Biomedical and Pharmacological Literature Archives

Vol 5, Issue 2, 2025

SELF-REPAIRING CARRY-LOOKAHEAD ADDER WITH HOT-STANDBY TOPOLOGY

Mr. B. KALYANCHARAVARTHY[1], NAKKA VEERA VENKATA SATYANARAYANA SAGAR[2], PINNINTI JYOTHIKA[3], BONDA VEERA VENKATA SAYTANARAYANA MURTHY[4]

[1]Assistant Professor, Dept. of ECE, PRAGATI ENGINEERING COLLEGE

[234]UG Students, Dept. of ECE, PRAGATI ENGINEERING COLLEGE

ABSTRACT

In this project, 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. 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

INTRODUCTION

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 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.

LITERATURE SURVEY

1."An area-delay efficient multi-operand binary tree adder using modified carry select adder" by M. Singh, M. Sharma, and A. K. Verma (2016):

This paper proposes an area-delay efficient MOBTA that uses a modified carry select adder (MCSA) as the building block. The proposed adder is shown to have a smaller area and delay than other existing MOBTAs while still maintaining a similar power consumption.

2. "Low power and high-speed multi-operand binary tree adder" by A. Mittal, M. Gupta, and R. S. Anand (2017):

This paper proposes a low-power and high-speed MOBTA that reduces power consumption by optimizing the carry propagation path and reducing the number of logic gates required to implement the adder. The proposed adder is shown to have a lower power consumption and a faster speed than other existing MOBTAs.

3. "Low-Power Multi-Operand Binary Tree Adder Design Based on Signed-Digit Number System" by C. Li, Y. Li, and J. Li (2019):

This paper proposes a low-power MOBTA design based on the signed-digit number system (SDNS). The proposed design reduces power consumption by exploiting the redundancy in the SDNS representation and using a carry-save adder (CSA) as the building block.

4. "Design of low-power multi-operand binary tree adder using hybrid binary adder cells" by S. Patra, S. Pal, and D. K. Mandal (2020):

This paper proposes a low-power MOBTA design using hybrid binary adder cells (HBACs). The proposed design reduces power consumption by optimizing the carry propagation path and reducing the number of logic gates required to implement the ADDER

PROPOSED SYSTEM

PROPOSED SELF-REPAIRING CARRY LOOK-AHEAD ADDER DESIGN

A. CLA Topology And Operation

In CLA, all the internal carry bits are pre-computed in parallel to facilitate its operation. 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 below Fig.

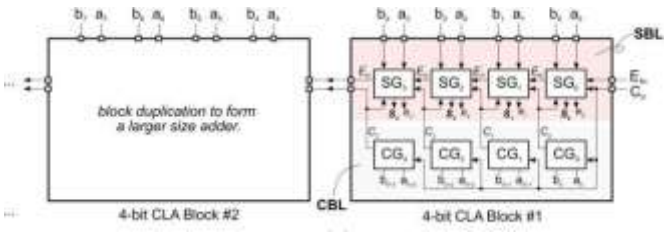


Figure.2 CLA Block Diagram

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., $Gi = ai \cdot bi$), whereas the carry will be propagated if either one or both input bits are high (i.e., $Pi = ai \oplus bi$ or $Pi = ai + bi$). By combining these two operations, the *ith* carry bit can be computed. As inherent to the CLA, each carry bit should be generated in parallel using independent circuitry.

$$C_i = G_i + P_i C_{i-1} \quad (1)$$

$$C_0 = G_0 + P_0 C_{in} \quad (2)$$

$$C_1 = G_1 + P_1 C_0 = G_1 + P_1 G_0 + P_1 P_0 C_{in} \quad (3)$$

$$C_2 = G_2 + P_2 C_1 = G_2 + P_2 G_1 + P_2 P_1 G_0 + P_2 P_1 P_0 C_{in} \quad (4)$$

$$C_i = G_i + P_i G_{i-1} + \cdots + P_0 \ldots P_{i-1} P_i C_{in} \quad (5)$$

This logic sharing is further extended to compute the sum-bits, which are equal to $Pi \oplus Ci$-1. 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. As a result, the number of carry bits generated by each CBL is equal to the block size, as shown in Fig. 1(a). The final carry-out bit *Cout* generated by each CBL will be used as *Cin* for the next CBL. The

To reduce computational delay, the carry block should be designed such that *Cin* is the last element needed for computation. As soon as *Cin* is received from the previous block, the output could be updated immediately. 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

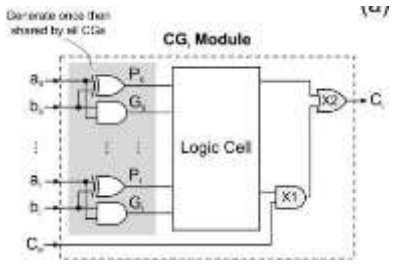


Figure.2 CG module block diagram

B.Proposed Self-Checking CLA With Fault Localization

To address this issue, we propose a hardware-friendly self-checking and fault localization approach for CLA, in which the *ith* sum-bit (*Si*) and carry-out bit (*Ci*) respectively generated by the SBL and CBL, are compared with the *ith* input bits *ai* and *bi* to determine any potential fault. Its operation can be summarized as: *Si* of the SBL and *Ci* of the CBL will be equal to each other, if and only if the previous carry-bit *Ci*-1 of the CBL and the *ith* input bits are all equal, that is:

If (ai == bi == Ci-1) then Si = Ci otherwise Si != Ci.

