Available online www.jsaer.com

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]:

$$2^r \geq m + r + 1 \quad (1)$$

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].

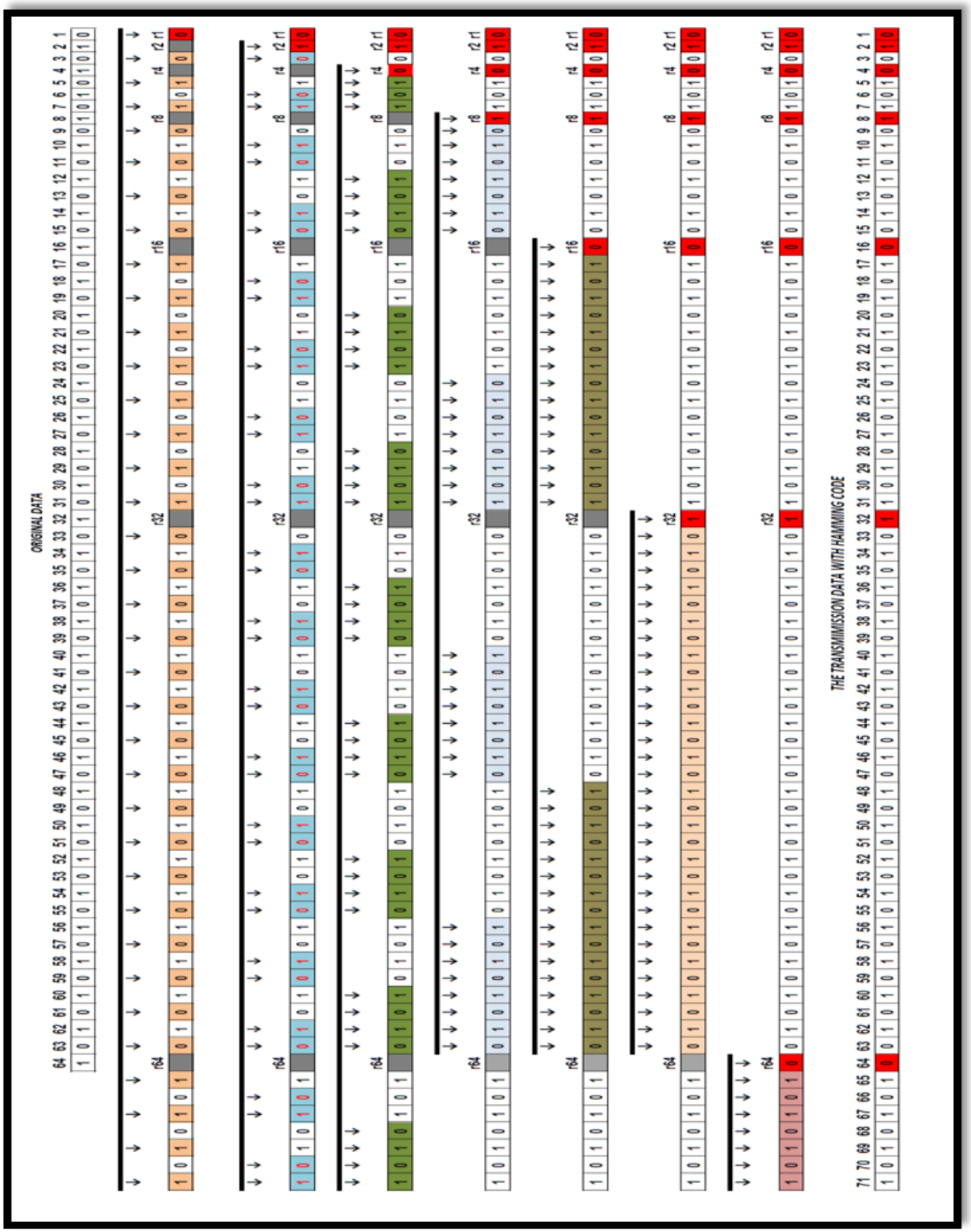

Figure 1: Hamming Code Generation for 64 Bits

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:

