International Journal of
Emerging Research in Management &Technology
ISSN: 2278-9359 (Volume-4, Issue-10)

Research Article | **October 2015**

Design of 32-Bit Fault Tolerant ALU using VHDL

Nidhi Awasthi, Saifur Rahman
Department of Electronics and Communication
Integral University, Lucknow U.P., India

Abstract-

The original contribution of the work in this paper is to understand the sources of errors during transmission system for any Arithmetical or Logical computation. In this work the BCH codes for Error detection and Error Correction upto 5 bit in any location of 32 bit data at both end of ALU Inputs are presented. Each transmitted bit has probability p ≥ 0 being received incorrectly. On memory less channels every transmitted symbol may be considered independently, so only random errors occur. Unfortunately, most channels have memory and usually several successive symbols are corrupted. These kinds of errors are called burst errors. Burst errors can be most efficiently corrected through use of burst error correcting codes, e.g. Reed Solomon (RS) codes/ BCH codes. The simplest block codes are Hamming codes. They are capable of correcting only one random error and therefore are not practically useful, unless a simple error control circuit is required. More sophisticated error correcting codes are the Bose, Chaudhuri and Hocquenghem (BCH) codes that are a generalisation of the Hamming codes for multiple-error correction. In this paper the subclass of binary, random error correcting BCH codes is considered, hereafter called BCH codes. BCH codes operate over finite fields or Galois fields. The mathematical background concerning finite fields is well specified and in recent years the hardware implementation of finite fields has been extensively studied.

Keywords- A.L.U., BCH Code, Fault Tolerance, VHDL, Xilinx.

I. INTRODUCTION

Transmission of information through a practical communication system is not free from noise. Information may rather be corrupted by noise in the channel. Therefore it is necessary for communication systems to have adequate means for the detection and correction of errors in the information received over communication channels. BCH codes, turbo codes and LDPC (low density parity check) codes are most commonly used for error detection and correction nowadays. LDPC codes are linear codes that are used for error correction with the help of sparse bipartite graph. Error correcting codes have been successfully implemented in wire-line and wireless communication to offer error-free transmission with high spectral efficiency. Error correcting codes are generally used with communication of digitally encoded data in order to reduce errors.

In digitally encoded data, there is a block of symbols generally in the form of 0 and 1. Suppose that we want to transmit the information that "There is no class on Monday". This information can be defined by 10110, say, and transmitted over a channel where some "noise" may be introduced. Noise means simply errors. The aim of an error-correcting code is to lengthen the message is such a way that the original message can be recovered even if errors are present here. By error it means that there is unwanted change in data which is required to be correct for efficient and reliable reception of data.

The following diagram represents the communications channel.

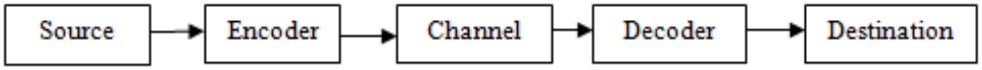

Fig. 1: Communication Systems

Now suppose that due to presence of noise, second digit of information is changed from 0 to 1, and then decoder is able to correct it with the help of extra bits. The process of error correction uses redundancy means some extra data is added to a message, which receivers can use to check consistency of the delivered message, and to recover data determined to be corrupted.

II. PROPOSED WORK

In this procedure BCH (63,36) encoder circuit have been designed to find the codeword and BCH (63,36) decoder have been designed to correct any faults in codeword. The designing of (63, 36) BCH encoder and decoder for t=5 has been provided in [4] by the way encoder and decoder has been described and implemented summarily.

