المؤتمر العلمي الدولي الثالث للعلوم و الهندسة
The 3RD Scientific International Conference in Science & Engineering

http://bwu.edu.ly/icse2024
Received 25/07/2024 Revised 16/08/2024 Published 10/09/2024
icse@bwu.edu.ly *ترسل الورقات كاملة علي

Designing and Implementing a Self-checking Shifter Utilizing Berger Code Methodology.

A. M. Ejamail[a], Alhadi A. Khalleefah[b], A. H. Maamar[c]
[a]Computer engineering & IT, College of Technology Sciences, Bani-Walid -Libya
[b]Department of Computer engineering, College of ELECTRONIC TECHNOLOGY, Bani-Walid-Libya
[c]Department of Computer engineering, College of ELECTRONIC TECHNOLOGY, Bani-Walid-Libya

[a]Adel.m.Ejamail@gmail.com [b]grimida2008@gmail.com [c]Ali_h_maamar@yahoo.uk.com

Abstract: Digital systems today are more complex than ever before. The complexity of digital circuits leads to increased crosstalk, noise, and other potential sources of transient errors during normal operation. Conventional offline testing procedures cannot ensure location of these issues; they may be recognized through online or concurrent error detection (CED) strategies. Concurrent error detection empowers advanced frameworks to approve the exactness of comes about amid standard movement. Berger Code is prevalent approach for concurrent error detection applications as it can discover all unidirectional errors in computerized frameworks. This manuscript introduces a design for self-checking shifter Using Berger code.

Keywords: Berger Code, shift register, Concurrent Error Detection, Two-Rail Checker, Self-checking.

1. Introduction

Concurrent error detection methodologies have been extensively utilized in the realm of commercial digital systems since the 1960s [1]. The primary aim associated with the implementation of concurrent error detection is to conduct real-time evaluations on the system's outputs to ensure data integrity by identifying transient or permanent faults while the system operates under normal conditions. The significance of Concurrent Error Detection spans across a diverse array of applications, encompassing sectors such as space technology, avionics, telephone switching networks, and online transaction processing. A multitude of real-time error detection mechanisms can be integrated into a computing system. These mechanisms fundamentally rely on redundancy in various aspects, including hardware replication, redundant codes, software, and execution time. Redundancy entails the utilization of additional resources that surpass the necessities of the non-verified system [2]. This paper exclusively focuses on information redundancy, specifically employing Berger code checkers within the framework of Totally Self-Checking (TSC).

2. Shift register

A shift register comprises a digital circuit that involves a series of flip flops, in which the output of one flip flop is linked to the input of the subsequent flip flop. The information held in the shift register has the capability to be transferred from one position to the succeeding

position. By interconnecting N-flip flops, an N-bits shift register can be established, with each flip flop retaining a single bit of data. The data contained within the shift register is movable either towards the left or the right.

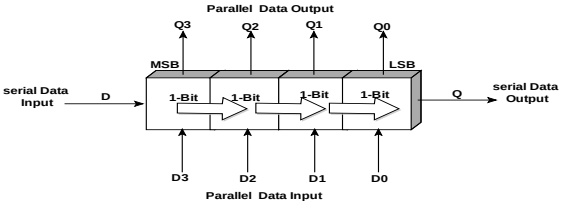

Fig. 1: shift register.

A left logical shift of one position moves each bit to the left by one. The vacant least significant bit (LSB) is filled with zero and the most significant bit (MSB) is discarded as shown in Figure 2.

If X = $(b_n, b_{n-1}, \ldots\ldots\ldots, b_1, b_0)$ is to be shifted to the left, then all the bits of X are shifted to the left one positon, and X becomes,

$$X=(b_{n-1}, b_{n-2}, \ldots\ldots\ldots, b_1, b_0, 0)$$

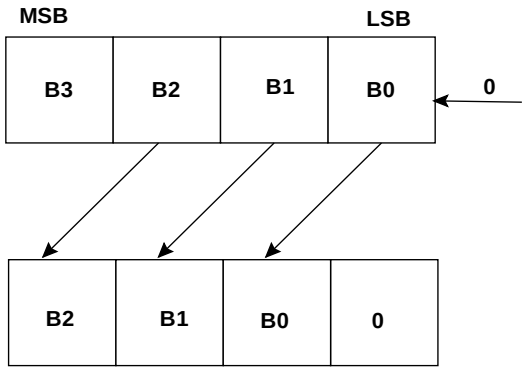

Fig. 2: shift Left register.