R(1) = D1⊕ D2 ⊕ D4⊕ D5 ⊕ D7 ⊕ D9 ⊕D11 ⊕ D12⊕ D14 ⊕ D16 ⊕ D18 ⊕ D20 ⊕ D22 ⊕ D24 ⊕ D26 ⊕ D27⊕ D29 ⊕ D31 ⊕ D33 ⊕ D35 ⊕ D37 ⊕ D39 ⊕ D41 ⊕ D43 ⊕ D45 ⊕ D47 ⊕ D49 ⊕ D51 ⊕ D53 ⊕ D55 ⊕ D57 ⊕ D58 ⊕ D60 ⊕ D62⊕ D64 (2)

R(2) = D1⊕ D3 ⊕ D4⊕ D6 ⊕ D7 ⊕ D10 ⊕D11 ⊕ D13⊕ D14 ⊕ D17 ⊕ D18 ⊕ D21 ⊕ D22 ⊕ D25 ⊕ D26 ⊕ D28⊕ D29 ⊕ D32 ⊕ D33 ⊕ D36 ⊕ D37 ⊕ D40 ⊕ D41 ⊕ D44 ⊕ D45 ⊕ D48 ⊕ D49 ⊕ D52 ⊕ D53 ⊕ D56 ⊕ D57 ⊕ D59 ⊕ D60 ⊕ D63⊕ D64. (3)

R(4) = D2⊕ D3 ⊕ D4⊕ D8 ⊕ D9 ⊕ D10 ⊕D11 ⊕ D15⊕ D16 ⊕ D17 ⊕ D18 ⊕ D23 ⊕ D24 ⊕ D25 ⊕ D26 ⊕ D30⊕ D31 ⊕ D32 ⊕ D33 ⊕ D38 ⊕ D39 ⊕ D40 ⊕ D41 ⊕ D46 ⊕ D47 ⊕ D48 ⊕ D49 ⊕ D54 ⊕ D55 ⊕ D56 ⊕ D57 ⊕ D61 ⊕ D62 ⊕ D63⊕ D64. (4)

R(8) = D5⊕ D6 ⊕ D7⊕ D8 ⊕ D9 ⊕ D10 ⊕D11 ⊕ D19⊕ D20 ⊕ D21 ⊕ D22 ⊕ D23 ⊕ D24 ⊕ D25 ⊕ D26 ⊕ D34⊕ D35 ⊕ D36 ⊕ D37 ⊕ D38 ⊕ D39 ⊕ D40 ⊕ D41 ⊕ D50 ⊕ D51 ⊕ D52 ⊕ D53 ⊕ D54 ⊕ D55 ⊕ D56 ⊕ D57. (5)

R(16) = D12⊕ D13 ⊕ D14⊕ D15 ⊕ D16 ⊕ D17 ⊕D18 ⊕ D19⊕ D20 ⊕ D21 ⊕ D22 ⊕ D23 ⊕ D24 ⊕ D25 ⊕ D26 ⊕ D42⊕ D43 ⊕ D44 ⊕ D45 ⊕ D46 ⊕ D47 ⊕ D48 ⊕ D49 ⊕ D50 ⊕ D51 ⊕ D52 ⊕ D53 ⊕ D54 ⊕ D55 ⊕ D56 ⊕ D57. (6)

R(32) = D27⊕ D28 ⊕ D29⊕ D30 ⊕ D31 ⊕ D32 ⊕D33 ⊕ D34⊕ D35 ⊕ D36 ⊕ D37 ⊕ D38 ⊕ D39 ⊕ D40 ⊕ D41 ⊕ D42⊕ D43 ⊕ D44 ⊕ D45 ⊕ D46 ⊕ D47 ⊕ D48 ⊕ D49 ⊕ D50 ⊕ D51 ⊕ D52 ⊕ D53 ⊕ D54 ⊕ D55 ⊕ D56 ⊕ D57. (7)

R(64) = D58⊕ D59 ⊕ D60⊕ D61⊕ D62 ⊕D63 ⊕ D64. (8)

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.

