International Journal of Advance Research and Innovation
Vol. 6(2), Apr-Jun 2018, pp. 31-34
Doi: 10.51976/ijari.621806
www.gla.ac.in/journals/ijari
© 2018 IJARI, GLA University

Article Info
Received: 25 Jan 2018 | Revised Submission: 20 Feb 2018 | Accepted: 28 Feb 2018 | Available Online: 15 Jun 2018

Implementation of Parity Checker Using CMOS Logic Techniques

Bhargava Yasasvi, Charu Rana** and Pankaj Rakheja****

ABSTRACT

The technology is growing rapidly where the sizes of the components are getting reduced as the size gets decreased the, possibility of errors gets increased. These errors can't be prevented as they are generated in the running phase. To handle such problems, we need a circuit which will be monitoring continuously and correcting the errors generated. This paper proposes different ways to implement a parity checker in the previous self-checking register. When compared with previous techniques. The circuits are stimulated in spice using 90nm CMOS technology.

Keywords: *Parity-Checker; Transmission Gates; Pass Transistors; Self-Checker; Glitch Filter.*

1.0 Introduction

Errors are generated in every circuit by which the performance can get affected. These errors can be soft errors or hard errors. Soft errors are caused due to the interaction or colloidal of particle and a change in state can be observed. Soft errors can be classified into Intrinsic-Temperature, Noisy and Extrinsic Errors-Cosmic rays, neutrons.

Considering these factors, the performance of circuit gets reduced. Thus, to maximize the performance, a self-checking register is used to control the errors generated and to correct them. Previously there were so many techniques to control the errors, but some techniques are not so efficient as the main circuit errors get solved but the redundant circuit has some errors. So, to control them a SETTOFF self-checking register was used for radiation errors.

The self-checking register in the previous techniques has parity checker, glitch filter followed by td checker. In this paper discuss various ways to implement the parity checker.

The parity checker is basically an XOR tree which can be implemented by using CMOS, pass transistor and transmission gates. The main purpose of developing a new circuit is to increase the performance in minimum time and to decrease the area as much as possible.

This paper compares the difference between different implementation techniques and shows that the area over head and delay are reduced.

2.0 Previous Techniques

The previous technique implemented parity checker in a conventional way of using CMOS in which there are 8 MOSFETS used to design an XOR gate and the output is passed to the input of the Glitch filter which will reduce the glitches and further connecting it with the TD checker which generate the final output[1].

The disadvantage of such parity checker is the usage of large area overheads which is a major drawback of the circuit design. This will also increase the power consumption.

2.1 Parity checker

Parity checker is the process that transmits the data between nodes during communication. Original bits are used to create an even and odd bit number. These bits are transmitted through a link that can be any medium.

Data is considered to be accurate when the transmitted and received bits are equal. Parity checker is implemented using the Xor tree.

**Corresponding Author: Department of Electronics and Electrical Engineering, The North Cap University, Gurugram, Haryana, India.*
***Department of Electronics and Electrical Engineering, The North Cap University, Gurugram, Haryana, India.*
****Department of Electronics and Electrical Engineering, The North Cap University, Gurugram, Haryana, India.*

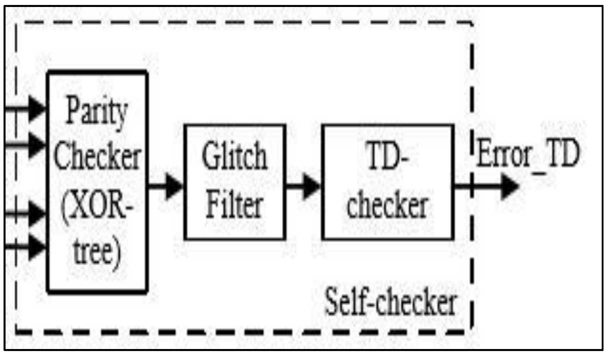
3.0 Implementation of Proposed Parity Checker Using Different Techniques

The Self-checking circuit is comprised of 3 components parity checker, glitch filter and TD-checker. In this paper, the parity checker is implemented through different methods and compar-

ed the output and come to conclusion that which method can be used. Parity checker is implemented by the following ways: -

1 CMOS
2 Pass Transistors
3 Transmission gate

Fig 1: Self-Checker



3.1 CMOS

CMOS using both PMOS and NMOS to implement a logic gate. One MOSFET works at one time and gives us a strong '0' and '1'. In this paper, CMOS is implemented by using 8 MOSFETS and 4 MOSFETS.

