Implementation of Fault Tolerant Method using BCH Code on FPGA

Mahadevaswamy V. P., Sunitha S. L., B. N. Shobha

Abstract*—The Fault tolerance degradation is the property that enables a system (often computer-based) to continue operating properly in the event of the failure of (or one or more faults within) some of its components. To designing a new 32-bit Arithmetic Logic Unit (ALU) that is secure against many attacks or faults and able to correct any 5-bit fault in any position of its 32 bits input register of ALU. Because the radiation effects on electronic circuits may cause to be inverted data bits of registers or memories. If one bit of main storage system is changed the mission of system would be completely different. The high motivation in choice of BCH (Bose, chaudhuri, and Hocquenghem) codes is that, it is able to correct multiple errors and these classes of codes are kind of powerful random error correcting cyclic codes. In comparison with area penalty methods, 32-bit fault tolerant ALU using BCH code is a better choice in terms of area as compared to Triple Modular Redundancy (TMR) and Residue code. This is due to the fault tolerant method for 32-bit ALU using TMR with single or triplicated voting need single voting scheme or tripled voter and two extra 32-bit ALU which has been increased the hardware overhead by 202% and 208% respectively. The Residue code requires hardware overhead of 148.9%. However, in comparison with TMR and Residue code, BCH code needs the hardware overhead is 70 to 75%, which causes that the overall cost and power consumption will get reduces. Thus proposed fault tolerant hardware overhead has lower hardware and multiple error correction when compared to the other techniques.

***Index Terms*—Fault Tolerant, BCH codes, ALU, Residue code, TMR, Encoder, Decoder, FPGA.**

I. INTRODUCTION

This work presents a BCH based hardware implementation of 32-bit Fault-tolerant ALU in which is compared with the current techniques such as Residue code[2][3], TMR with single voting and TMR with triplicated voter that are widely used in space application to mitigate the upsets, in terms of area penalty[7]. The various attacks exist in space on integrated circuits that comes from sun activity. Such as solar rays which are composed of charged particles. The radiation from sun effects in integrated circuits makes digital damage and upsets such as SEU (Single Event Upset), SET (Single Event Transient) [1] and etc. Such attacks can upset either combinational logic or sequential logic. In other words a bit flip can occur in register bits and if one bit of main storage system is changed the mission of system would be completely different. In such scenario the error control or fault tolerant methods are employed to keep integrated circuits against these attacks in space.

This work consider Error Detection and Correction Codes (EDAC) method[4]. It is usually used to mitigate Single Event Upset in integrated circuits which are required that the encoder and the decoder blocks to be able to detect and correct errors respectively. This technique gives strong faults coverage and less overhead hardware. This techniques are requires hardware overhead, increased area and less faults coverage. So for this reason more sophisticated error correcting codes are the BCH codes are considered that are a generalisation of the Hamming codes for multiple-error correction.

The BCH codes operate over finite fields. The BCH codec (encoder, decoder) that is implemented on FPGA hardware[8]. The new implementation of ALU employing BCH code on Spartan-3 FPGA has been provided. The results show reduced area requirements compared to the other technique and it can correct any 5-bit error in any positions of 32-bits input registers of ALU.

A. BCH Code

There are many design based techniques to give the fault tolerant scheme such as detection techniques and mitigation techniques. This technique gives strong faults coverage and less overhead hardware. For this reason we consider the BCH codes and a binary BCH codes is considered. As a result of using BCH codes, we have achieved to design encoder and decoder circuits to detect and correct 5-bit faults.

Encoding Technique: The BCH codes are implemented as cyclic codes [6][8], that is, the digital logic implementing the encoding and decoding algorithms is organised into shift-register circuits that mimic the cyclic shifts and polynomial arithmetic required in the description of cyclic codes. Using the properties of cyclic codes, the remainder b(x) can be obtained in a linear (n-k)-stage shift register with feedback connections corresponding to the coefficients of the generator polynomial $g(x) = 1 + g_1x + g_2x^2 + ... + g_{n-k-1}x^{n-k-1} + x^{n-k}$.