Suppose, we want to transmit 64 information data bit is "0101010101010101010101010101010101010101010101010101010101010101" which equal in Hexadecimal "5555555555555555".Calculation for redundancy bits, by XORing input bit, according to hamming code with even parity redundancy the transmitted data will be 71 bits "0100 1011 0101 0100 1010 1010 1010 1011 0101 0101 0101 0101 0101 0101 0101 1010 101" which equal in Hexadecimal "25AA5555AAAAAAAA55" as explained in Figure 1.

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**.**

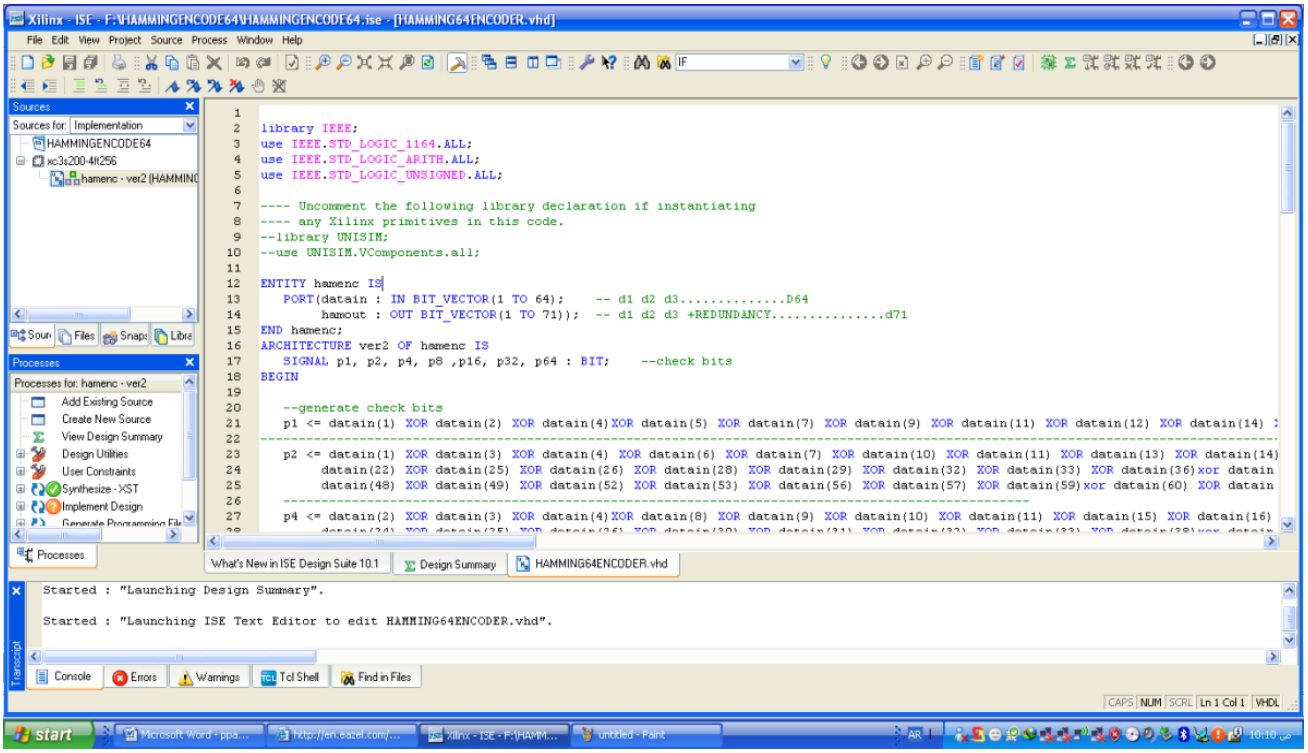

Figure 2: Hamming Encoder in VHDL Using ISE 10.1

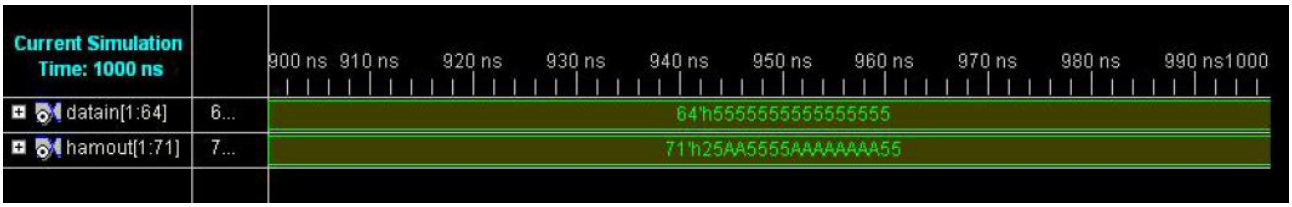

Figure 3: Hamming Code Generation for 64 Bits in Hexadecimal Form

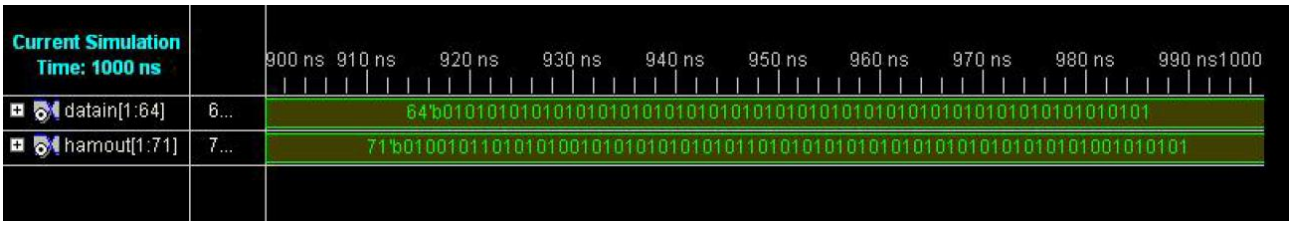

Figure 4: Hamming Code Generation for 64 Bits in Binary Form

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:

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

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

R(2) = D1⊕ D3 ⊕ D4⊕ D6 ⊕ D7 ⊕ D10 ⊕D11 ⊕ D13⊕ D14 ⊕ D17 ⊕ D18 ⊕ D21 ⊕ D22 ⊕ D25 ⊕ D26 ⊕ D28⊕ D29 ⊕ D32 ⊕ D33 ⊕ D36 ⊕ D37 ⊕ D40 ⊕ D41 ⊕ D44 ⊕ D45 ⊕ D48 ⊕ D49 ⊕ D52 ⊕ D53 ⊕ D56 ⊕ D57 ⊕ D59 ⊕ D60 ⊕ D63⊕ D64⊕ D67⊕ D68⊕ D71⊕ D72⊕ D75⊕ D76⊕ D79⊕ D80⊕ D83⊕ D84⊕ D87⊕ D88⊕ D91⊕ D92⊕ D95⊕ D96⊕ D99⊕ D100⊕ D103⊕ D104⊕ D107⊕ D108⊕ D111⊕ D112⊕ D115⊕ D116⊕ D119⊕ D120⊕ D122⊕ D123⊕ D126⊕ D127 (10)

R(4) = D2⊕ D3 ⊕ D4⊕ D8 ⊕ D9 ⊕ D10 ⊕D11 ⊕ D15⊕ D16 ⊕ D17 ⊕ D18 ⊕ D23 ⊕ D24 ⊕ D25 ⊕ D26 ⊕ D30⊕ D31 ⊕ D32 ⊕ D33 ⊕ D38 ⊕ D39 ⊕ D40 ⊕ D41 ⊕ D46 ⊕ D47 ⊕ D48 ⊕ D49 ⊕ D54 ⊕ D55 ⊕ D56 ⊕ D57 ⊕ D61 ⊕ D62 ⊕ D63⊕ D64 ⊕ D69⊕ D70⊕ D71⊕ D72⊕ D77 ⊕ D78⊕ D79⊕ D80⊕ D85⊕ D86⊕ D87⊕ D88⊕ D93 ⊕ D94⊕ D95⊕ D96⊕ D101⊕ D102⊕ D103⊕ D104⊕ D109⊕ D110⊕ D111⊕ D112⊕ D117⊕ D118⊕ D119⊕ D120⊕ D124⊕D125⊕ D126⊕ D127 (11)

