Journal of Scientific and Engineering Research, 2015, 2(1):1-15



Research Article

ISSN: 2394-2630 CODEN(USA): JSERBR

Design of Hamming Encoder and Decoder Circuits For (64, 7) Code and (128, 8) Code Using VHDL

Adham Hadi Saleh

Department of Electronic engineering, University of Diyala, Iraq

Abstract In this paper, we have described how we can generate redundancy bit for (64 and 128) information data bits. These redundancy bits are to be interspersed at the bit positions (n = 1, 2, 4, 8, 16, 32, 64 and 128) of the original data bits. So to transmit 64 bit information data we need 7 redundancy bit to make 71 bit data string and 8 redundancy bit to make 136 bit data string. At the destination receiver point, the data may be corrupted due to noise. In Hamming technique the receiver will decide if data have an error or not, so if it detected the error it will find the position of the error bit and corrects it. This paper presents the design of the transmitter and the receiver with Hamming code redundancy technique using VHDL for (64 and 128) input data. The Xilinx ISE 10.1 Simulator was used for simulating VHDL code for both the transmitter and receiver sides.

Keywords Hamming code, error correction, error detection, even parity check method, Redundancy bits, VHDL language, Xilinx ISE 10.1 Simulator

1. Introduction

The theory of linear block codes is well established since many years ago. In 1948 Shannon's work showed that any communication channel could be characterized by a capacity at which information could be reliably transmitted. In 1950, Hamming introduced a single error correcting and double error detecting codes with its geometrical model [1].

In telecommunication, Hamming code as a class of linear block codes is widely used, Hamming codes are a family of linear error-correcting codes that generalize the Hamming (7,4)-code. Hamming codes can detect up to two-bit errors or correct one-bit errors. By contrast, the simple parity code cannot correct errors, and can detect only an odd number of bits in error. Hamming codes are perfect codes, that is, they achieve the highest possible rate for codes with their block length and minimum distance 3 [2-3].

Due to the limited redundancy that Hamming codes add to the data, they can only detect and correct errors when the error rate is low. This is the case in computer memory (Error Checking & Correction, ECC memory), where bit errors are extremely rare and Hamming codes are widely used. In this context, an extended Hamming code having one extra parity bit is often used. Extended Hamming codes achieve a Hamming distance of 4, which allows the decoder to distinguish between when at most one bit error occurred and when two bit errors occurred. In this sense, extended Hamming codes are single-error-correcting (SED) and double-error-detecting (DED). The ECC functions described in this application note are made possible by Hamming code, a relatively simple yet powerful ECC code. It involves transmitting data with multiple check bits (parity) and decoding the associated check bits when receiving data to detect errors. The check bits are parallel parity bits generated from XORing certain bits in the original data word. If bit error(s) are introduced in the codeword, several check bits show parity errors after decoding the retrieved codeword. The combinations of these check bit errors display the nature of the error. In addition, the position of any single bit error is identified from the check bits [2,4].

Error detection and correction codes are used in many common systems including: storage devices (CD, DVD, and DRAM), mobile communication (cellular telephones, wireless, and microwave links), digital television, and high-speed modems. Hamming codes is a Forward Error Correction (FEC), as a fundamental principle of channel coding techniques, provides the ability to correct transmission errors without requiring a feedback channel for a correct retransmission. The exact correction capability of an FEC code varies depending on the coding schemes used [5,6].

The basic idea for achieving error detection is to add some redundancy bits to the original message to be used by the receivers to check consistency of the delivered message and to recover the correct data. Error-detection schemes can be either systematic or non-systematic: In a systematic scheme the transmitter sends the original data and attaches a fixed number of check bits. That is derived from the data bits by some deterministic algorithm. If only error detection is required a receiver can simply apply the same algorithm to the received data bits and compare its output with the received check bits if the values do not match an error has occurred at some point during the transmission. In a system that uses a non-systematic code the original message is transformed into an encoded message that has at least as many bits as the original message. Error correction & detection Hamming code may perform using Even parity or Odd parity [7,8].

2. Error Detection and Correction

For a given practical requirement, detection of errors is simpler than the correction of errors. The decision for applying detection or correction in a given code design depends on the characteristics of the application. When the communication system is able to provide a full duplex transmission (that is, a transmission for which the source and the destination can communicate at the same time, and in a two way mode, as it is in the case of telephone connection, for instance), codes can be designed for detecting errors, because the correction is performed by requiring a repetition of the transmission [3,8].

These schemes are known as automatic repeat request (ARQ) schemes. In any ARQ system there is the possibility of requiring a retransmission of a given message. There are on the other hand communication systems for which the full-duplex mode is not allowed. An example of one of them is the communication system called paging, a sending of alphanumerical characters as text messages for a mobile user. In this type of communication system, there is no possibility of requiring retransmission in the case of a detected error, and so the receiver has to implement some error-correction algorithm to properly decode the message. This transmission mode is known as forward error correction (FEC) [3,8].

3. Hamming Code

Hamming code is a linear error-correcting code named after its inventor, Richard Hamming. Hamming codes can detect up to two simultaneous bit errors, and correct single-bit error. By contrast, the simple parity code cannot correct errors, and can only detect an odd number of errors. In 1950 Hamming introduced the (7, 4) code. It encodes 4 data bits into 7 bits by adding three parity bits. Hamming (7, 4) can detect and correct single – bit errors. With the addition of overall parity bit, it can also detect (but not correct) double bit errors. Hamming code is an improvement on parity check method. It can correct 1 error bit only [9].

Hamming code used two methods (even parity and odd parity) for generating redundancy bit. The number of redundancy bits depends on the size of information data bits as shown below [8,9,10,11]:

$$2^r \geq \ m+r+1$$

(1)

Where r = number of redundancy bit.

m = number of information data bits.

According to (1), 7 redundancy bits required for a 64 input data bits and 8 redundancy bits required for 128 input data bits. Hamming-based codes are widely used in memory systems for reliability improvements. The algorithm consists of two phases: encoding and decoding. Hamming encoding involves deriving a set of parity check bits over data bits. These parity check bits are concatenated or merged with the data bits. These extra bits are called redundancy bits. We add these redundancy bits to the information data at the source end and remove at destination end. Presence of redundancy bit allows the receiver to detect or correct corrupted bits. The

concept of including extra information in the transmission for error detection is a good one. But in place of repeating the entire data stream, a shorter group of bits may be added to the end of each unit. This technique is called redundancy because the extra bits are redundant to the information [8,12,13,14].

