localized due to the distributed checking mechanism. Moreover, it is possible to have a shared propagate and generate signal in CBL because each respective sum-bit and carry-bit is self-checking with respect to their internal functionality. However, the propagate signal cannot be shared with the SG because there is a possibility that the faulty propagate signal will affect the sum-bit and carry-bit together. It should be noted that the technique to compare the sum-bit and carryout-bit has been presented in [15] for ripple carry adder, whereas it has not been examined for CLA. The basic architecture of self-checking multiplexer (MUX) design was introduced in [16], which has been improved with reduced transistor count in [17]. To limit the area-overhead of our proposed design, the self-checking MUX and OR gate are implemented using the design presented in [17].

C. PROPOSED SELF-REPAIRING CLA WITH PARTIAL RECONFIGURATION

Typically, in self-repairing designs based on hot-standby approach, a faulty module is replaced by a functioning one. The implementation of this approach is easy for adders whose internal carry bits are generated by inter-connected circuitry, and therefore, the input bits of a faulty module can be simply shifted to the spare one. However, the complexity of the shifting process is high for adders using independent circuits to generate internal carry bits. This is because the circuit linked to the carry of the respective faulty module should be replaced with the next available CG in such a way that each CG of CBL should be aware of the shift operation. This awareness cannot be achieved without modifying the circuitry because each carry-bit has a unique equation. For example, the logic circuit to generate C_2 requires the signal G_0, G_1, G_2, P_0, P_1 and P_2 as in (4). Suppose C_1 gets faulty, then the values of G_1, G_2, P_1, and P_2 should be modified so that the circuitry for generating C_2 becomes equivalent to that of C_1. A simple shift operation is insufficient as it can only modify G_2 and P_2. Therefore, a partial reconfiguration is required with the shift operation so that the hot-standby approach becomes applicable for adder having independent carry circuits, such as the CLA.

It has been observed that during the shifting process, if the operands of the faulty module of CBL is set to 0 and 1, then the propagate and generate signal for that position will be set to 1 and 0, respectively. By doing this, the logic cell of each CG module proceeding to the faulty one will be modified because the portion of their circuitry handling the propagates and generate signal of the faulty module will become null and void with 1 and 0 values. Consider again the previously stated example of a faulty C_1, the signals G_2 and P_2 have already been modified due to the shifted input values, whereas the signals G_1 and P_1, which depend on the input values of the faulty module will be set to 0 and 1, respectively. Hence the operation of (4) will become equivalent to (3) and the shift operation required for the hot-standby approach will be achieved. This process is called "partial reconfiguration" because the logical operation of CG modules has been modified by deactivating parts of the circuit.

A 4-bit self-repairing CLA using the proposed approach is shown in Fig. 2. As stated, e_i represents the individual error of the SG/CG pair, it is therefore used to update the input bits of the faulty module to 1,0 and also to divert the input carry of the faulty module to the next SG. Since the logic cell of each CG has already been modified, the positions of all other proceeding carry-bits will remain unchanged. Whereas E_f represents the universal error, whose value is a function of all individual e_i. E_f will be high for all SGs after the faulty one, whereas its value remains low for all the SGs prior to the faulty one. Therefore, it is used to control the shift operation of the input and output bits. CG_X and SG_X in Fig. 2 are the spare modules that are used during the recovery process.

III. PERFORMANCE COMPARISON

The proposed approach has been compared in terms of area, fault coverage, and latency with the recently reported approaches for self-checking and -repairing CLA.

A. AREA OVERHEAD

The area overhead mainly depends on the implementation approaches. In conventional CLA designs, the implementation is carried out with shared logic. However, shared logic is not encouraged in self-checking approaches because the fault in the shared logic will get masked. Therefore, each block in our proposed approach is implemented with an independent logic circuit, which inevitably increases the area overhead. However, it can be optimized at the transistor level by using approaches like pass transistors.