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

R(16) = D12⊕ D13 ⊕ D14⊕ D15 ⊕ D16 ⊕ D17 ⊕D18 ⊕ D19⊕ D20 ⊕ D21 ⊕ D22 ⊕ D23 ⊕ D24 ⊕ D25 ⊕ D26 ⊕ D42⊕ D43 ⊕ D44 ⊕ D45 ⊕ D46 ⊕ D47 ⊕ D48 ⊕ D49 ⊕ D50 ⊕ D51 ⊕ D52 ⊕ D53 ⊕ D54 ⊕ D55 ⊕ D56 ⊕ D57⊕ D73 ⊕ D74⊕ D75⊕ D76⊕ D77⊕ D78⊕ D79⊕ D80⊕ D81⊕ D82⊕ D83⊕ D84⊕ D85⊕ D86⊕ D87⊕ D88⊕ D105⊕ D106⊕ D107⊕ D108⊕ D109⊕ D110⊕ D111⊕ D112⊕ D113⊕ D114⊕ D115⊕ D116⊕ D117⊕ D118⊕ D119⊕ D120 (13)

R(32) = D27⊕ D28 ⊕ D29⊕ D30 ⊕ D31 ⊕ D32 ⊕D33 ⊕ D34⊕ D35 ⊕ D36 ⊕ D37 ⊕ D38 ⊕ D39 ⊕ D40 ⊕ D41 ⊕ D42⊕ D43 ⊕ D44 ⊕ D45 ⊕ D46 ⊕ D47 ⊕ D48 ⊕ D49 ⊕ D50 ⊕ D51 ⊕ D52 ⊕ D53 ⊕ D54 ⊕ D55 ⊕ D56 ⊕ D57⊕ D89⊕ D99⊕D100⊕ D101⊕ D102⊕ D103⊕ D104⊕ D105⊕ D106 ⊕ D107⊕ D108⊕ D109⊕ D110⊕ D111⊕ D112⊕ D113⊕ D114⊕ D115⊕ D116⊕ D117⊕ D118⊕ D119⊕ D120 (14)

R(64) = D58⊕ D59 ⊕ D60⊕ D61⊕ D62 ⊕D63 ⊕ D64 ⊕ D65 ⊕ D66⊕D67⊕ D68⊕ D69⊕ D70⊕ D71⊕ D72⊕ D73⊕ D74⊕D75⊕ D76⊕ D77⊕ D78⊕ D79⊕ D80⊕ D81 ⊕ D82⊕D83⊕ D84⊕ D85⊕ D86⊕ D87⊕ D88⊕ D89⊕ D90⊕D91⊕ D92⊕ D93⊕ D94⊕ D95⊕ D96⊕ D97⊕ D98⊕D99⊕ D100⊕ D101⊕ D102⊕ D103⊕ D104⊕ D105⊕ D106⊕ D107⊕ D108⊕ D109⊕ D110⊕ D111⊕ D112⊕ D113 ⊕ D114⊕ D115⊕ D116⊕ D117⊕ D118⊕ D119⊕ D120 (15)

R(128) = D121⊕ D122⊕ D123⊕ D124⊕ D125⊕ D126⊕ D127⊕ D128 (16)

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

When the sender is transmit 128 information data bit is which equal in Hexadecimal "00000000000000001111111111111111".Calculation for redundancy bits, by XORing input bit ,according to hamming code with even parity redundancy the transmitted data will be 136 bits which equal in Hexadecimal "01000000000000000022222222222211" as explained in Figure 6.