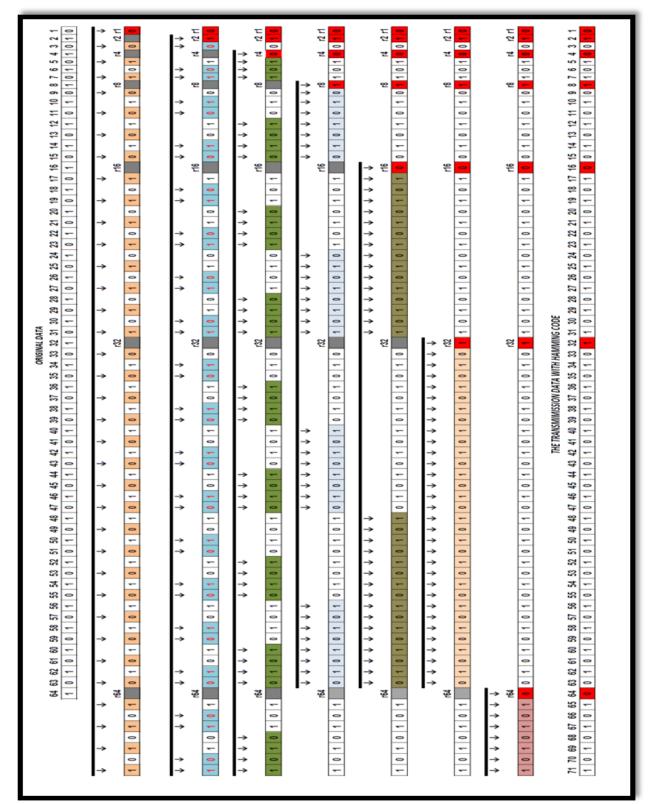


Figure 1: Hamming Code Generation for 64 Bits

4.Hamming Encoder with (64, 7) Code and (128,8) Code

In communication system need two main part one of them is the source for sending data and another is the destination to receive the transmitted data. Even parity check method count the number of one's if number of one's are even it adds zero (0) otherwise it adds one (1) [8].

At the transmitter the 64 bit information data needs 7 redundancy bit according to equation (1). Suppose, these redundancy bits are R(1),R(2),R(4),r(8),R(16),R(32),R(64),and to calculate these redundancy bits easily done by XORing operation of the original data bit positions as shown below:

 $\begin{aligned} R(1) &= D1 \bigoplus D2 \bigoplus D4 \bigoplus D5 \bigoplus D7 \bigoplus D9 \bigoplus D11 \bigoplus D12 \bigoplus D14 \bigoplus D16 \bigoplus D18 \bigoplus D20 \bigoplus D22 \bigoplus D24 \bigoplus D26 \bigoplus D27 \bigoplus D29 \bigoplus D31 \bigoplus D33 \bigoplus D35 \bigoplus D37 \bigoplus D39 \bigoplus D41 \bigoplus D43 \bigoplus D45 \bigoplus D47 \bigoplus D49 \bigoplus D51 \bigoplus D53 \oplus D55 \oplus D57 \oplus D58 \oplus D60 \oplus D62 \oplus D64 \end{aligned}$

 $\begin{array}{c} R(2) = D1 \bigoplus D3 \bigoplus D4 \bigoplus D6 \bigoplus D7 \bigoplus D10 \bigoplus D11 \bigoplus D13 \bigoplus D14 \bigoplus D17 \bigoplus D18 \bigoplus D21 \bigoplus D22 \bigoplus D25 \bigoplus D26 \bigoplus D28 \bigoplus D29 \bigoplus D32 \bigoplus D33 \bigoplus D36 \bigoplus D37 \bigoplus D40 \bigoplus D41 \bigoplus D44 \bigoplus D45 \bigoplus D48 \bigoplus D49 \bigoplus D52 \bigoplus D53 \oplus D56 \oplus D57 \oplus D59 \oplus D60 \oplus D63 \oplus D64. \end{array}$

 $\begin{array}{l} R(4) = D2 \bigoplus D3 \bigoplus D4 \bigoplus D8 \bigoplus D9 \bigoplus D10 \bigoplus D11 \bigoplus D15 \bigoplus D16 \bigoplus D17 \bigoplus D18 \bigoplus D23 \bigoplus D24 \bigoplus D25 \bigoplus D26 \bigoplus D30 \bigoplus D31 \bigoplus D32 \bigoplus D33 \bigoplus D38 \bigoplus D39 \bigoplus D40 \oplus D41 \bigoplus D46 \oplus D47 \oplus D48 \bigoplus D49 \oplus D54 \oplus D55 \oplus D56 \oplus D57 \oplus D61 \oplus D62 \oplus D63 \oplus D64. \end{array}$

 $\begin{array}{l} R(8) = D5 \bigoplus D6 \bigoplus D7 \bigoplus D8 \bigoplus D9 \bigoplus D10 \bigoplus D11 \bigoplus D19 \bigoplus D20 \bigoplus D21 \bigoplus D22 \bigoplus D23 \bigoplus D24 \bigoplus D25 \bigoplus D26 \bigoplus D34 \bigoplus D35 \bigoplus D36 \bigoplus D37 \bigoplus D38 \bigoplus D39 \bigoplus D40 \oplus D41 \bigoplus D50 \oplus D51 \oplus D52 \bigoplus D53 \bigoplus D54 \bigoplus D55 \oplus D56 \oplus D57. \end{array}$

 $\begin{array}{l} R(16) = D12 \bigoplus D13 \bigoplus D14 \bigoplus D15 \bigoplus D16 \bigoplus D17 \bigoplus D18 \bigoplus D19 \bigoplus D20 \bigoplus D21 \bigoplus D22 \bigoplus D23 \bigoplus D24 \bigoplus D25 \bigoplus D26 \bigoplus D42 \bigoplus D43 \bigoplus D44 \bigoplus D45 \bigoplus D46 \bigoplus D47 \oplus D48 \bigoplus D49 \bigoplus D50 \oplus D51 \bigoplus D52 \bigoplus D53 \bigoplus D54 \oplus D55 \oplus D56 \oplus D57. \end{array}$

 $\begin{array}{l} R(32) = D27 \oplus D28 \oplus D29 \oplus D30 \oplus D31 \oplus D32 \oplus D33 \oplus D34 \oplus D35 \oplus D36 \oplus D37 \oplus D38 \oplus D39 \oplus D40 \oplus D41 \oplus D42 \oplus D43 \oplus D44 \oplus D45 \oplus D46 \oplus D47 \oplus D48 \oplus D49 \oplus D50 \oplus D51 \oplus D52 \oplus D53 \oplus D54 \oplus D55 \oplus D56 \oplus D57. \end{array}$

 $R(64) = D58 \bigoplus D59 \bigoplus D60 \bigoplus D61 \bigoplus D62 \bigoplus D63 \bigoplus D64.$

The value of redundancy bits can be calculated using an even parity check method. The value of redundancy bit can be calculated by XORing of different locations of information data bits, as shown in Figure 1. The calculation of redundancy bit of Hamming encoder is done by VHDL code written in Xilinx ISE 10.1 project navigator window as shown in Figure 2.