A right logical shift of one position moves each bit to the right by one. The least significant bit is discarded and the vacant MSB is filled with zero as shown in Figure 3.

If X = $(b_n, b_{n-1}, \ldots\ldots\ldots, b_1, b_0)$ is to be shifted to the right, then all the bits of X are shifted to the right one place, and X becomes,

$$X=(0, b_n, b_{n-1}, b_{n-2}\ldots\ldots\ldots, b_1)$$

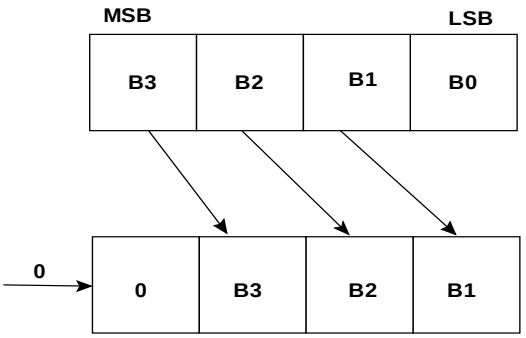

Fig. 3: shift Right register.

3. SELF-CHECKING CIRCUITS

Self- checking can be described as the capacity to automatically confirm the presence of a fault in logic (such as chips, boards, or assembled systems) [3]. Circuits with self-checking capabilities enable real-time error detection, allowing for the identification of faults while the circuit operates normally. These circuits are capable of identifying both temporary and permanent faults [4]. The concept involves integrating additional circuitry into the original functional unit. This supplementary circuit assesses the functional unit's output to determine the existence of any errors [5]. Various codes are available for devising self-checking circuits. This study focuses solely on separable codes, where the information bits and check bits are distinctly separated. The Berger code calculates the quantity of 1's in the word and represents it in binary form. By complementing the binary word and appending

the count to the data, Berger codes serve as optimal systematic AUED (All Unidirectional Error Detecting) codes. The identification of unidirectional bit errors is achievable through its ability to detect instances where one or more ones transition to zeros, while remaining unable to identify cases where zeros transition to ones. In the event of an equal number of bit transitions from zero to one and one to zero, the error will not be discerned [6]. For a Totally Self-Checking (TSC) circuit to function effectively, a single bit output proves inadequate due to the inability to detect a Stuck-at-fault situation on the output yielding an erroneous "good" output [2, 7, 8, 9]. Typically, the output of a TSC circuit undergoes encoding utilizing a two-rail (1-out-of-2) code. The utilization of a two-rail checker unit (TRC) is essential in the comparison of two complementary codewords to ascertain the validity of the output from the functional circuit. [2]. The block diagram in figure 4 shows the proposed design to achieve the self-checking checker for the shift register. The proposed 4 Bit shift Left/Right register it looks like a serial-in/ serial-out shift register with taps added to each stage of Input and output. Serial data shifts in at Din (Serial Input). After a number of clocks equal to the number of bits in the shift register, the first Din bit data appears at (Q4).Entry of the four bits (1001) into the register, beginning with the right most bit. Since data are entered into the register, the Check Symbol Generator (CSG) generate the code word for data, the 3-bit register will store the complementary code word which will be used in two rail inputs for comparing.

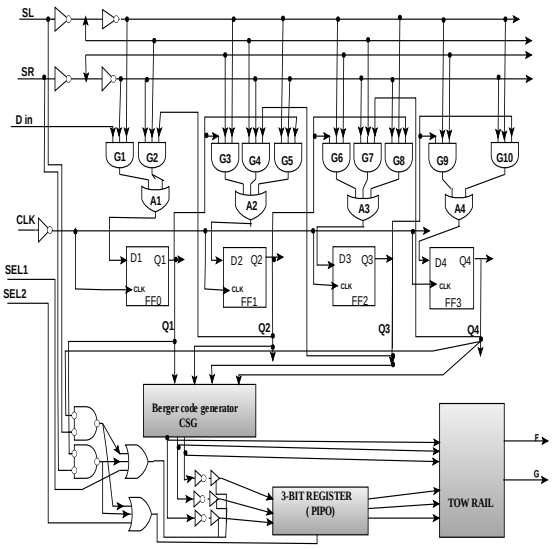

Fig. 4: 4 Bit shift Left/Right register.