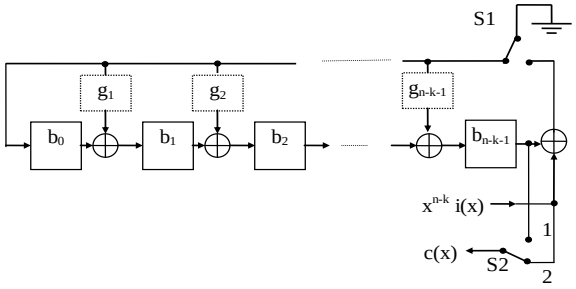

Figure 1. Encoding circuit for a (n, k) BCH code

The encoder shown in Figure 1 operates as follows

- For clock cycles 1 to k, the information bits are transmitted in unchanged form (switch S2 in position 2) and the parity bits are calculated in the LFSR (switch S1 is on).

Manuscript received September 02, 2012.

Mahadevaswamy V P, ECE, Visvesvaraya Technological University, Mandya, India,
Dr. Sunitha S.L., ECE, Visvesvaraya Technological University, Mandya, India,
B.N. Shobha, ECE, Visvesvaraya Technological University, Banglore, India,

- For clock cycles k+1 to n, the parity bits in the LFSR are transmitted (switch S2 in position 1) and the feedback in the LFSR is switch off (S1 - off).

Decoding Technique: The decoding of BCH code is composed of three main steps that are expressed as Calculating the syndromes, Solving the key equation and Finding the error locations.

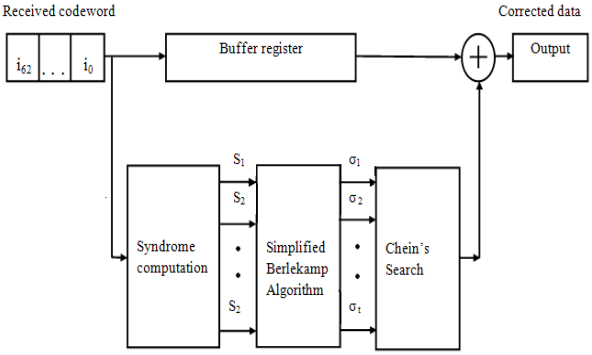

Figure 2. Block diagram for decoder system using BCH code.

Using these steps identify the position of erroneous bit. All these steps shown in the following block diagram in figure 2.Fortunately, for some BCH codes step number 2 can be omitted. To decode BCH codes in this work, three different strategies have been employed, for Single Error Correcting, Double Error Correcting and Triple and More Error Correcting BCH codes.

Regarding step 1, the calculation of the syndromes is identical for all BCH codes. For Single Error Correcting codes step number 2 - solving the key equation - can be omitted, as a syndrome gives rise to the error location polynomial coefficient. For Double Error Correcting codes step number two can also be omitted but the error location algorithm is rather more complicated. Finally, when implementing the TMEC decoding algorithm all three steps must be carried out, where step 2 - the solution of the key equation is the most complicated.

II. IMPLEMENTATION OF BCH CODE

As shown in figure 3. From the Algorithm we generate RTL (Register transfer level) code. This implies that our Verilog code describes how data is transformed as it is passed from register to register. The transforming of the data is performed by the combinational logic that exists between the registers. RTL code also applies to pure combinational logic.

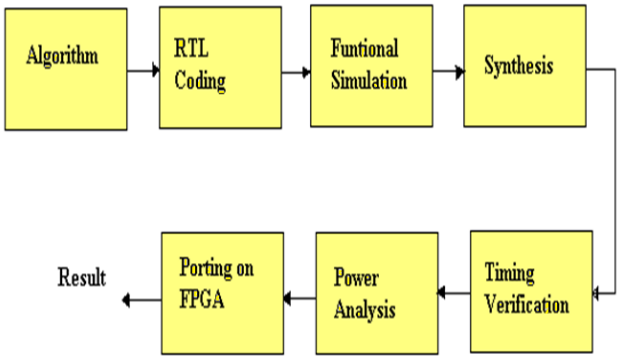

Figure 3. Implementation Block Diagram of 32-bit Fault Tolerant ALU Methods Using BCH code on FPGA

The Design verification is an important aspect of each work design. Before implementing circuit in the target device. Functional simulation can be done after a Verilog file has been created and synthesized. Functional simulation gives information about the logic operation of the circuit. It does not provide any information about timing delays. On the other hand, timing simulation will tell you how fast signals travel through the gates and how fast the overall circuit can be operated.