The simulate of Hamming code generation code for VHDL code by using Xilinx ISE 10.1 Simulator for source end shown in given below Figure 3, Figure 4.



(8)

Xilinx - ISE - F:VIAMMINGENCODE64VIAMMINGENCODE64, ise - [HAMMING64ENCODER.vhd]	
File Edit View Project Source Process Window Help	_)(7)(×
ife []]]]]] · · · · · · · · · · · · ·	
Sources × 1	
Sources for: Implementation 🕑 2 library IEEE;	
HAMMINGENCODE64 3 use IEEE.STD_LOGIC_1164.ALL;	
	-
L Den hamero - ve2 (HAMMING 5 use IEEE.STD_LOGIC_UNSIGNED.ALL;	
7 Uncomment the following library declaration if instantiating	
8 any Xilinx primitives in this code.	
9 library UNISIN;	
10use UNISIM.VComponents.all;	
11 12 ENTITY havence IS	
13 PORT(datain: IN BIT VECTOR(1 TO 64); d1 d2 d3	
14 hamout : OUT BIT VECTOR(1 TO 71)); d1 d2 d3 +REDUNDANCYd71	
Br Soun These es Snap: Libre 15 END hameno;	
16 ARCHITECTORE Ver2 OF hamenc 15	
Processes X 17 SIGNAL pl, p2, p4, p8, p16, p32, p64 : BIT:check bits	
Processes for: hamenc - ver2	
Add Existing Source 20generate check bits	
Create New Source 21 p1 <= datain(1) XOR datain(2) XOR datain(4) XOR datain(5) XOR datain(7) XOR datain(9) XOR datain(11) XOR datain(12) XOR datain(14)	1) 2
Y Vew Deign Summay 22 32 32 32 32 32 32 32 32 33 34 34 35 35 32 33 34 35 35 36 36 37 36 37 38 39 39 30 30 30 31 30 31 32 32 32 33 34 35 36 36 37 37 38 38 39 39 39 30 30 31 32 32 33 34 35 36 36 37 37 38 39 39 39 30 <	1.1.4.1
progroumer 23 p2 <- datain(1) AVR datain(2) AVR datain(3) AVR datain(4) AVR datain(1) AVR datain(1) AVR datain(1) AVR datain(1) AVR datain(1) AVR datain(2) AVR datain(
Understand a second sec	
Dimplement Design 26	
P General Diometric 12 VOR datain(3) XOR datain(4) XOR datain(9) XOR datain(10) XOR datain(11) XOR datain(11) XOR datain(11) XOR datain(11) XOR datain(11) XOR datain(12) XOR datain(12	16) 🗸
	>
🕰 Processes Vyhat's New in ISE Design Suite 10.1 👔 Design Summary 🔯 HAMMING64ENC0DER, vhd	
X Started : "Launching Design Summary".	~
Started : "Launching ISE Text Editor to edit HAMMING64ENCODER.vhd".	
8	~
	>
🔮 🗉 Console 🛛 Errors 🔔 Warnings 🚾 Tcl Shell 🛛 🙀 Find in Files	
CAP5 MUM (SCRE Ln 1 Cal 1	VHDL .:
🛃 start 🔄 🗹 Marosoft Word - pps 🔄 http://en.eszel.com/ 🔝 Vinx 152 - Fri //AWM 🍞 untitled - Pant 👌 AR I 👯 😌 📯 🕏 😓 🚱 🖉 🕄 🔂 🚱 🖉 1	مى 0:10

Figure 2: Hamming Encoder in VHDL Using ISE 10.1

Current Simulation Time: 1000 ns		900 ns 910 ns 920 ns 930 ns 940 ns 950 ns 960 ns 970 ns 980 ns 990 ns1000
🖽 🚮 datain[1:64]	6	64'h555555555555555
🖪 🚮 hamout[1:71]	7	.71'h25AA5555AAAAAAA55

Figure 3: Hamming Code Generation for 64 Bits in Hexadecimal Form

	900 ns 910 ns 920 ns 930 ns 940 ns 950 ns 960 ns 970 ns 980 ns 990 ns1000
6	64%01010101010101010101010101010101010101
7	71%0100101101010101010101010101010101010
	6 7

Figure 4: Hamming Code Generation for 64 Bits in Binary Form

In the same way we generated (128,8) code as 128 bit information data needs 8 redundancy bit according to equation (1). Suppose, these redundancy bits are R(1),R(2),R(4),r(8),R(16),R(32),R(64) and R(128) to calculate these redundancy bits easily done by XORing operation of the original data bit positions as shown below:

 $\begin{array}{c} R(1) = D1 \bigoplus D2 \bigoplus D4 \bigoplus D5 \bigoplus D7 \bigoplus D9 \bigoplus D11 \bigoplus D12 \bigoplus D14 \bigoplus D16 \bigoplus D18 \bigoplus D20 \bigoplus D22 \bigoplus D24 \bigoplus D26 \bigoplus D27 \bigoplus D29 \bigoplus D31 \bigoplus D33 \bigoplus D35 \bigoplus D37 \bigoplus D39 \bigoplus D41 \bigoplus D43 \bigoplus D45 \bigoplus D47 \bigoplus D49 \bigoplus D51 \bigoplus D53 \oplus D55 \bigoplus D57 \oplus D58 \oplus D60 \oplus D62 \oplus D64 \oplus D66 \oplus D68 \oplus D70 \oplus D72 \oplus D74 \oplus D76 \oplus D78 \oplus D78$

 $D80 \oplus D82 \oplus D84 \oplus D86 \oplus D88 \oplus D90 \oplus D92 \oplus D94 \oplus D96 \oplus D98 \oplus D100 \oplus D102 \oplus D104 \oplus D106 \oplus D108 \oplus D110 \oplus D112 \oplus D114 \oplus D116 \oplus D118 \oplus D120 \oplus D121 \oplus D123 \oplus D125 \oplus D127$ (9)