In this paper, the area overhead is compared in terms of gate counts. Since each NAND gate requires 4 transistors, for a fair comparison, the NAND equivalent circuit of each module based on transistor count is computed and compared with similar approaches used in [12] and [13]. For example, an XOR gate and a self-checking MUX can each be implemented using six transistors [17]. Therefore, each pair of XOR gates and self-checking MUXs have been considered equivalent to three NAND gates. Similarly, two NOT gates are equivalent to one NAND gate and a pair of self-checking OR gate is equal to three NAND gates. The required gate count for CBL is computed by converting the equation of each carry bit to its NAND equivalent form as presented in (8). Since the number of spares in our proposed self-repairing CLA can vary depending on the application, the gate count is therefore computed with and without spare

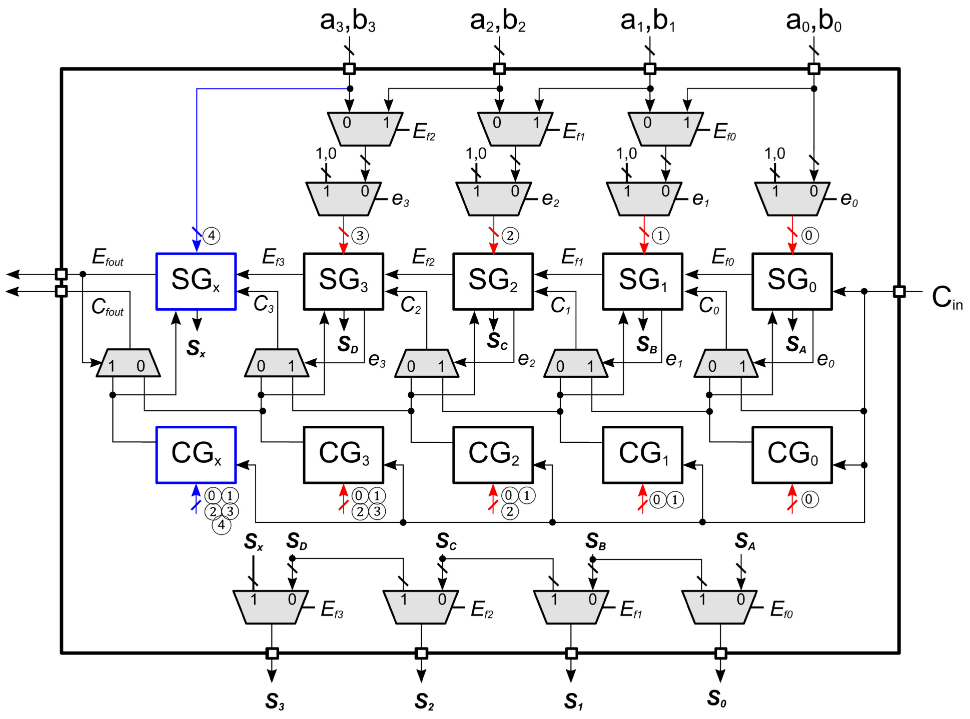


FIGURE 2. **The proposed architecture of 4-bit self-repairing CLA.**

TABLE 1. **NAND Gate required for SBL and CBL of a 4-bit self-checking and -repairing CLA.**

Block	Self-checking CLA	Self-repairing CLA without Spare	Self-repairing CLA with Spare
SBL	36	76	90
CBL	42	42	65

TABLE 2. **Comparison results of the area overhead in terms of NAND gate count.**

Adder Size (bit)	CLA	[12]	[13]	Proposed self-checking CLA	Proposed self-repairing CLA
16	184	785	745	312	509
32	368	1569	1489	624	981
64	736	3137	2977	1248	1925

modules. In both cases, the required number of gates for MUX is included in SBL, as shown in Table 1.

$$C_i = \overline{\overline{G_i} \cdot \overline{P_i C_{i-1}}}. \tag{8}$$

The overall area overhead for different designs with different input bit-width (adder size) is shown in Table 2. Our proposed approach with multiple error detection and localization feature requires only 69.6% area overhead compared to the conventional CLA without self-checking. While due to the use of MUX as a shifter, the resultant area overhead of our proposed single spare based self-repairing CLA increased to 161.5%, which is 35.3% less than the previously reported partial self-repairing CLA [13]. Note that for a single spare case, the area overhead of the spare module will only be counted once for each size of an adder.