3.2. Pass Transistors

Pass transistors are generally NMOS MOSFETS. Transistors are used as switches to pass logic levels between nodes of a circuit, instead of as switches connected directly to supply voltages. This reduces the number of active devices, but has the disadvantage that the difference of the voltage between high and low logic levels decreases at each stage. Each transistor in series is less saturated at its output than at its input.

3.3 Transmission gates

Transmission gate works similar as a relay which can conduct in both the direction or control the signal with voltage potential. It is a CMOS-based switch, in which PMOS passes a strong 1 but poor 0, and NMOS passes strong 0 but poor 1. Both PMOS and NMOS work simultaneously. Self-checker is implemented using CMOS inverters, Transmission gate at 1v. The W/L ratio for PMOS will be 280nm/100nm and for NMOS is 480nm/100nm. The parity checker when implemented by

3.3.1 CMOS- 8 MOSFETS

This is the conventional way in which 4 PMOS and 4 NMOS are connected serially with each other while the inputs are A, B, ABAR and BBAR. These inputs are given to NMOS and PMOS such that we get a XOR gate. Fig-2, the implementation using MOSFETS.

Fig 2: CMOS-8 MOSFETS

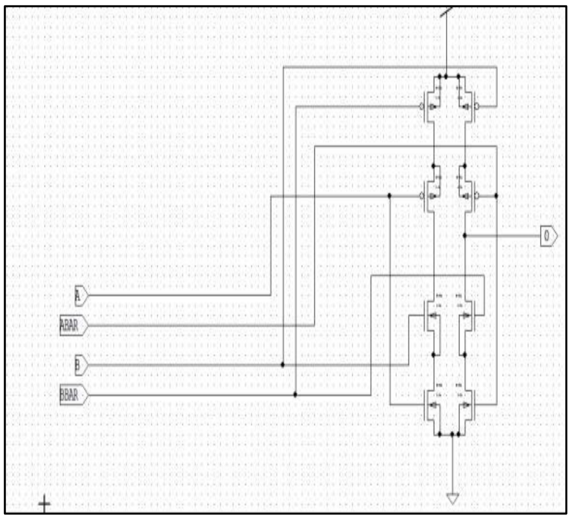


Fig 3: CMOS- 4 MOSFETS

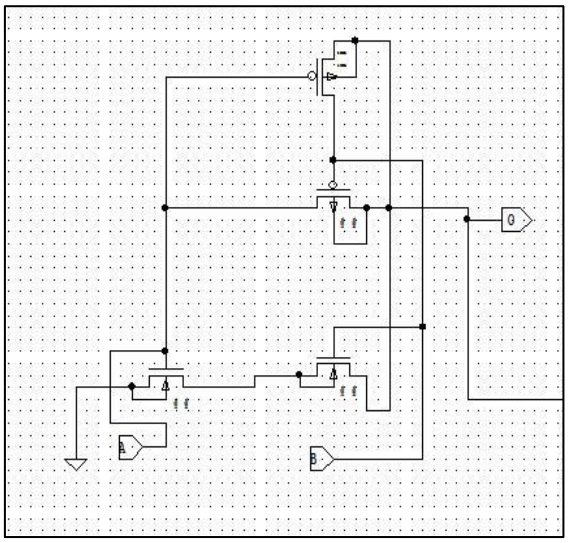


Fig 4: XOR Output

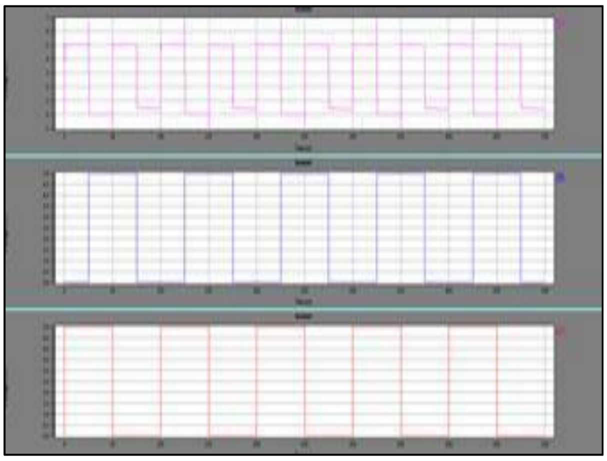


Fig 5: Self-Checker Output

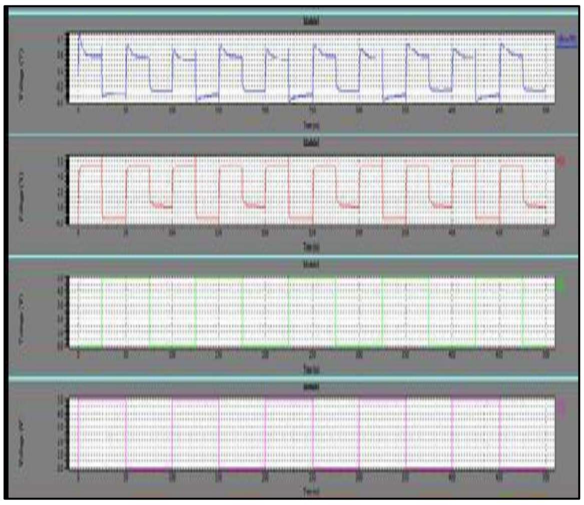


Fig 6: XOR with Transmsision Gate

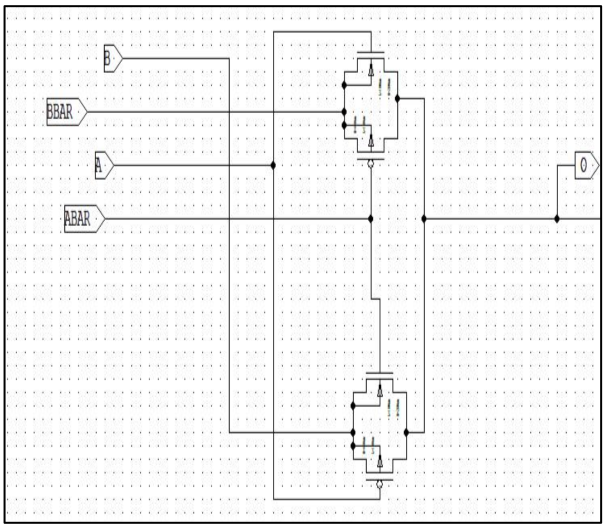
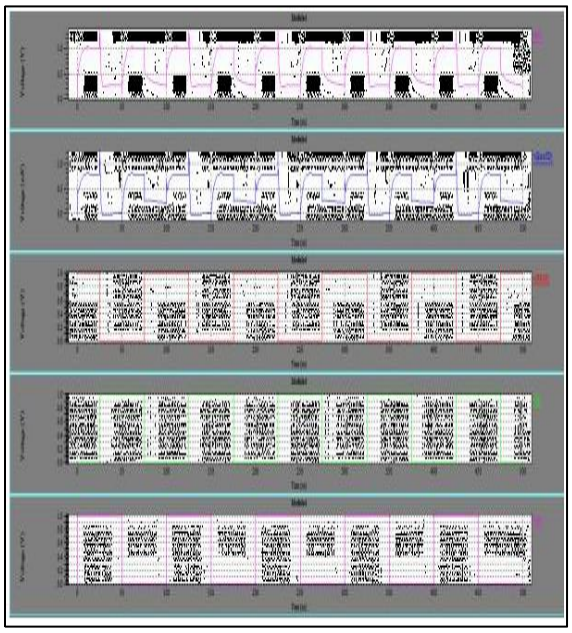


Fig 7: XOR and Self-Checker Output



Another way of implementing self-checker is is using 4 MOSFET, 2 pmos and 2 nmos

3.3.2 . Transmission gate

In this technique Xor gate is implemented by using 2 transmission gates, with inputs A, B, ABAR and BBAR

3.3.3 Pass transistors

This is implemented by using 2 Pass transistors in which the supply is given to the drain and gate of the MOSFETS, with inputs A, B, ABAR and BBAR.