 $\begin{array}{l} R(2) = D1 \bigoplus D3 \bigoplus D4 \bigoplus D6 \bigoplus D7 \bigoplus D10 \bigoplus D11 \bigoplus D13 \bigoplus D14 \bigoplus D17 \bigoplus D18 \bigoplus D21 \bigoplus D22 \bigoplus D25 \bigoplus D26 \bigoplus D28 \bigoplus D29 \bigoplus D32 \bigoplus D33 \bigoplus D36 \bigoplus D37 \bigoplus D40 \bigoplus D41 \bigoplus D44 \bigoplus D45 \bigoplus D48 \bigoplus D49 \bigoplus D52 \bigoplus D53 \bigoplus D56 \bigoplus D57 \bigoplus D59 \bigoplus D60 \bigoplus D63 \bigoplus D64 \bigoplus D67 \bigoplus D68 \bigoplus D71 \bigoplus D72 \bigoplus D75 \bigoplus D76 \bigoplus D79 \bigoplus D80 \bigoplus D83 \bigoplus D84 \bigoplus D87 \bigoplus D88 \bigoplus D91 \bigoplus D92 \bigoplus D95 \bigoplus D96 \bigoplus D99 \bigoplus D100 \oplus D103 \bigoplus D104 \oplus D107 \bigoplus D108 \bigoplus D111 \oplus D112 \oplus D115 \oplus D116 \oplus D119 \oplus D120 \oplus D122 \oplus D123 \oplus D126 \oplus D127 \end{array}$

 $\begin{array}{l} \mathsf{R}(4) = \mathsf{D2} \bigoplus \mathsf{D3} \bigoplus \mathsf{D4} \bigoplus \mathsf{D8} \bigoplus \mathsf{D9} \bigoplus \mathsf{D10} \bigoplus \mathsf{D11} \bigoplus \mathsf{D15} \bigoplus \mathsf{D16} \bigoplus \mathsf{D17} \bigoplus \mathsf{D18} \bigoplus \mathsf{D23} \bigoplus \mathsf{D24} \bigoplus \mathsf{D25} \bigoplus \mathsf{D26} \bigoplus \mathsf{D30} \bigoplus \mathsf{D31} \bigoplus \mathsf{D32} \bigoplus \mathsf{D33} \bigoplus \mathsf{D38} \bigoplus \mathsf{D39} \bigoplus \mathsf{D40} \bigoplus \mathsf{D41} \bigoplus \mathsf{D46} \bigoplus \mathsf{D47} \bigoplus \mathsf{D48} \bigoplus \mathsf{D49} \bigoplus \mathsf{D54} \bigoplus \mathsf{D55} \bigoplus \mathsf{D56} \bigoplus \mathsf{D57} \bigoplus \mathsf{D61} \bigoplus \mathsf{D62} \bigoplus \mathsf{D63} \bigoplus \mathsf{D64} \oplus \mathsf{D69} \bigoplus \mathsf{D70} \bigoplus \mathsf{D71} \bigoplus \mathsf{D72} \bigoplus \mathsf{D77} \bigoplus \mathsf{D78} \oplus \mathsf{D79} \bigoplus \mathsf{D80} \bigoplus \mathsf{D85} \bigoplus \mathsf{D86} \bigoplus \mathsf{D87} \bigoplus \mathsf{D88} \bigoplus \mathsf{D93} \oplus \mathsf{D94} \bigoplus \mathsf{D95} \bigoplus \mathsf{D96} \oplus \mathsf{D101} \oplus \mathsf{D102} \oplus \mathsf{D103} \oplus \mathsf{D104} \oplus \mathsf{D109} \oplus \mathsf{D110} \oplus \mathsf{D111} \oplus \mathsf{D112} \oplus \mathsf{D117} \oplus \mathsf{D118} \oplus \mathsf{D119} \oplus \mathsf{D120} \oplus \mathsf{D124} \oplus \mathsf{D125} \oplus \mathsf{D126} \oplus \mathsf{D127} \end{array}$

 $\begin{array}{l} R(8) = D5 \oplus D6 \oplus D7 \oplus D8 \oplus D9 \oplus D10 \oplus D11 \oplus D19 \oplus D20 \oplus D21 \oplus D22 \oplus D23 \oplus D24 \oplus D25 \oplus \\ D26 \oplus D34 \oplus D35 \oplus D36 \oplus D37 \oplus D38 \oplus D39 \oplus D40 \oplus D41 \oplus D50 \oplus D51 \oplus D52 \oplus D53 \oplus D54 \oplus \\ D55 \oplus D56 \oplus D57 \oplus D65 \oplus D66 \oplus D67 \oplus D68 \oplus D69 \oplus D70 \oplus D71 \oplus D72 \oplus D81 \oplus D82 \oplus D83 \oplus D84 \oplus \\ D85 \oplus D86 \oplus D87 \oplus D88 \oplus D97 \oplus D98 \oplus D99 \oplus D100 \oplus D101 \oplus D102 \oplus D103 \oplus D104 \oplus D113 \oplus D114 \oplus \\ D115 \oplus D116 \oplus D117 \oplus D118 \oplus D119 \oplus D120 \oplus D128 \end{array}$ (12)

 $\begin{array}{l} R(16) = D12 \bigoplus D13 \bigoplus D14 \bigoplus D15 \bigoplus D16 \bigoplus D17 \bigoplus D18 \bigoplus D19 \bigoplus D20 \bigoplus D21 \bigoplus D22 \bigoplus D23 \bigoplus D24 \bigoplus D25 \oplus D26 \bigoplus D42 \bigoplus D43 \bigoplus D44 \bigoplus D45 \oplus D46 \oplus D47 \oplus D48 \oplus D49 \oplus D50 \oplus D51 \oplus D52 \oplus D53 \oplus D54 \oplus D55 \oplus D56 \oplus D57 \oplus D73 \oplus D74 \oplus D75 \oplus D76 \oplus D77 \oplus D78 \oplus D79 \oplus D80 \oplus D81 \oplus D82 \oplus D83 \oplus D84 \oplus D85 \oplus D86 \oplus D87 \oplus D88 \oplus D105 \oplus D106 \oplus D107 \oplus D108 \oplus D109 \oplus D110 \oplus D111 \oplus D112 \oplus D113 \oplus D114 \oplus D115 \oplus D116 \oplus D117 \oplus D118 \oplus D119 \oplus D120 \end{array}$