The encoder circuit [2], calculates the parity bits using the LFSR (Linear Feedback Shift Register). The generator polynomial of the (63, 36) BCH code is:

$$G(X)=X^{27}+X^{22}+X^{21}+X^{19}+X^{18}+X^{17}+X^{15}+X^{8}+X^{4}+X^{1}+1$$

Encoding circuits are shown in fig. 2

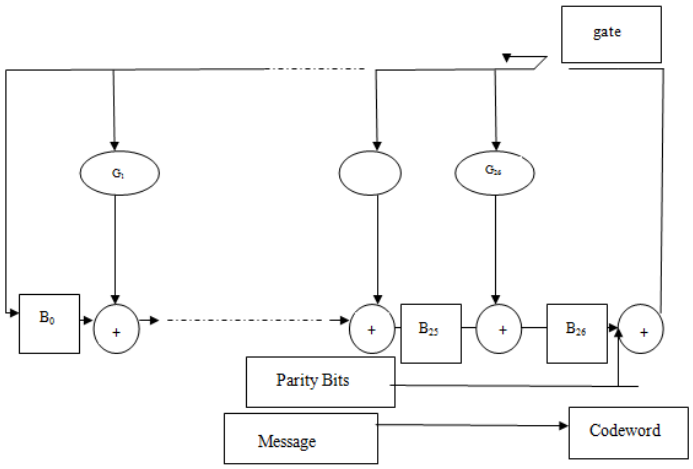

Fig 2: Encoding circuit for an BCH code

The decoding of BCH code is composed of three main steps that are expressed as follows:
1) Compute the syndromes from the received codeword.
2) Obtain the error locator polynomial σ(x) (ELP) through the BMA (Berlekamp-Massy-Algorithm).
3) Determine the error-location numbers by finding the roots of error location polynomial (identifying the position of erroneous bit). All these steps shown in the following block diagram in fig 3.

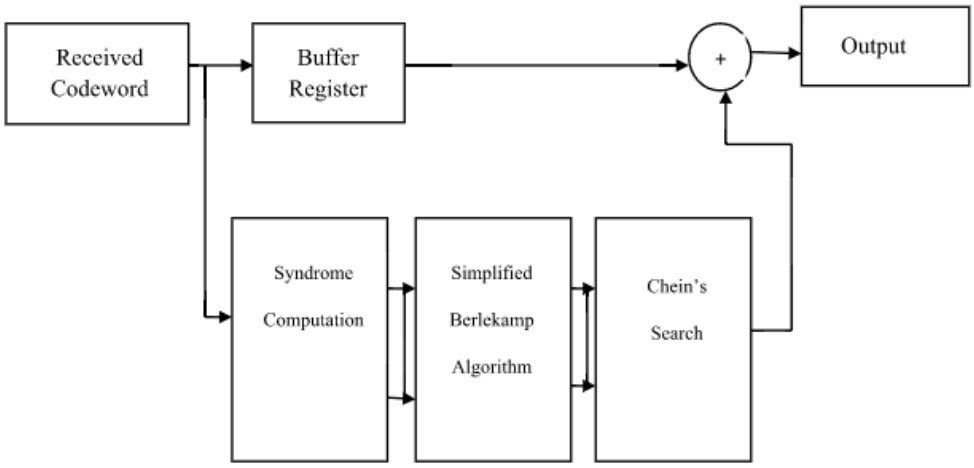

Fig. 3: Block diagram for decoder system using BCH code

2.1 The syndrome computations:
The first step of decoding procedure is "syndrome computation". The syndromes identify whether error has occurred. If the syndromes all are zero, there will be no error in codeword and if the syndromes not be zero there will be error in codeword. For computing the syndromes, the syndrome S_i is defined as:

$$S_i = r(\alpha^i) = r_{n-1}\alpha^{(n-1)i}+r_{n-2}\alpha^{(n-2)i}+.....+r_0$$
$$=...((r_{n-1}\alpha^i+r_{n-1})\alpha^i+r_{n-3})\alpha^i+....+r_1)\alpha^i+r_0$$

Where i is $1\leq i \leq 2t$. Each syndrome component is calculated by dividing r(x) by the minimal polynomial mi(x) of α^i.

$$r(x) = q_i(x)m_i(x) + b_i(x)$$

$b_i(x)$ is the reminder. When the entire received codeword has entered the decoder, 10 syndromes components (s_1, s_2,…, s_10) are formed. It takes 63 clock cycles to complete the computation. Since, the generator polynomial is a product of at most 5 minimal polynomials Therefore at most 5 feedback shift register , each consist of at most 6 stages, are required to form the 10 syndrome components. For more information about the computation of syndrome refer to [4].The syndrome computation circuit for (63, 36, 5) BCH code which has been presented in [4] is also implemented.