B. LATENCY

The main advantage of CLA is its speed. Ideally, the delay of CLA only depends on the last CG and SG module as shown in (9). This superiority will disappear if the speed of CLA degrades to a similar level as that of a ripple carry adder. The generation of redundant carry bits in [13] caused their architecture to be two times slower than the conventional CLA because the two redundant carry bits with shared logic and dependent circuitry require the same time as that of two internal carry bits generated by a ripple carry adder. The additional delay is caused by the voter circuitry because the internal carry will not be delivered to SG until the voter decides the final carry out. In contrast, the latency for our self-checking CLA is the same as that of a conventional CLA because the checking mechanism does not interrupt the circuit operation whereas, the latency will increase for self-repairing CLA because of the shift operation. The critical path for the worst case when a fault is detected in the first SG/CG pair is shown in Fig. 3.

The MUX needs to update the input values because of the detected fault; in the meantime, error signal E_f will also be forwarded to the spare module. However, SG_X needs to wait for the carry bit C_3 to produce the sum-bit (S_x). The final sum-bit is obtained after an additional delay of a MUX that is responsible for shifting the output bits, while CG_3 and CG_X will generate the carry out in parallel. Therefore, the block delay is increased to five MUXs, one SG and (n-2)

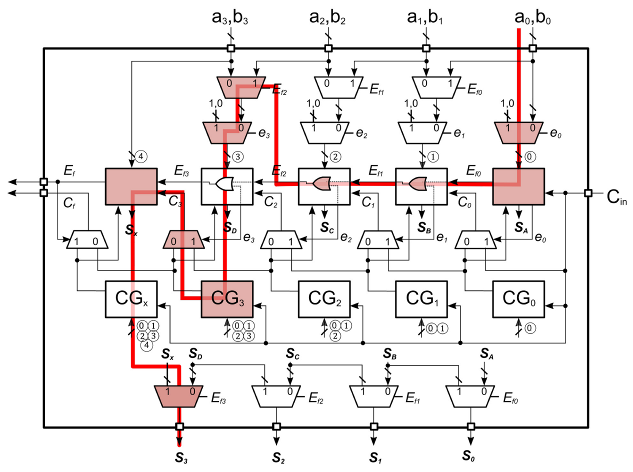


FIGURE 3. The critical path for the Self-Repairing CLA.

OR gate delay, as expressed in (10); where n is the size of adder for a single spare case. In the case of having a spare module for each block, n will be equal to the size of the block because the universal error signal (E_f) of each block will remain localized. For instance, with 4-bit block size, the delay overhead is 5-MUX + 2-OR + 1-XOR gate, which will remain constant irrespective of the adder's size due to the parallel operation of all the blocks of CLA.

$$T_{CLA} = T_{SCH_{CLA}} = 1SG + 1CG_{last} \tag{9}$$

$$T_{SR_{CLA}} = T_{CLA} + 5MUX + (n-2)OR + 1SG. \tag{10}$$

C. FAULT COVERAGE

For the state-of-the-art design [13], fault recovery is limited to CBL, while the fault in SBL is indicated by a parity prediction approach. Our proposed approach however, can detect multiple faults at a time with the condition that each MUX, OR and the CG/SG pair should not have more than one fault at a time. Due to the distributed self-checking mechanism, the proposed approach will not only detect the faults but can also provide fault localization. It can even be adopted in CLA circuits built by using standard look-ahead carry generator chips because it only requires an equivalence tester and a checker circuitry, which can be connected externally. Moreover, it can detect the occurrences of both temporary and permanent faults.

The self-repairing part is not limited to the CBL but also covers the faulty SG/CG module pair by replacing it with the spare one. However, the proposed hot-standby approach is able to recover the faults which can be detected by the checker present in SG module. While the checker can only detect the fault if any one of the S_i, $E_{qt(i)}$ and C_i bits is faulty, whereas if any two of them gets faulty then the fault may or may not be detected. In order to further evaluate the self-repairing characteristics of our proposed architecture, we consider four cases such that in each of them the fault targets a particular module of the design. It should be noted that in digital systems some faults get internally masked and could not affect the final output due to the inherent self-repairing ability of the circuit. These faults are not considered in this research for evaluating the fault correction ability. Moreover, each concerned component can produce single bit output, therefore the fault occurrence indicates that the output bit is inverted of its actual value. With this concept the resultant output of two sequentially connected faulty modules is always a non-faulty one because the first faulty module produces inverted output which is either masked or produces erroneous output from the next sequentially connected module. If the second module is also faulty and the output is affected by the previous fault, then it will produce inverted bit of the erroneous output as a result the output bit of the second module is corrected automatically.