 $\begin{array}{l} R(32) = D27 \bigoplus D28 \bigoplus D29 \bigoplus D30 \bigoplus D31 \bigoplus D32 \bigoplus D33 \bigoplus D34 \bigoplus D35 \bigoplus D36 \bigoplus D37 \bigoplus D38 \bigoplus D39 \bigoplus D40 \bigoplus D41 \bigoplus D42 \bigoplus D43 \bigoplus D44 \bigoplus D45 \oplus D46 \bigoplus D47 \oplus D48 \oplus D49 \oplus D50 \bigoplus D51 \oplus D52 \oplus D53 \oplus D54 \oplus D55 \oplus D56 \oplus D57 \oplus D89 \oplus D99 \oplus D100 \oplus D101 \oplus D102 \oplus D103 \oplus D104 \oplus D105 \oplus D106 \oplus D107 \oplus D108 \oplus D109 \oplus D110 \oplus D111 \oplus D112 \oplus D113 \oplus D114 \oplus D115 \oplus D116 \oplus D117 \oplus D118 \oplus D119 \oplus D120 \\ \end{array}$

 $\begin{array}{l} R(64) = D58 \oplus D59 \oplus D60 \oplus D61 \oplus D62 \oplus D63 \oplus D64 \oplus D65 \oplus D66 \oplus D67 \oplus D68 \oplus D69 \oplus D70 \oplus D71 \oplus D72 \oplus D73 \oplus D74 \oplus D75 \oplus D76 \oplus D77 \oplus D78 \oplus D79 \oplus D80 \oplus D81 \oplus D82 \oplus D83 \oplus D84 \oplus D85 \oplus D86 \oplus D87 \oplus D88 \oplus D89 \oplus D90 \oplus D91 \oplus D92 \oplus D93 \oplus D94 \oplus D95 \oplus D96 \oplus D97 \oplus D98 \oplus D99 \oplus D100 \oplus D101 \oplus D102 \oplus D103 \oplus D104 \oplus D105 \oplus D106 \oplus D107 \oplus D108 \oplus D109 \oplus D110 \oplus D111 \oplus D112 \oplus D113 \oplus D114 \oplus D115 \oplus D116 \oplus D117 \oplus D118 \oplus D119 \oplus D120 \end{array}$

 $R(128) = D121 \oplus D122 \oplus D123 \oplus D124 \oplus D125 \oplus D126 \oplus D127 \oplus D128$ (16)

The value of redundancy bit can be calculated by XORing of different locations of information data bits, as shown in Figure 5.

Journal of Scientific and Engineering Research

D1	R1	R1							
D2 D3	D1	→ D1							
D4 D5	D2	→ D2							
D6 D7	D3 D4	→ D4	→ D3 → D4	\rightarrow D3 \rightarrow D4					
D8 D9	R8 D5	-> D5							
D10 D11	D6 D7	→ D7	→ D6 → D7		→ D6				
D12 D13	D8 D9								
D14	D10		→ <u>D10</u>	→ D10	→ D8 → D9 → D10 → D11				
D15 D16	D11 R16	→ D11	→ D11	→ D11	D11	→ R16			
D17 D18	D12 D13	► D12	→ D13			→ R16 → D12 → D13			
D19 D20	D14 D15	► D14	→ D14	- → D15		→ D14 → D15 → D16 → D17 → D19 → D20 → D21 → D22 → D23 → D24			
D21 D22	D16 D17	→ D16	► D17	→ D16		→ D16 → D17			
D23 D24	D18 D19	→ D18		→ D17 → D18	→ D19	► D18 ► D19			
D25 D26	D20 D21	→ D20	D21		→ D20	→ D20 → D21			
D27 D28	D22 D23	-> D22	→ D21 → D22	N	D20 D21 D22 D23 D24 D25	→ D22			
D29	D24	→ D24		 ▶ D23 ▶ D24 ▶ D25 	→ D24 → D25 → D26				
D30 D31	D25 D26	→ D26	→ D25 → D26	→ D25 → D26	- D25 D26	→ D25 → D26			
D32 D33	R32 D27	→ D27					→ R32 → D27		
D34 D35	D28 D29	→ D29	→ D28 → D29				→ D28 → D29		
D36 D37	D30 D31	-> D31		→ D30 → D31			→ D30 → D31 → D32		
D38 D39	D32 D33	-> D33	→ D32 → D33	→ D32 → D33			→ D32 → D33		
D39 D40 D41	D34 D35	D35					→ D34 → D35 → D36		
D42	D36		→ D36 → D37						
D43 D44	D37 D38	D37		→ D38 → D39	→ D34 → D35 → D36 → D37 → D38 → D39 → D40 → D41		→ D37 → D37 → D38 → D39 → D40		
D45 D46	D39 D40	→ D39	→ D40 → D41	-> D40			→ D39 → D40		
D47 D48	D41 D42	► D41	\rightarrow D40 \rightarrow D41	→ D41	→ D41	→ D42	→ D41		
D49 D50	D43 D44	► D43	→ D44			D43	→ D43 → D44		
D51 D52	D45 D46	→ D45	→ D44 → D45	→ D46		→ D44 → D45 → D46 → D47 → D48 → D49 → D50 → D51 → D52 → D53	- D45		
D53 D54	D47 D48	► D47		→ D47		→ D47 → D48	→ D46 → D47 → D48 → D49 → D50 → D51 → D52		
D54 D55 D56	D48 D49 D50	→ D49	→ D48 → D49	→ D48	D 50	→ D48 → D49 → D50	D48 D49 D50		
D58 D57 D58	D51	► D51			→ D50 → D51 → D52	D50	→ D50 → D51		
D59	D52 D53	→ D53	→ D52 → D53		-> D53	→ D52 → D53	DEG		
D60 D61	D54 D55	→ D55		→ D54 → D55	-> D55	→ D54 → D55	→ D54 → D54 → D55 → D56 → D57		
D62 D63	D56 D57	→ D57	→ D56 → D57	→ D56 → D57	→ D56 → D57	→ D56 → D57	→ D56 → D57		
D64 D65	R64 D58							R64	
D66 D67	D59 D60		→ D59 → D60					+ R64 + D58 + D59 + D60 + D60 + D60 + D66 + D66 + D66 + D66 + D66 + D66 + D68 + D68 + D70 + D70 + D72 + D75 + D76 + D7 +	
D68 D69	D61 D62			→ D61 → D62				 D60 D61 D62 D63 	
D70	D63	► D62	→ D63 → D64	_► D63				→ D62	
D71 D72	D64 D65	→ D64	► D64	_► D64	► D65			D64 D65	
D73 D74	D66 D67	► D66	→ D67		→ D66 → D67			D66 D67	
D75 D76	D68 D69	→ D68	\rightarrow D67 \rightarrow D68		→ D68 → D69			D68 D69	
D77 D78	D70 D71	→ D70	→ D71	→ D70 → D71	-> D70			→ D70	
D79 D80	D72 D73	→ D72	→ D71 → D72	→ D72	→ D71 → D72			→ D70 → D71 → D72 → D73 → D74 → D75 → D76 → D76 → D77 → D78	
D81 D82	D74 D75	→ D74	D75			 → D73 → D74 → D75 		→ D74	
D83 D84	D76 D77	→ D76	► D75 ► D76	677		► D76		► D75	
D85	D78	► D78		→ D77 → D78		 ▶ D77 ▶ D78 		→ 077 → 078 → 078 → 080 → 081 → 082 → 084 → 084 → 086 → 086 → 087 → 088 → 088 → 090	
D86 D87	D79 D80	→ D80	→ D79 → D80	→ D79 → D80		D79 D80		→ D79 → D80	
D88 D89	D81 D82	→ D82			D81	D81 D82		D81 D82	
D90 D91	D83 D84	→ D84	→ D83 → D84		→ D83 → D84	→ D83 → D84		→ D83 → D84	
D92 D93	D85 D86	→ D86		→ D85 → D86	 D85 D86 	→ D85 → D86		→ D85 → D86 → D87	
D94 D95	D87 D88	-> D88	→ D87 → D88	→ D87 → D88	→ D87 → D88	→ D87 → D88		D87	
D96 D97	D89 D90	-> D90					→ D89 → D90 → D91	→ D88 → D89 → D90	
D98 D99	D91 D92		→ D91 → D92						
D100 D101	D92 D93 D94			→ D93 → D94			→ D93 → D94	→ D92 → D93 → D94	
D102	D95		→ D95	-> D95			-> D95	-> D95	
D103 D104	D96 D97			→ D96	→ D97 → D98		→ D96 → D97 → D98	→ D96 → D97 → D98	
D105 D106	D98 D99	→ D98	→ D99		-> D99		-> D99	D98 D99	
D107 D108	D100 D101	► D100		→ D101	 D100 D101 D102 D103 		→ D100 → D101	 D99 D100 D101 D102 	
D109 D110	D102 D103	► D102	→ D103	→ D101 → D102 → D103	→ D102 → D103		-> D103	> D103	
D111 D112	D104 D105	→ D104	→ D104	→ D104	→ D104	→ D105	→ D104	-> D104	
D113 D114	D106 D107	► D106	→ D107			→ D106 → D107	D105 D106 D107	> D106	
D114 D115 D116	D107 D108 D109	→ D108	D107	→ D109		→ D108	D108	D108	
D117	D110	→ D110		-► D110		-• D110	-• D110	→ D109 → D110 → D111	
D118 D119	D111 D112	→ D112	→ D111 → D112	→ D111 → D112		-> D112	→ D112	 D110 D111 D112 D113 D114 D115 	
D120 D121	D113 D114	→ D114			► D113 ► D114	D113	→ D113 → D114 → D115	D113 D114	
D122 D123	D115 D116	► D116	→ D115 → D116		 D115 D116 D117 	→ D115 → D116	→ D116		
D124 D125	D117 D118	D118		→ D117 → D118	D118	→ D117 → D118	→ D117 → D118	D117 D118	
D126 D127	D119 D120	→ D120	→ D119	→ D119 → D120	→ D119 → D120	→ D119 → D120	→ D119 → D120	D119 D120	
D127 D128	R128 D121	D120			0120	0120	0120	0120	→ R128
	D121 D122 D123	D121	→ D122						→ D121 → D122 → D123
	D124			→ D124 → D125					D123 D124 D125
	D125 D126	→ D125	→ D126	→ D126					
	D127 D128	→ D127	→ D127	→ D127	→ D128				→ D127 → D128
<u>н</u>									