2.2 The Berlekamp-Massy Algorithm:
The second step of decoding for finding the error location polynomial has been done through the simplified Berlekamp-Massy Algorithm which is shown in fig 4. It is assumed that the numbers of errors $v \leq t$ have occurred and error locator polynomial σ(x) is:

$$\sigma(x)= \sigma_0 + \sigma_1 x^1+.....+ \sigma_v x^v$$
$$\sigma(x)=(1+\beta_1 x)(1+\beta_2 x).....(1+\beta_v x)$$

The coefficient of error locator polynomial and the error location numbers are related by the following set of equations: [3].

$$\sigma_0=1$$
$$\sigma_1=\beta_1+\beta_2+.....+\beta_v$$

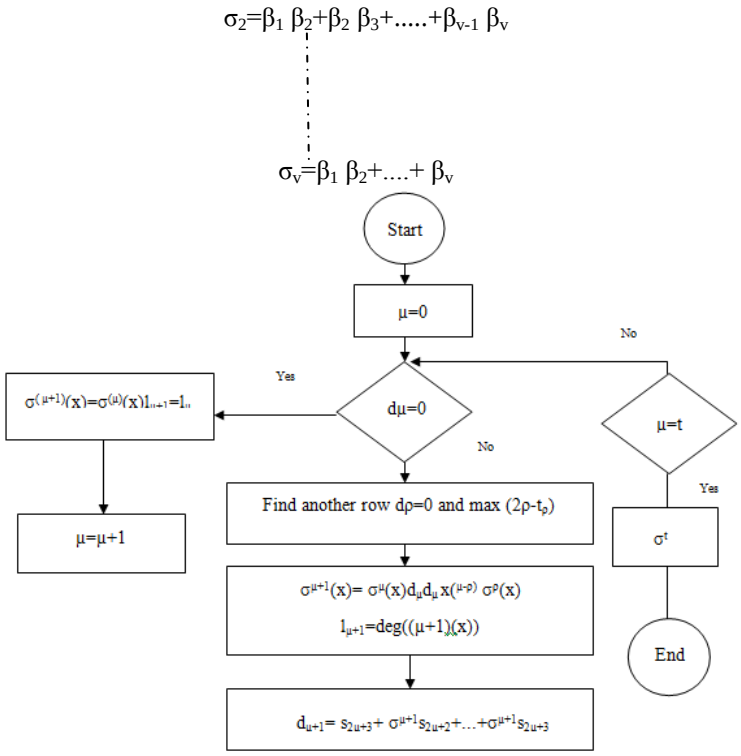

Fig. 4:The inversion Berlekamp massy algorithm

2.3 Chein's searching circuit:

This process stored the received codeword in a buffer register to compute the syndrome. It takes 63 clock cycles to complete the computation. A Chein's searching circuit [2], for the 5-error correcting (63, 36) BCH has been implemented in [4]. This circuit is applied to identify the position of erroneous bits into the 63-bit received codeword and then correct it.