Case-1 Fault in SG: There are three sub-modules in SG which are responsible for producing S_i, $E_{qt(i)}$ and e_i bits. The fault stuck to any of them will be detected and recovered effectively by the proposed approach.

Case-2 Fault in CG: The output of the CG is fed to the respective SG for checking. Any fault in CG will be detected by the checker of the SG and therefore the recovery is possible for this case.

Case-3 Fault in Carry-Bypass-MUXs: The carry-bypass-MUXs (CBM) are responsible to divert the input carry of the faulty module to the next SG module, as shown in Fig. 2. If the fault stuck to any CBM then it will produce erroneous C_{i-1} value for the connected SG module. However, the faulty C_{i-1} can only effect the S_i and $E_{qt(i)}$ bits. Whereas, the carry-bit (C_i) which needs to be compared with the S_i bit remain valid because each carry-bit in CLA is produced with independent circuitry. The truth table of SG module in the presence and absence of the faulty C_{i-1} bit is shown in Table 3, where half of the faulty cases cause by the CBM is detected effectively by the SG. Hence, the fault correction will be reduced to 50% because the SG module cannot detect all the faults occurred in S_i and $E_{qt(i)}$ bits, simultaneously.

The main reason of having undetected faults is because both S_i and $E_{qt(i)}$ bits are produced by using single C_{i-1} bit received. This limitation can be solved, if both S_i and $E_{qt(i)}$ bits use different C_{i-1} bit which is possible because each self-checking MUX produce dual output that is the inverted and non-inverted output using independent circuitry. With this slight modification, the C_{i-1} bit can only be used for computing $E_{qt(i)}$ bit, whereas the S_i bit will be computed using the inverted value of C_{i-1}, i.e., $\overline{C_{i-1}}$. The equation for the sum-out bit will also be modified as shown in (11), but the fault recovery will become feasible for all possible single event upset occur in CBM because a fault in C_{i-1} bit can only effect $E_{qt(i)}$, whereas the S_i bit will remain valid and vice versa.

$$Sum = A \oplus B \odot \overline{C_{i-1}}. \tag{11}$$

Case-4 Fault in Other Components: The input/output shifter MUXs along with the OR gate in SG module are self-checking which means that the fault can be detected but without recovery. Fault recovery is not considered for

TABLE 3. Functionality of SG module with and without faulty C_{i-1} bit.

SG without faulty C_{i-1}						SG with faulty C_{i-1}						Fault diagnosis by SG
C_{i-1} (Actual)	A	B	C_i	S_i	$E_{qt(i)}$	C_{i-1} (After fault)	A	B	C_i	S_i	$E_{qt(i)}$	
0	0	0	0	0	1	1	0	0	0	1	0	Not detected
0	0	1	0	1	0	1	0	1	0	0	0	Detected
0	1	0	0	1	0	1	1	0	0	0	0	Detected
0	1	1	1	0	0	1	1	1	1	1	1	Not detected
1	0	0	0	1	0	0	0	0	0	0	1	Not detected
1	0	1	1	0	0	0	0	1	1	1	0	Detected
1	1	0	1	0	0	0	1	0	1	1	0	Detected
1	1	1	1	1	1	0	1	1	1	0	0	Not detected

these components due to the area constraints. It is possible to replace the self-checking MUX with the self-repairing one presented in [18]. Although the design in [18] is not completely self-repairing but it can effectively handle multiple fault conditions. Another possible way is to perform re-computation for recovering the temporary faults which in turn will increase the time penalty of the system.

Apart from the above mentioned cases, the proposed repairing approach is limited by the number of spare modules, which can be increased based on the application so that each block has its own available spare module. The block-wise distributed-spare modules ensure that if any block fails to perform self-repairing, the rest of the blocks will remain self-repairing. For example, we can divide a k-bit adder into m-blocks, where each block performs 4-bit addition with a single spare module. It means five SG and five CG modules are present in each block. Assuming x random faults are introduced in the adder, the probability of having y faults (y is in the range of 0 to 5) in the same block can be calculated by (12), where P_{bf} is the probability of block failure. Note that for a single spare design, the block with $y > 1$ cannot be repaired. Therefore, the probability of block recovery (P_{br}) can be computed by (13). The probability of complete failure (P_{cf}) when each block has $y > 1$ is given by (14). It should be noted that the self-checking property of the adder will always remain valid whether the self-repairing is functioning or not.