Figure 5: Hamming Code Generation for 128Bits



Current Simulation Time: 1000 ns		900	920		940		960		980
🗉 🚮 datain[1:128]	1			128'h00	000000000	00000111 ⁻	11111111111	111	
🗉 😽 hamout[1	1			136'h0100	000000000	00000222	22222222	22211	

Figure 6: Hamming Code Generation for 128 Bits in Hexadecimal Form

The design summary of Hamming Encoder with (64, 7) Code and (128, 8) Code is shown in Table 1 and Table 2 respectively.

Table 1: Hamming Encoder Design Status with (64, 7) Code

		HAMMINGENCO	DE64 Project	Status		
Project File:	HAMMIN	GENCODE64.ise	Current Stat	e:	Synthesized	
Module Name:	hamenc		• Erro	irs:	No Errors	
Target Device:	xc3s200-4	4ft256	• War	nings:	No Warnings	
Product Version:	ISE 10.1 ·	WebPACK	Rou	ting Results:		
Design Goal:	Balanced		• Timi	ing Constraints:		
Design Strategy:	Xilinx Def	ault (unlocked)	 Fina 	I Timing Score:		
No partition information wa		/INGENCODE64 Pa	rtition Summa	ry		E
No partition information was		/INGENCODE64 Pa	rtition Summa	ry		H
No partition information wa	s found.	/INGENCODE64 Pa				
No partition information was	s found.			ues)	ilable	
Logic Utilization	s found.	Jtilization Summary (ues)	ilable 1920	[-]
	s found.	Jtilization Summary (estimated val	ues)		[-] Utilization

Table 2: Hamming Encoder Design Status with (128, 8) Code

	HAMMINGFINAL	128 Project Status	
Project File:	hammingfinal128.ise	Current State:	Synthesized
Module Name:	hamenc	Errors:	No Errors
Target Device:	xc5vlx30-3ff676	Warnings:	No Warnings
Product Version:	ISE 9.2i	 Updated: 	الخميس 1. كانون الثاني 17:23 19:17 2015
No partition information was fou		28 Partition Summary	
No partition information was fou	nd.		
•	nd. Device Utilization Sum	mary (estimated values)	
No partition information was fou	nd.		Utilization
•	nd. Device Utilization Sum	mary (estimated values) Available	
Logic Utilization	nd. Device Utilization Sum Used	mary (estimated values) Available	09

5. Hamming Decoder with (64, 7) Code and (128, 8) Code

At the receiver side 71 bit information data is received, 64 bit encrypted information data and redundancy 7 bits. At the destination, the receiver receives 71 bit encrypted data and check for any error that may occurred. If any error is occurred, receiver find the error location and corrects it. Hamming decoder detect the error by EXORing data and corrected it by a NOT gate ⁽⁸⁾. Then the receiver removes the redundancy bit and get the original data information, if there are no error the result of even parity check was (0000000) else it detect the location of error bit as shown in Figure 7.