III. FAULT TOLERANT ALU USING BCH CODE

This 32-bit ALU model consist of the following operations Full Adder, Subtractor XOR, AND, OR, NOT, shifting to the left and right. The codec circuits are applied to correct any

5-bit error occurred in any position of 32-bits input registers of ALU. Our algorithm of fault tolerant ALU is shown in Fig. 5. In this algorithm at first the 63 bits input register A and B is read out one by one, by the decoding system. Registers A, B consist of 27-bits parity check; 36-bits data which are 4-bits are extra. These extra bits are used as parity check bits as shown in fig. 5. If any error occurs in any position of 63-bits the decoder will correct the erroneous bit at once. In output the decoding system gives 36-bits where only 32-bits are needed as 2-inputs of ALU (here 4-bits of 36-bits are not stored). So, the output of 32-bit ALU is the C resister. The 4-bits zero are added for leftmost of C register to convert the 36-bits for the input of encoder system. After encoding the data of register C´ is kept in register A.

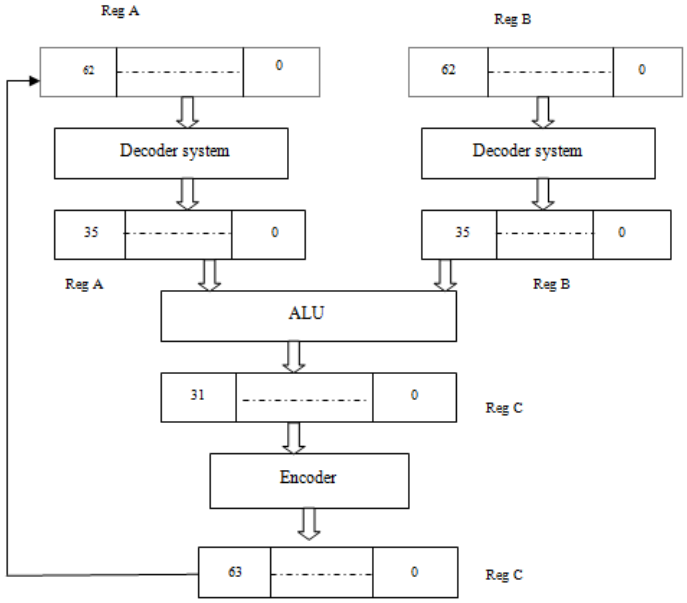

Fig. 5: The fault tolerant ALU Algorithm using BCH code

IV. RESULTS AND DISCUSSION

In this fig 6 a complete simulation for 32 bit ALU operation has been shown. Two 32 bit inputs a and b as well as 5 bit error for both signal have been given so in this case it can show that our ALU is working for maximum 5 bit of error for each signal that is automatically resolved. Here 3 bit opcode "000" has been taken that means it will give 1's complement of input a. In this case value of a is "00000011" is taken in hexadecimal format and value of b is "01010101" in hexadecimal format. The output of ALU is "FFFFFFEE". That is a desired output.

Fig. 6:32 bit ALU operation with 5 bit error

In this fig 7 a complete simulation for 32 bit ALU operation has been shown. Two 32 bit inputs a and b as well as more than 5 bit error for both signal have been given so in this case it can be shown that this ALU is working for maximum 5 bit of error for each signal that is automatically resolved. 3 bit opcode "000" has been taken that means it will give 1's complement of input a. In this case value of a is taken "00000011" in hexadecimal format and value of b is taken "01010101" in hexadecimal format. the output is "FFFFFFE1". That is not a desired output.