Fig 8: XOR with Pass Transistors

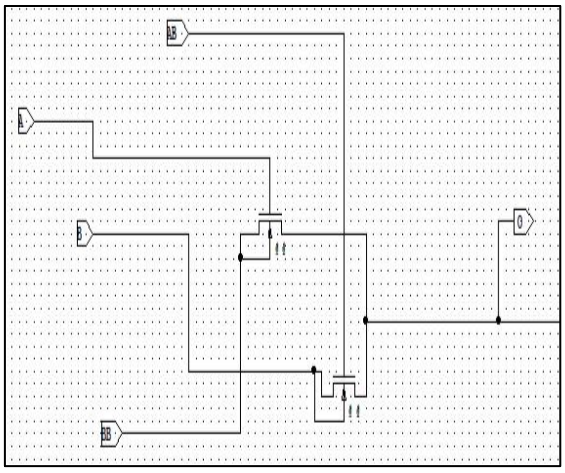
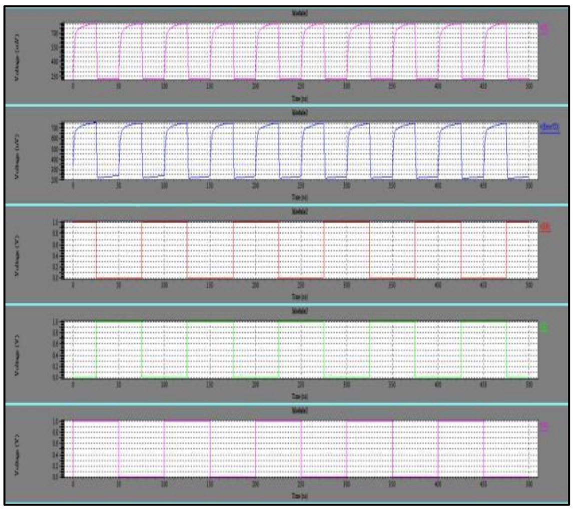


Fig 9: XOR and Self-Checker Output



4.0 Comparative Analysis

The area gets decreased in the all the 3 Techniques as compared to the conventional way. The Technique A, B, C uses MOSFETS 4:4:2. The power consumption also gets decreased in all the methods as per the usage of the MOSFETS. The delays and area of the different methods with respect to conventional CMOS are in the table I.

Tool: - T-spice.

Aspect ratio: - 90nm.

Table I

Device	**No. of Transi stors**	**Area Reduc ed**	**Delay of XOR**	**Delay of Self-**
		(%)		Checker
CMOS-4	4	50	7.2364 e-	1.2377e-
MOSFET			010	009
Transmiss ion	4	50	7.2364 e-	1.2377e-
Gate			010	009
Pass	2	75	1.0306 e-	4.5640e-
Transistor			009	010

5.0 Conclusions

In this paper, different techniques are used for implementing the parity checker that compares the following parameters that are no. of transistors, reduction of an area and the delay produced by XOR gate.

Techniques used in the comparison are transmission gates, pass transistors and CMOS mosfet. Comparing all these techniques one can conclude that using pass transistors is efficient than the other techniques as pass transistor reduces the area by 75% and delay produced is minimum.

References

[1] Y Lin, M Zwolinski. A Cost-Efficient Self-Checking Register Architecture for Radiation Hardened Designs.

[2] MN Babu, PNVK. Hasini, N Pavithra. An Efficient High-Speed 9-bit Parity Checker using 4-2 Compressors (IJARECE), 4(5), 2015.

[3] S Roy, RH Vanlalchaka. Power efficient odd parity & checker Circuits (ICETACS), 1st International Conference 2013.

[4] DA Anderson, G Metze. Design of Totally Self-Checking Check Circuits for m-Out-of-n Codes, IEEE Transactions on Computers, C-22(3), 1973.

[5] J Khakbaz, EJ McCluskey. Self-Testing Embedded Parity Checkers, IEEE Transactions on Computers, C-33(8,) 1984

[6] N Kanopoulos, JH Carabetta. Design and implementation of a totally self-checking 16 × 16 bit array multiplier, 2, 2003.