$$P_{bf} = \frac{\binom{5}{y} \times \binom{5(m-1)}{x-y}}{\binom{5m}{x}} \tag{12}$$

$$P_{br} = 1 - \frac{10 \times \binom{5(m-1)}{x-2}}{\binom{5m}{x}} \tag{13}$$

$$P_{cf} = P_{bf} \times m \tag{14}$$

The probability of fault recovery and complete failure when 2 out of 3 faults occur in the same block is shown in Table 4. The probability of block recovery increases with the size of the adder, and for a 64-bit adder, the probability of fault recovery reaches 99.1%. The improved reliability with the size of an adder is because of the increased number of blocks; thus more spare modules can be utilized.

TABLE 4. Probability of fault recovery for x = 3 and y = 2.

Adder Size (bit)	**8**	**16**	**32**	**64**
No. of blocks (m)	2	4	8	6
$P_{bf}\%$	41.7	13.1	3.5	0.9
$P_{br}\%$	58.3	86.8	96.4	99.1
$P_{cf}\%$	83.3	52.6	28.3	14.6

D. VERIFICATION

The proposed approaches and the conventional design of ripple carry adder and CLA are implemented using the standard 180nm process. The simulation results for the area, power, and delay are obtained using 1.8V supply. The delay of the proposed approaches is compared with the ripple carry adder and CLA. The reason of selecting ripple carry adder is because the delay of self-repairing CLA is higher than CLA, therefore it is essential to check the efficiency as compared to conventional ripple carry adder.

Using (9), the delay of the proposed self-checking 64-bit CLA is estimated to be equal to the delay of the conventional CLA, which is 50.9% less than that of RCA. For the proposed 64-bit self-repairing CLA, the delay overhead is increased to 20.9% as compared to conventional CLA. Although, the delay efficiency is still 40.7% better as compared to conventional ripple carry adder design, as shown in Table 5. In terms of area, the proposed 64-bit self-checking CLA requires 54.6% overhead when compared to conventional CLA, while the area overhead of a similar-sized self-repairing CLA is estimated to be 160.5%. In terms of power consumption, the proposed 64-bit self-checking architecture requires an overhead of 40.9% compared to the conventional CLA; while the same-sized self-repairing architecture requires an overhead of 154.5% over the conventional CLA. The simulation results validates the manually estimated area overhead and performance efficiency of the proposed self-checking and -repairing CLA design.

IV. CONCLUSION

In contrast to the traditional approach of using parity prediction for designing self-checking CLA, a novel

TABLE 5. Simulation results of area, power, and delay for conventional ripple carry adder, CLA, proposed self-checking and -repairing CLA.

Adder Size	Ripple carry adder			CLA			Proposed Self-Checking CLA			Proposed Self-Repairing CLA		
(bit)	Area (mm^2)	Power (nW)	Delay (ns)	Area (mm^2)	Power (nW)	Delay (ns)	Area (mm^2)	Power (nW)	Delay (ns)	Area (mm^2)	Power (nW)	Delay (ns)
16	0.0159	8.5547	3.8713	0.0206	11.9508	2.1577	0.0319	16.9283	2.1577	0.0564	30.6868	2.6124
32	0.0319	17.9125	6.1658	0.0412	24.0082	3.2142	0.0638	33.8867	3.2142	0.1092	61.2591	3.86745
64	0.0636	36.3925	10.7565	0.0825	48.1229	5.2718	0.1275	67.8035	5.2718	0.2149	122.518	6.37745

approach of detecting and localizing faults in CLA architecture is presented. The designed approach used the concept of self-checking and fault localization full adder in which the fault is detected by comparing the input and output bits. The proposed 64-bit self-checking CLA requires 69.6% more area than conventional CLA, whereas it can detect and localize multiple faults at a time with the condition that a single module should not have more than one fault at a time.

A hot-standby approach along with a novel partial reconfigurable approach is adopted for self-repairing CLA. The proposed self-repairing CLA approach with single spare requires 161.5% more area than CLA which is 35.3% less as compared to the previously reported approach. The reliability and fault coverage are also higher than the previously reported approach. A 64-bit CLA with 16 blocks can recover 3 consecutive faults with 99.1% probability with the condition that each block has a single spare module. Moreover, the self-checking property of the circuit remains valid whether the recovery is possible or not. The power consumption of a 64-bit self-checking CLA is 40.9% more than conventional CLA design, whereas the overhead is further increased to 154.5% for the respective size of self-repairing CLA.