With the above conditional decision, an equality tester is required to check whether *ai, bi* and *Ci*- 1 are equal and produce a comparison output *Eqt(i)*, followed by a checker to determine whether a fault happens.For an error-free adder, if *E qt(i)* = 1, *Si* and *Ci* must be equal; otherwise, they must be complementary. The *Eqt(i)* bit can be computed using (6), and the checker can be implemented.

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

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

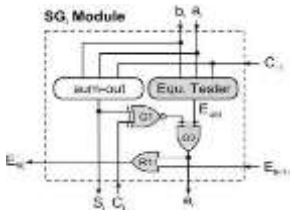


Figure.3 SG module block diagram

Proposed Self-Repairing Cla With Partial Reconfiguration

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.

A 4-bit self-repairing CLA using the proposed approach is shown in Fig. As stated, *ei* 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 *Ef* represents the universal error, whose value is a function of all individual *ei*. *Ef* 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. *CGX* and *SGX* in the spare modules that are used during the recovery process.

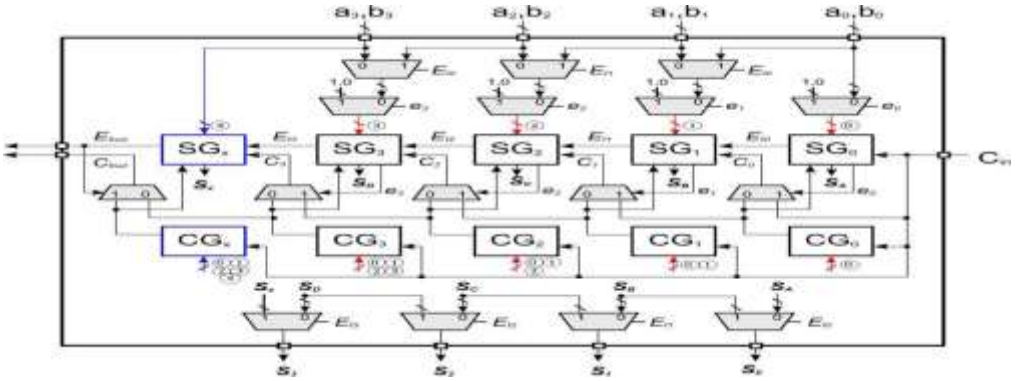
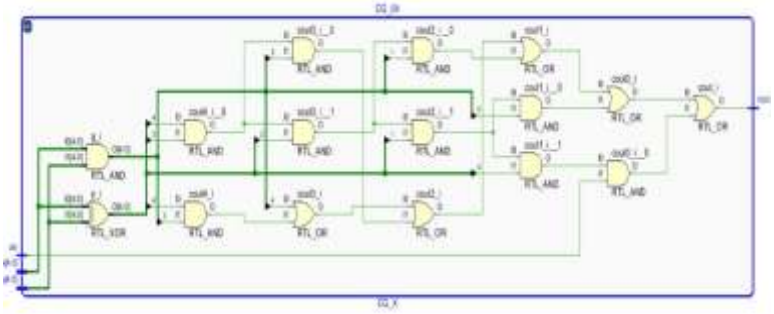