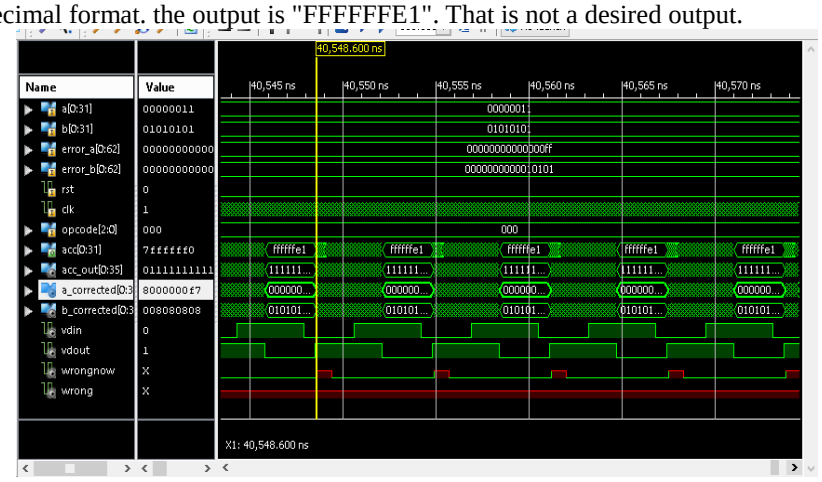

Fig 7:32 bit ALU operation with more than 5 bit error

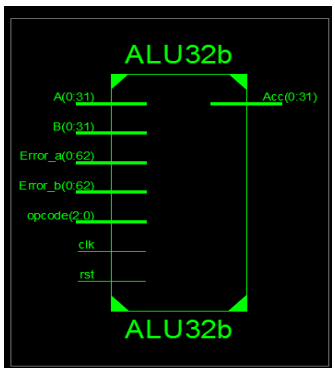

Fig. 8:Fault Tolerant ALU Main RTL Schematic

V. CONCLUSION

In conclusion the design of BCH codecs is rather difficult and the BCS system significantly shortens the time taken to design BCH codecs. In addition the system has been thoroughly simulated and is less error-prone than hand-crafted designs. However it should be noted that in some cases a modification of a basic BCH code is required, e.g. shortened or

extended BCH codes. Furthermore a different input/output format may be required or an alarm signal needed to be asserted if an uncorrectable error pattern has occurred. Finally, it is worth noting that the final VHDL files are almost gate level descriptions and that the resultant circuits are as hardware efficient as hand-crafted ones developed for just one set of parameters. Hence there are no hardware penalties incurred by using the BCS system instead of designing a BCH codec oneself, but obviously using the BCS system saves many design man hours.

REFERENCES

[1] Fernanda Lima Kastensmidt, L.Carro, R.Reis, "Fault tolerant techniques for SRAM-based FPGA" June, 2006.
[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] Lin, Shu, and Daniel J. Costello, Jr., "Error Control Coding: Fundamentals and Applications", Englewood Cliffs, NJ, Prentice-Hall, 1983.
[4] Vahid Khorasani, B.Vousoghi et al. " Designing a secure 32-bit ALU using (63, 36) BCH code", Worldcomp conference, July, 2011.
[5] D. A. Anderson. Design of Self-Checking Digital Networks Using Coding Techniques. Research Report # 527, Univ. of Illinois, Urbana Champaign, Coordinated Science Lab., September 1971.
[6] A. Vahid Khorasani et. al., "Analysis of 32-bit Fault Tolerant ALU Methods" 2010.
[7] Martin Straka et.al., " Fault Tolerant Structure for SRAM-based FPGA via Partial Dynamic Reconfiguration" 2010.
[8] Federico Baronti et.al., " Design and Verification of Hardware Building Blocks for High-Speed and Fault-Tolerant In-Vehicle Networks" IEEE transactions on industrial electronics, vol. 58, no. 3, march 2011.
[9] Vahid Khorasani et.al., " Analyzing Area Penalty of 32-bit Fault Tolerant ALU Using BCH Code" 2011 14th Euromicro Conference on Digital System Design.
[10] Mahadevaswamy V P et.al., " Implementation of Fault Tolerant Method Using BCH Code on FPGA" International Journal of Soft Computing and Engineering (IJSCE) ISSN: 2231-2307, Volume-2, Issue-4, September 2012.
[11] Jaimini Patel, Deepali H. Shah. "A realisation of Fault tolerant ALU-with TMR method", International Journal of Engineering Development and Research (IJEDR), ISSN:2321-9939, Vol.2, Issue 3, pp.3161-3164, Sept 2014.
[12] Rakshith, T.R**.** "Parity preserving logic based fault tolerant reversible ALU", Information and communication technologies (ICT), 2013 IEEE Conference