The time latency of conventional CLA will not be affected with the proposed self-checking approach because the checker is not affecting the actual computation process. Whereas, the latency of the proposed self-repairing approach is increased to the summation of one SG, five MUXs and (n-2) OR gate delays as compared to conventional CLA. The simulation results demonstrate that the delay efficiency of a 64-bit self-repairing CLA design is 40.7% higher than that of ripple carry adder.

REFERENCES

[1] F. Tang, A. Bermak, and Z. Gu, “Low power dynamic logic circuit design using a pseudo dynamic buffer,” *Integration*, vol. 45, no. 4, pp. 395–404, 2012.

[2] N. Mehdizadeh, M. Shokrolah-Shirazi, and S. G. Miremadi, “Analyzing fault effects in the 32-bit OpenRISC 1200 microprocessor,” in *Proc. 3rd Int. Conf. Avail. Rel. Security*, 2008, pp. 648–652.

[3] A. Meixner, M. E. Bauer, and D. J. Sorin, “Argus: Low-cost, comprehensive error detection in simple cores,” in *Proc. 40th Annu. IEEE/ACM Int. Symp. Microarchit.*, 2007, pp. 210–222.

[4] H. G. Kang and T. Sung, “An analysis of safety-critical digital systems for risk-informed design,” *Rel. Eng. Syst. Safety*, vol. 78, no. 3, pp. 307–314, 2002.

[5] J. E. Smith and P. Lam, “A theory of totally self-checking system design,” *IEEE Trans. Comput.*, vol. C-32, no. 9, pp. 831–844, Sep. 1983.

[6] A. G. Ganek and T. A. Corbi, “The dawning of the autonomic computing era,” *IBM Syst. J.*, vol. 42, no. 1, pp. 5–18, 2003.

[7] T. Koal, D. Schiet, and H. T. Vierhaus, “A concept for logic self repair,” in *Proc. 12th Euromicro Conf. Digit. Syst. Design Archit. Methods Tools*, Aug. 2009, pp. 621–624.

[8] M. P. Kumar and M. Kiran, “Design of optimal fast adder,” in *Proc. IEEE Int. Conf. Adv. Comput. Commun. Syst.*, 2013, pp. 1–4.

[9] M. Nicolaidis, “Carry checking/parity prediction adders and ALUs,” *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 11, no. 1, pp. 121–128, Feb. 2003.

[10] E. S. Sogomonyan, V. Ocheretnij, and M. Gossel, “A new code-disjoint sum-bit duplicated carry look-ahead adder for parity codes,” in *Proc. 10th Asian Test Symp.*, Kyoto, Japan, 2001, pp. 365–370.

[11] M. Valinataj, “A novel self-checking carry-lookahead adder with multiple error detection/correction,” *Microprocess. Microsyst.*, vol. 38, no. 8, pp. 1072–1081, 2014.

[12] M. Valinataj, “Fault-tolerant carry look-ahead adder architectures robust to multiple simultaneous errors,” *Microelectron. Rel.*, vol. 55, no. 12, pp. 2847–2857, 2015.

[13] M. Valinataj, “Enhanced multiple-error resilient carry look-ahead adders through new customized fault-tolerant voters,” *Microelectron. Rel.*, vol. 96, pp. 7–20, May 2019.

[14] P. Balasubramanian, C. Dang, D. L. Maskell, and K. Prasad, “Approximate ripple carry and carry-lookahead adders a comparative analysis,” in *Proc. 30th IEEE Int. Conf. Microelectron. (MIEL)*, 2017, pp. 299–304.

[15] M. A. Akbar and J.-A. Lee, “Self-repairing adder using fault localization,” *Microelectron. Rel.*, vol. 54, nos. 6–7, pp. 1443–1451, 2014.

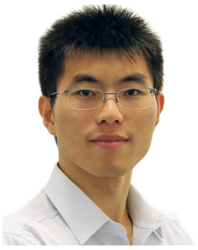
[16] D. P. Vasudevan, P. K. Lala, and J. P. Parkerson, “Self-checking carry-select adder design based on two-rail encoding,” *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 54, no. 12, pp. 2696–2705, Dec. 2007.