Figure.4 Block Diagram of Self-repairing carry look adder Schematic CG-X BLOCK

STIMULATION & SYNTHESIS RESULTS



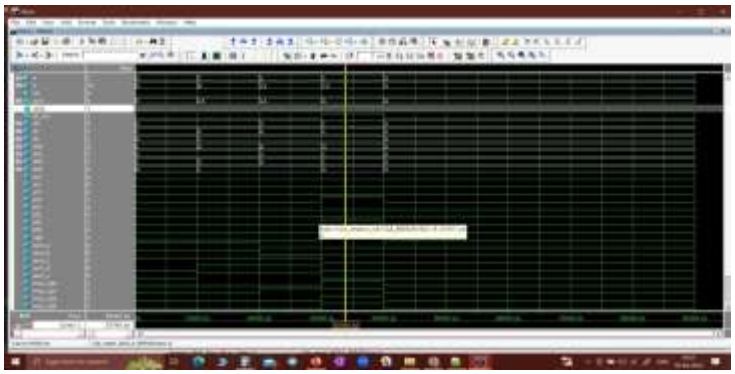


Figure.5 Simulation Wave Result

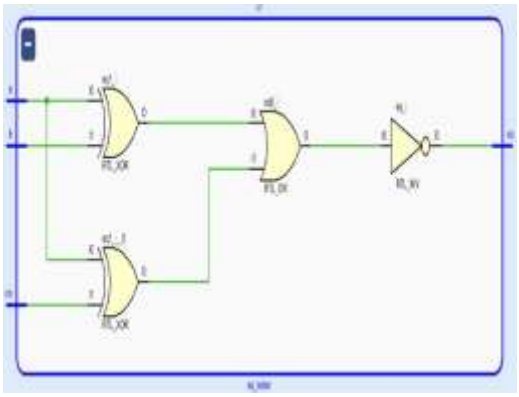


figure.6 Schematic SG-Eq-Tester BLOCK

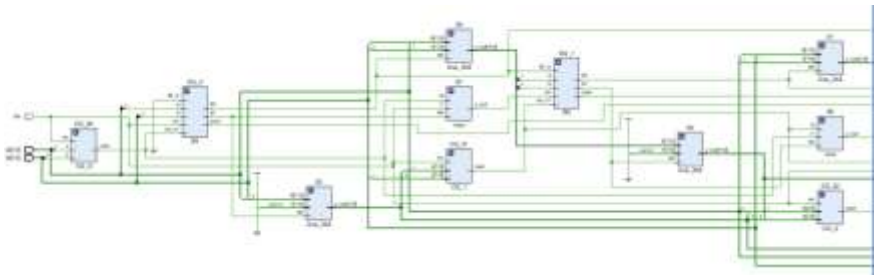


Figure.7 Schematic self-repairing CLA

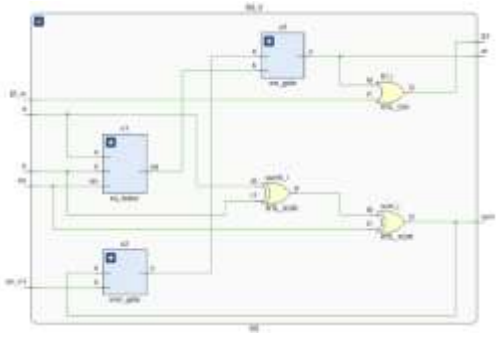


Figure.8 Schematic SG BLOCK

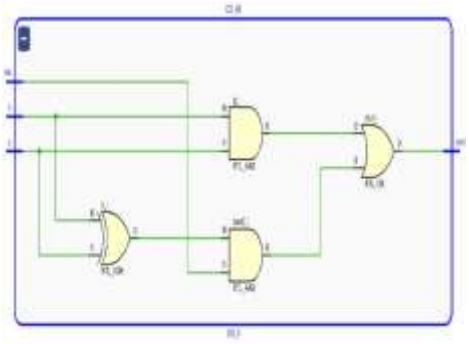


Figure.9 Schematic CG-0 BLOCK

Area Report

Site Type	Used	Fixed	Available	Util%
Slice LUTs*	5	0	303600	<0.01
LUT as Logic	5	0	303600	<0.01
LUT as Memory	0	0	130800	0.00
Slice Registers	0	0	607200	0.00
Register as Flip Flop	0	0	607200	0.00
Register as Latch	0	0	607200	0.00
F7 Muxes	0	0	151800	0.00
F8 Muxes	0	0	75900	0.00

Ref Name	Used	Functional Category
IBUF	9	IO
OBUF	6	IO
LUT5	4	LUT
LUT6	1	LUT
LUT4	1	LUT
LUT3	1	LUT

ADVANTAGES

Improved reliability: By using redundancy and fault tolerance techniques, the adder can operate with a higher level of reliability, even in the presence of faults.

Reduced downtime: With the use of hot-standby topology, the adder can switch seamlessly between the active and standby circuits, reducing downtime and improving overall system availability.

Improved fault tolerance: The adder is designed to detect and isolate faults, allowing for targeted replacement of faulty components without affecting the operation of the rest of the circuit.

Increased accuracy: By ensuring that the adder operates correctly and reliably, the system can produce accurate results, which is critical in applications that depend on precise .

APPLICATIONS

Medical devices: The adder can be used in medical devices such as MRI machines and other imaging equipment, where accuracy and reliability are essential.

Industrial automation: The adder can be used in industrial automation systems, such as robotic systems and process control systems, where high levels of accuracy and reliability are required.

Financial applications: The adder can be used in financial applications, such as stock trading and financial modelling, where accuracy and reliability are crucial.

Communications: The adder can be used in communication systems, such as wireless networks and satellite communications, where reliable and accurate computations are required for signal processing and data transmission.

CONCLUSION

The designed approach uses 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 8-bit self-checking CLA requires 20.6% 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.

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.

FUTURE SCOPE

Hardware security: The self-repairing approach can be extended to include hardware security features, such as tamper detection and response. This could be particularly useful for applications that require high levels of security, such as military and aerospace systems.

Internet of Things (IoT): The self-repairing approach could be applied to IoT devices, which often have limited resources and require high reliability. By using partial reconfiguration, the adder circuitry could be dynamically adjusted to optimize performance and reduce power consumption.

Machine learning: The self-repairing approach could be used to develop more robust and reliable machine learning algorithms. By incorporating fault localization and partial reconfiguration techniques, machine learning systems could continue to operate even in the presence of faults.

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, 2003.00