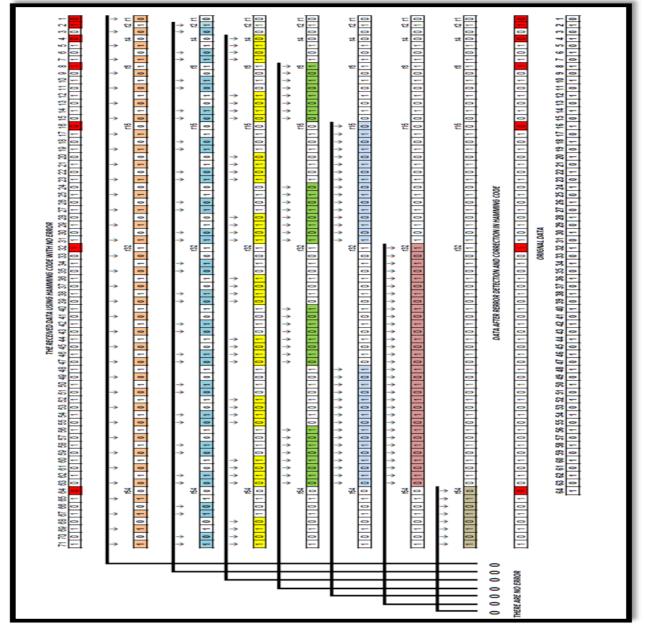


Figure 7: Hamming Code Detection Method for 64 Bits with no Error State

In the same way at the Hamming code (128, 8) receiver 136 bit information data was received, 128 bit encrypted information data and 8 bit is the redundancy which transmit by transmitter at source end as shown in Figure 8.

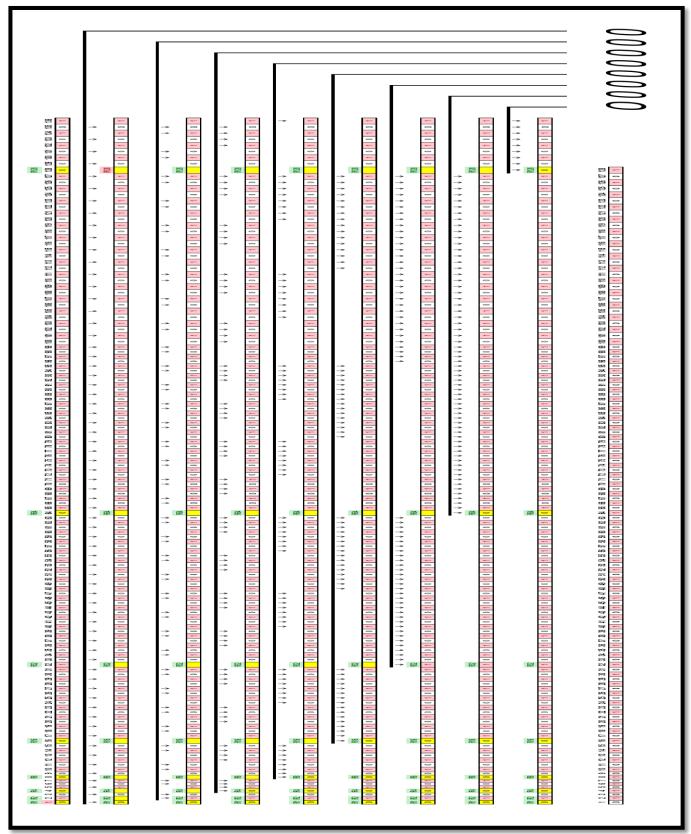


Figure 8: Hamming Code Detection Method for 128 Bits with no Error State

The detection and correction of single error bit by Hamming decoder is done by VHDL code written in Xilinx ISE 10.1 project navigator window.

So according the supposed example of (64, 7) code the received data with no error (noise less channel) will be "0100 1011 0100 1010 1010 1010 1011 0101 0101 0101 0101 0101 0101 0101 0101 1010 101" which equal in Hexadecimal "25AA5555AAAAAAA55", where 'ded means detection error' and 'ne means no error' is as shown in Figure 9 and Figure 10.

Current Simulation Time: 1000 ns		900 ns <mark>NO ERROR</mark> 920 ns 930 ns 940 ns 950 ns 960 ns 970 ns 980 ns 990 ns1000
🖬 😽 hamin[1:71]	7	71h25AA5555AAAAAAA55
🖽 🚮 dataout[1:64]	6	64'h656556555555555
oli ded	0	
one ne	1	

Figure 9: Hamming of (64,7) code with a Single Error in Hexadecimal Form (With no error state)

Current Simulation Time: 1.45431e+09 n		NO ERROR <mark>I</mark> IS 11053920 ns 11053940 ns 11053960 ns 11053980 ns 11054000
🖪 😽 hamin[1:71]	7	71/b0100101101010101010101010101010101010
🖬 🚮 dataout[1:64]	6	64%01010101010101010101010101010101010101
ded 🚺	0	
o ne	1	
G		

Figure 10: Hamming of (64,7) code with a Single Error in Binary Form (With no error state)

According to Hamming detection method take even parity check to get the address of error location is = 0000011 (the third bit at the input data) after getting the location of error bit receiver correct that error bit by replacing zero by one and one by zero. And we get actual encrypted data is transmitted by transmitter at source end.

	11 10 6 6 6 6 6 6 9 3 3 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5
0 0 0 0 0 1 1	1 1
The error is get at the BIT number 3	1 0 1

Figure 11: Hamming Code Detection Method for 64 Bits with Error State

We write VHDL code to find the error bit location, correction it and decrypt this encrypted data. Simulated results for destination end shown in Xilinx ISE 10.1 Simulation window which shows 71 bit receives encrypted data string and 64 bit actual error free information data string after correction the error, as shown in Figure 12 and Figure 13.

Current Simulation Time: 1000 ns		Detect Error Error position 900 ns 910 ns 920 ns 930 ns 940 ns 950 ns 960 ns 970 ns 980 ns 990 ns1000
🗉 🚮 hamin[1:71]	7	71h35AA5555AAAAAAAA55
🖽 🚮 dataout[1:64]	6	84'h655555555555555
oll ded	1	
o <mark>, i</mark> ne	0	

Figure 12: Hamming (64,7) Decoder for a Single Error with Error Received Data (Error at Third Bit) in Hexadecimal Form

Current Simulation Time: 1000 ns		Error position 00 hs 910 hs 920 ns 960 ns 970 ns 980 ns 990 ns1000 01 hs 910 hs 920 ns 940 ns 950 ns 960 ns 970 ns 980 ns 990 ns1000
🗉 😽 hamin[1:71]	7	71%0110101010101010101010101010101010101
🗉 🚮 dataout[1:64]	6	64%01010101010101010101010101010101010101
oll ded	1	
one 💦	0	

Figure 13: Hamming (64,7) Decoder for a Single Error with Error Received Data (Error at Third Bit) in Binary Form

In the same way Hamming Decoder with (128, 8) Code will find the error bit location, correction it as shown in Figure 14 with no error and Figure 15 with error state.