In order to do a timing simulation, one needs to implement the design in a specific target device. The timing Verification will give you detailed information about the time it takes for a signal to pass from one gate to the other (gate delay) and gives information on the circuit worst-case conditions. The total delay of a complete circuit will depend on the number of gates the signal sees and on the way the gates have been placed in the FPGA or CPLD. Then we perform the power analysis to verify the reduction in power.

III. RESULTS

The fault tolerant ALU has been implemented in XC3S400 from Spartan-3 FPGA family and the system has been simulated on Modelsim 6.2b and its performance has been verified by ISE 13.1i. The Figure 4 shows the simulation results. In the comparison of the previous methods with using the BCH code and TMR, the result implies that the performance of the proposed fault tolerant ALU algorithm, an encoding and decoding block have partially degraded. This is due to the delay of the number of XOR gates in serial form in codec circuits.

Moreover, as the number of error bits increases, the time for error correction may take longer. In the TMR method the delay is occurred in the voter scheme, then the performance is not deeply affected and it is constant with the number of bits to be detected. In comparison with area penalty of methods, 32-bit fault tolerant ALU using BCH code is a better choice in terms of area as compared to TMR and Residue code. This is due to the fault tolerant method for 32-bit ALU using TMR with single or triplicated voting need single voting scheme or tripled voter and two extra 32-bit ALU which has been increased the hardware overhead by 202% and 208% respectively. In comparison with fault tolerant method using Residue code, we need Hardware duplication for boolean operations, residue codes for arithmetic operation and extra 32-bit ALU which has been increased the hardware overhead by 148.9%.

However, in comparison with fault tolerant method using BCH code, we need encoding and decoding block then the hardware overhead is 70 to 75%, which causes that the overall cost and power consumption will get reduces. Thus our fault tolerant hardware overhead has lower hardware and multiple error correction when compared to the others technique of TMR and Residue code.

Top_Module Project Status			
Project File:	bch5.xise	**Parser Errors:**	No Errors
Module Name:	Top_Module	**Implementation State:**	Synthesized
Target Device:	xc4vfx12-12sf363	**•Errors:**	No Errors
Product Version:	ISE 13.1	**•Warnings:**	105 Warnings (93 new)
Design Goal:	Balanced	**•Routing Results:**	
Design Strategy:	Xilinx Default (unlocked)	**•Timing Constraints:**	
Environment:	System Settings	**•Final Timing Score:**	

Device Utilization Summary (estimated values)			[-]
Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slices	312	5472	5%
Number of Slice Flip Flops	387	10944	3%
Number of 4 input LUTs	516	10944	4%
Number of bonded IOBs	141	240	58%
Number of GCLKs	1	32	3%

Detailed Reports					
Report Name	**Status**	**Generated**	**Errors**	**Warnings**	**Infos**
Synthesis Report	Current	Fri Jun 8 18:58:39 2012	0	105 Warnings (93 new, 0 filtered)	0
Translation Report	Out of Date	Tue Jun 5 11:27:46 2012	0	0	0
Map Report	Out of Date	Tue Jun 5 11:29:30 2012	0	0	4 Infos (4 new, 0 filtered)
Place and Route Report	Out of Date	Tue Jun 5 11:30:04 2012	0	0	4 Infos (4 new, 0 filtered)
Power Report					
Post-PAR Static Timing Report	Out of Date	Tue Jun 5 11:30:14 2012			
Bitgen Report					

Table 1 shows the performance status

The Table 1 shows the performance status of BCH code like project file, module name, target device, project version, design goal and also it gives utilization summary. Our project uses 312 slice registers among 5472 and also it uses 516 slice LUTs among 10944. It uses 141 bounded IOBs among 240. It uses 58% bounded IO blocks and overall hardware overhead of BCH Code is 73%. Its performance has been verified by ISE 13.1i.