Since data are entered into the register, the CSG generates the code word of the data. The 3-bit register will store the complementary code word, since the information bit is expected to be discarded during right/left shift which needs to change the stored code word in the 3-bit register according to the following:

Shift left: If SL=1 and MSB=0, then the number of 1's in the code word will be as it is before shifting to the left, this means that the check symbol will not change if the code word is error free. If SL=1 and MSB=1,a new code word will be generated as the number of 1's decreased by 1, a new generated code word will be stored in the 3-bit register. Shift right: If SR=1 and LSB=0, the stored code word will remain stored. If SR=1 and LSB=1, a new generated code word will be stored in the 3-bit register. The process of conducting checks can be put into action as demonstrated by the visual representation provided in figure 5. The various configurations are translated into Verilog Hardware Description Language (HDL) to facilitate both simulation and synthesis processes. The waveform representations

depicted below exhibit the results generated by the Simulation Processor. Specifically, the waveform in Figure 6 illustrates the functioning of the Reset register, and the subsequent execution of either the shift left or shift right operations on the input data that has been loaded. Consequently, the subsequent signals can be observed to flow accordingly: CLK: The signal denoting the passage of time within a system. Din: The input data being fed into the register.Q1: The first bit of the output data representing the value of the initial bit in the register.Q2: The second bit of the output data indicating the value of the second bit in the register. Q3: The third bit of the output data signifying the value of the third bit in the register.Q4: The fourth bit of the output data showcasing the value of the fourth bit in the register. The table 1 illustrates the control state for the utilized register. Figure 7 presents the waveform of the Berger Code check symbol Generator for a 4-bit system, where the input for CSG is the output for the 4-bit shifter register, encompassing signals such as CLK for the clock signal. Q1: The first bit of the output

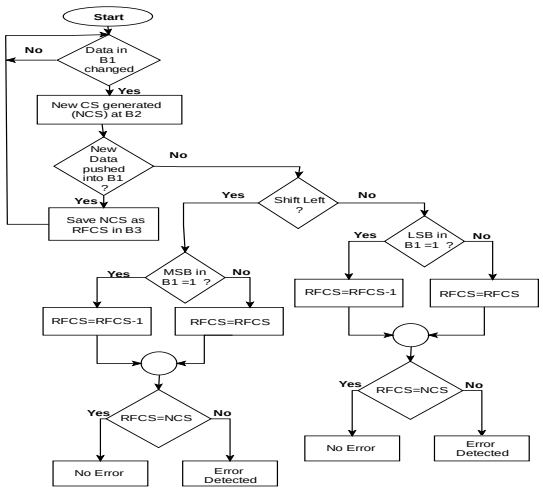

Fig. 5: A flowchart provides a visual representation of the sequential steps.

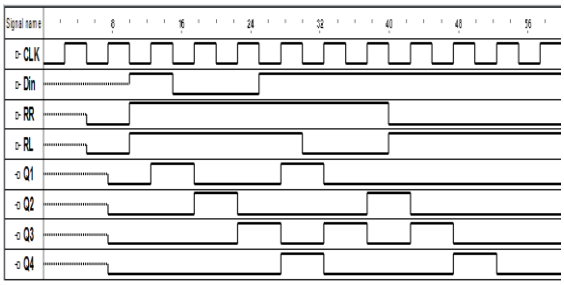

Fig. 6: The waveform depicts the process of transferring data into the shifter register

.

Table 1: control state for the register.

RL	RR	Function
0	0	Resetting register
0	1	Shift data right
1	0	Shift data left
1	1	Load data into register

data displaying the value of the initial bit in the register. Q2: The second bit of the output data showcasing the value of the second bit in the register. Q3: The third bit of the output data representing the value of the third bit in the register.