Current Simulation Time: 1000 ns			<mark>Derror</mark> 9/00	920 		940		960	
🗉 🚮 hamin[1:136]	1				136'h0	1000000000	00000002222	22222222222211	
🖽 🚮 dataout[1:128]	1	/			128	'h000000000	00000001111	111111111111	
🚮 ded	0								
🚮 ne	1								

Figure 14: Hamming Code Error Detection and Correction for a (128,8) Code (With no error state)

Current Simulation Time: 1000 ns		900 Detect Error	920	940	96	Error position	980
11110-1000113							
🗉 😽 hamin[1:136]	1		136	'h010000000000	000000222222	2222222011	
🗷 😽 dataout[1:128]	1/		1	28 [°] h000000000000	000001111111	11111111	
ded الچ	1						
🌏 🛛 ne	0						

Figure 15: Hamming Code Error Detection and Correction for a (128, 8) Code (Error at the 127th Bit) The design summary of Hamming Decoder with (64, 7) Code and (128, 8) Code is shown in Table 3 an Table 4 respectively.

Table 3: Hamming	decoder	design status	with (64	7) Code
Lable 5. Hamming	accouct	ucoign status		<u>от</u> ,	/) Couc

	HAMMINGde	coder64 Project Status			
Project File:	HAMMINGdecoder64.ise	/INGdecoder64.ise Current State: Synthesized			
Module Name:	hamdec	Errors:	No Erro	ors	
Target Device:	xc3s200-4ft256	Warnings:	<u>68 War</u>	nings	
Product Version:	ISE 10.1 - WebPACK	 Routing Re 	esults:		
Design Goal:	Balanced	Timing Cor	nstraints:		
Design Strategy:	Xilinx Default (unlocked)	 Final Timin 	g Score:		
No partition information wa	HAMMINGdecoder64 is found.	Partition Summary			H
No partition information wa	is found.				
		ary (estimated values)	Available		[-] [-] Utilization
No partition information wa	IS found. Device Utilization Summ	ary (estimated values)	Available	1920	EI
Logic Utilization Number of Slices	IS found. Device Utilization Summ	ary (estimated values)	Available	1920 3840	[-] Utilization
Logic Utilization	IS found. Device Utilization Summ	ary (estimated values) d 135	Available		[-] Utilization 7%

Table 4: Hamming decoder design status with (128, 8) Code

	HAMMINGDECOD	ER128 Project Status	
Project File:	hammingdecoder128.ise	Current State:	Synthesized
Module Name:	hamdec	Errors:	No Errors
Target Device:	xc5vlx30-3ff676	Warnings:	130 Warnings
Product Version:	ISE 9.2i	Updated:	الخميس 1. كانون الثاني 21:24:55 2015
	Device Utilization Sum	imary (estimated values)	
Logic Utilization	Used	Available	Utilization
Number of Slice LUTs	424	4 19200	0 2%
Number of fully used Bit Slices	(0 424	4 0%
Number of bonded IOBs	266	6 400	0 66%

1



Number of BUFG/BUFGCTRLs

32

3%

6. Conclusion

As a conclusion, Hamming code error detection and correction with even parity check method can be design using (64 and 128) bits data string in VHDL and can be implemented in FPGA. it speed up the communication as we can encode the total data bits as a whole and send as soon, so there are no need for data splitting, therefore more combination (more information in a single frame) of data can be transmitted easily. The complexity of circuit also reduced for regenerating actual information data from encrypted corrupt received data at destination end by using of the same method at the source end, so the original data can be correctly recovered.

References

- 1. Gomes, J., & Mishra, B. K. (2010). Double Error Correcting Long Code. *International Journal of Computer Networks and Communications*, 2(5), 58-69.
- 2. en.wikipedia.org/wiki/Hamming code.
- 3. Moon, T. K. (2005). Error correction coding. Mathematical Methods and Algorithms. Jhon Wiley and Son.
- 4. Tam, S. (2006). Single error correction and double error detection. *Xilinx Application Note*.
- 5. Gupta, B. K. (2013) Novel Hamming code for error correction and detection of higher data bits using VHDL. *International Journal of Scientific & Engineering Research*, 4(4), 272-275.
- 6. Faraj, P., Leibrich, J., & Rosenkranz, W. (2003). Coding gain of basic FEC block-codes in the presence of ASE noise. In *Transparent Optical Networks*, 2003. Proceedings of 2003 5th International Conference on (Vol. 2, pp. 80-83). IEEE.
- Singh, J. (2012, January). A comparative study of error detection and correction coding techniques. In Advanced Computing & Communication Technologies (ACCT), 2012 Second International Conference on (pp. 187-189). IEEE.
- 8. Forouzan, A. B. (2006). Data Communications & Networking (sie). Tata McGraw-Hill Education.
- 9. Gupta, B. K., & Dua, R. L. (2011). 30 Bit Hamming Code for Error Detection and Correction with Even Parity and Odd Parity Check Method by using VHDL. *International Journal of Computer Applications*, *35*(13).
- 10. Gupta, B. K., & Dua, R. L. Communication by 31 Bit Hamming Code Transceiver with Even Parity and Odd Parity Check Method by Using VHDL. *Editorial Board*, 278.
- 11. Kumar, U. K., & Umashankar, B. S. (2007, February). Improved hamming code for error detection and correction. In *Wireless Pervasive Computing*, 2007. ISWPC'07. 2nd International Symposium on. IEEE.
- 12. Mirzoyan, D. (2009). Fault-tolerant memories in FPGA based embedded systems. Skolan för informations-och kommunikationsteknik, Kungliga Tekniska högskolan.
- 13. Gupta, B. K., & Dua, R. L. (2012). Various Methodologies Used For 25 Bit Information Data String Communication Through Hamming Code. *International Journal of Applied Information Systems*. 2(2), 57-65.
- 14. Baloch, S., Arslan, T., & Stoica, A. (2005, March). Efficient Error Correcting Codes for On-Chip DRAM Applications for Space Missions. In *Aerospace Conference*, 2005 IEEE (pp. 1-9). IEEE.