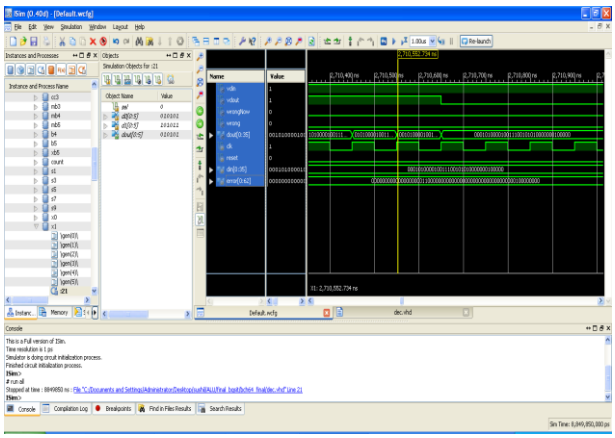

Figure 4. Shows the simulation results

The fault tolerant ALU has been implemented in XC3S400 from Spartan-3 FPGA family and the system has been simulated on Modelsim 6.2b and its performance has been verified by ISE 13.1i. The figure 4 shows the simulation results. In the comparison of the previous methods with using the BCH code and Triple Modular Redundancy, the result implies that the performance of the proposed fault tolerant ALU algorithm, an encoding and decoding block have partially degraded. This is due to the delay of the number of XOR gates in serial form in codec circuits.

IV. CONCLUSION

The work describes a new implementation of the ALU for BCH code .We also compared our 32-bit fault tolerant ALU by using a (63, 36) BCH code with the other Fault tolerant methods (Residue code, Triple Modular Redundancy with single voting scheme and Triple Modular Redundancy with triplicated voting method).

In the comparisons, for instance, fault tolerant method using BCH shows 70 to 75% hardware overhead. The implementation provides a high level of fault tolerance with relatively and small hardware penalty. Further, the proposed system has been simulated on Modelsim 6.2b and its performance has been verified by ISE 13.1i. The results indicate any five bits error in any position of 32-bit input registers of Arithmetic Logic Unit would be corrected.

REFERENCES

1. S.Bourdarie and M.Xapsos, senior member, IEEE, "The near earth space radiation environment", IEEE Transaction on Nuclear Science, August, 2008.
2. Veeravalli, V.S. "Fault tolerant Arithmetic and Logic Unit", IEEE international conference, Rutgers State Univ. of New Jersey, Piscataway, NJ, USA, March, 2009.
3. R.Hentschke, F.Marques, F.Lima, L.Carro et al. "Analyzing area and performance penalty of protecting different digital modules with Hammnig code and Triple Modular redundancy". IEEE International Conference on Integrated Circuits and Systems Design, 2002.
4. Fernanda Lima Kastensmidt, L.Carro, R.Reis, "Fault tolerant techniques for SRAM- Based FPGA" June, 2006.
5. Israel Koren and C.Mani K rishna. "Fault- Tolerant System".Morgan Kaufmann Publishers 2007.
6. W.W. Peterson, "Encoding and error-correction procedures for the Bose-Chaudhuri Codes", IRE Trans. Inf. Theory, IT-6, pp. 459-470, September 1960.
7. Lin, Shu, and Daniel J. Costello, Jr., "Error Control Coding: Fundamentals and Applications", Englewood Cliffs, NJ, Prentice-Hall, 1983.
8. Vahid Khorasani, B.Vousoghi et al. "Designing a secure 32-bit ALU using (63, 36) BCH code", Worldcomp conference, July, 2011.

AUTHORS PROFILE

Mahadevaswamy V. P., was born in Shivapura, Mysore, in 1982. I received the B.E. degree in ECE from VTU, Belgaum, Karnataka, in 2008 and M.Tech degree in VLSI Design and Embedded systems from VTU, Belgaum in 2012,