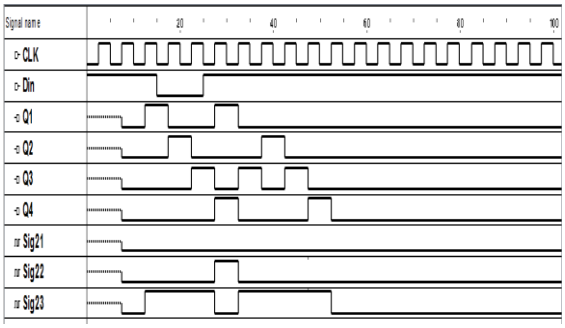

Fig. 7: The waveform exhibits the Berger code check symbol generator.

Q4: The fourth bit of the output data indicating the value of the fourth bit in the register. Sig21: The first bit of the CSG output revealing the value of the initial bit in the CSG output. Sig22: The second bit of the CSG output displaying the value of the second bit in the CSG output. Sig23: The third bit of the CSG output signifying the value of the third bit in the CSG output. The waveform in Figure 8 exhibits the process of loading data into the register, generating the check symbol bits word, pushing the complement of the check symbol bits word to a 3-bit register, shifting the register left/right, transmitting data to a two-rail checker, and finally obtaining the result.

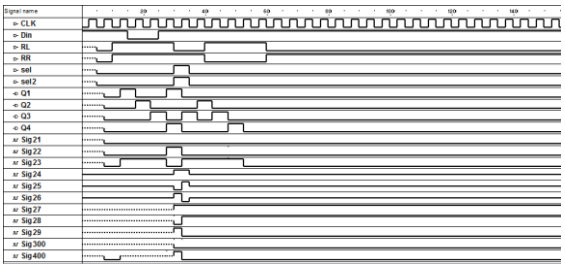

Fig. 8: The waveform portrays the simulation of the shifter implementation processor.

4. Conclusion

This manuscript introduces a methodology for the implementation of fault-tolerant shifters utilizing Berger code. Within this framework, the Berger code generator is employed to produce check bits for every input word in the shift register. This ensures that in the absence of any discrepancies between the information bits and the number of 1's, errors remain undetected. Subsequent to the left/right shifting operation of the register, the Berger code generator is utilized to generate check bits for the respective word. To identify faults post-operation, the stored check bits along with their inverted counterparts are directed to the Two-rail checker. Through this approach, the design attains heightened reliability levels while maintaining reasonable hardware expenditure. The system is proficient in detecting all instances of Unidirectional Error within the shifter.

5. References

[1]. Subhasish Mitra, "Diversity Techniques for Concurrent Error Detection", Technical Report, Center for reliable computing, May 2000.

[2] P. K. Lala. Self-Checking and Fault-Tolerant Digital System Design. Morgan Kaufman Publishers, San Francisco, 2001.

[3] Huda Abugharsa, and Ali Maamar," Self Checking Systolic LIFO Stack",7th WSEAS Int. Conf. on Instrumentation Measurement, Circuits and Systems (IMCAS '08), Hangzhou, China, April 6-8, 2008.

[4] KHADIJA F. O. ALGHEITTA. AMAL J. MAHFOUD. ALI H. MAAMAR. Design of a Self-Checking Up Counter. International Conference on Advanced in Computing, Engineering and Learning Technologies, Abu Dhabi, UAE, 2013.

[5] Mustafa Abd-El-Barr," Design and Analysis of Reliable and Fault-Tolerant Computer Systems ", Imperial College Press, 2007, ISBN 1-86094-668-2.

[6] VARADAN SAVULIMEDU VEERAVALLI. Diagnosis And Error Correction For A Fault-Tolerant Arithmetic And Logic Unit For Medical Microprocessors, Graduate School- New Brunswick Rutgers, The State University of New Jersey October, 2008.

[7] B. W. Johnson. Design and Analysis of Fault Tolerant Digital Systems. Addison-Wesley, Reading, MA, 1989.

[8] D. K. Pradhan. Fault-Tolerant Computing: Theory and Techniques, volume I. Prentice Hall, Englewood Cliffs, New Jersey, 2003.

[9] T. R. Stankovic, M. K. Stojcev, and G. Ordjevic. Design of Self-Checking Combinational Circuits. In Proc. of the International Conf. on Telecommunications in Modern Satellite, Cable and Broadcasting Services, volume 17, pages 763-768, October 2003.