[17] M. A. Akbar, B. Wang, and A. Bermak, “Self-repairing hybrid adder with hot-standby topology using fault-localization,” *IEEE Access*, vol. 8, pp. 150051–150058, 2020.

[18] A. K. Neelam and S. Musala, “Real-time self repairable multiplexer for fault tolerant systems,” in *Proc. IEEE Int. Conf. Commun. Signal Process. (ICCSP)*, 2020, pp. 1124–1127.

MUHAMMAD ALI AKBAR received the B.Eng. degree (Hons.) in electronic engineering from NED University, Karachi, Pakistan, in 2011, and the M.Sc. degree in computer engineering from Chosun University, South Korea, in 2014. He is currently pursuing the Ph.D. degree in computer engineering with Hamad Bin Khalifa University, Qatar.

In 2014, he joined Qatar University as a Research Assistant to work on different research project related to hardware design and machine learning applications. His main research focus was on fault localization, self-reliable systems, sensor design, and artificial intelligence.



BO WANG (Member, IEEE) received the B.Eng. degree (Hons.) in electrical engineering from Zhejiang University, Hangzhou, China, in 2010, and the M.Phil. and Ph.D. degrees in electronic and computer engineering from The Hong Kong University of Science and Technology (HKUST), Hong Kong, in 2012 and 2015, respectively. In 2015, he joined HKUST as a Postdoctoral Researcher and led the HKUST-MIT Consortium Project on wireless sensing node design for smart green building applications. Afterwards, he was with the Massachusetts Institute of Technology in 2016, on low power data converter design for this project. In 2017, he joined Hamad Bin Khalifa University, Qatar Foundation, as a Founding Faculty, where he is currently an Assistant Professor with the Division of Information and Computing Technology, College of Science and Engineering. His research interests include energy-efficient analog mixed-signal circuits, sensor and sensor interface, and heterogeneous integrated systems for in vitro/vivo health monitoring. He was a recipient of the IEEE ASP-DAC Best Design Award in 2016. He serves as a Technical Committee Member of the IEEE CAS Committee on sensory systems.

AMINE BERMAK (Fellow, IEEE) received the master's and Ph.D. degrees in microelectronics and microsystems from Paul Sabatier University, Toulouse, France, in 1994 and 1998, respectively.

He joined the ECE Department, Hong Kong University of Science and Technology (HKUST), where he held all academic ranks and subsequently promoted to a Full Professor. He was also an ECE Associate Head for Research and Postgraduate studies. He is currently with Hamad Bin Khalifa University, Qatar Foundation, Qatar, holding a Full Professor Appointment as well as an Associate Dean. He taught 25 different courses at the undergraduate and postgraduate levels. He is recognized as the world-leading author in the sensors area as well as the inventor of time-domain sensing. He has graduated 25 Ph.D.'s and 20 master students. He has published over 350 articles in journals, book chapters and conference proceedings and designed over 30 chips.

Prof. Bermak has received six distinguished awards, including the 2004 IEEE Chester Sall Award from IEEE Consumer Electronics Society, the IEEE Service Award from IEEE Computer Society, the Best Paper Award at the 2005 International Workshop on System-On-Chip for Real-Time Applications, the Best Student Paper Award at IEEE International Symposium on Circuits and systems ISCAS 2010, and the Best University Design Contest Award at ASP Design Automation Conference, Macau, in 2016. For his excellence and outstanding contribution to teaching, he was nominated for the 2013 Hong Kong UGC Best Teacher Award (for all HK Universities). He is a recipient of the 2011 University Michael G. Gale Medal for distinguished teaching (the Highest University-wide Teaching Award). He is also a two-time recipient of the Engineering School Teaching Excellence Award in HKUST for 2004 and 2009, respectively. He has served on the editorial board of IEEE TRANSACTIONS ON VERY LARGE SCALE INTEGRATION (VLSI) SYSTEMS and the *Sensors* Journal. He is also currently serving on the editorial board of IEEE TRANSACTIONS ON BIOMEDICAL CIRCUITS AND SYSTEMS and the IEEE TRANSACTIONS ON ELECTRON DEVICES. He is also an editor for Nature *Scientific Reports*. He is an IEEE Distinguished Lecturer.