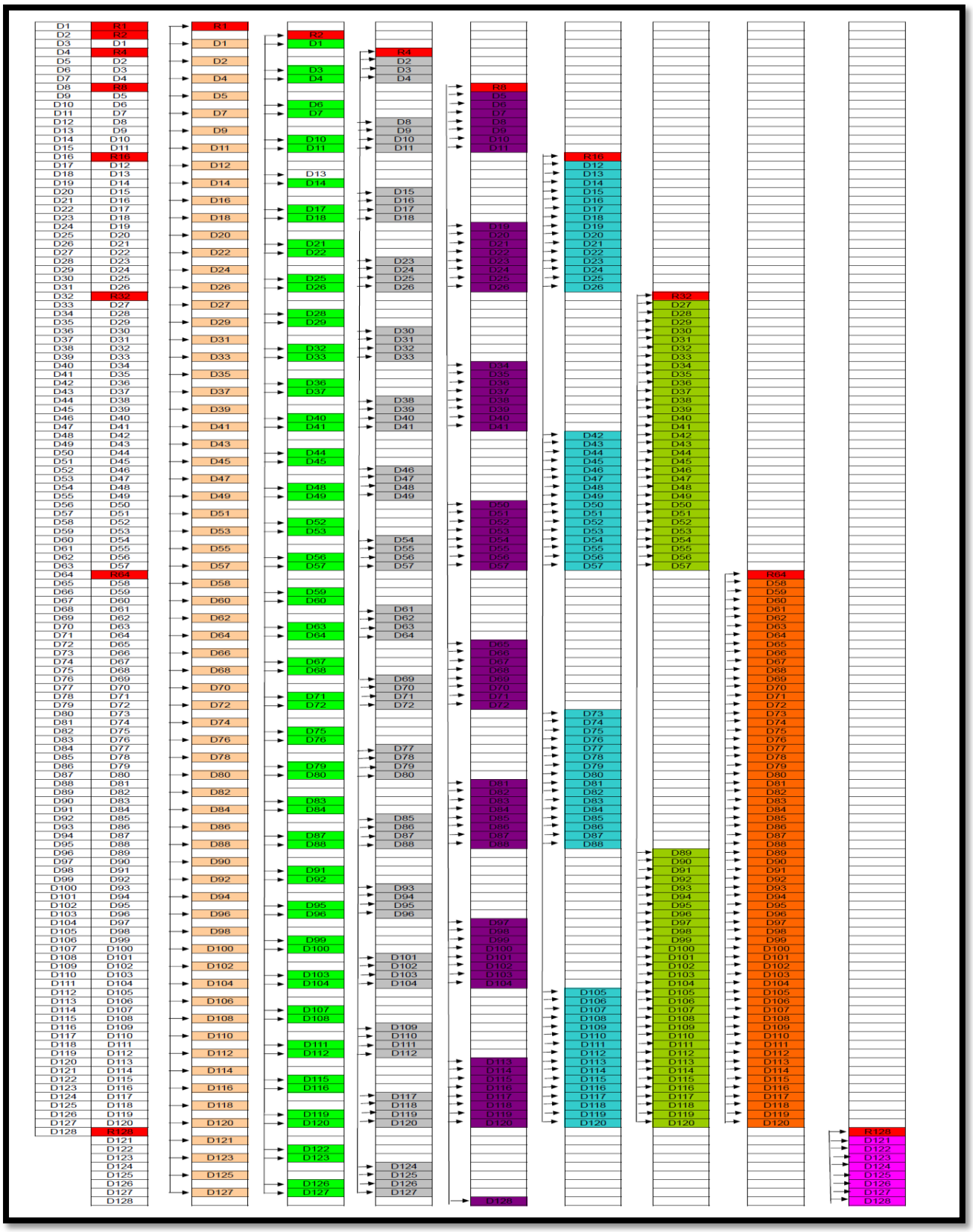

Figure 5: Hamming Code Generation for 128Bits

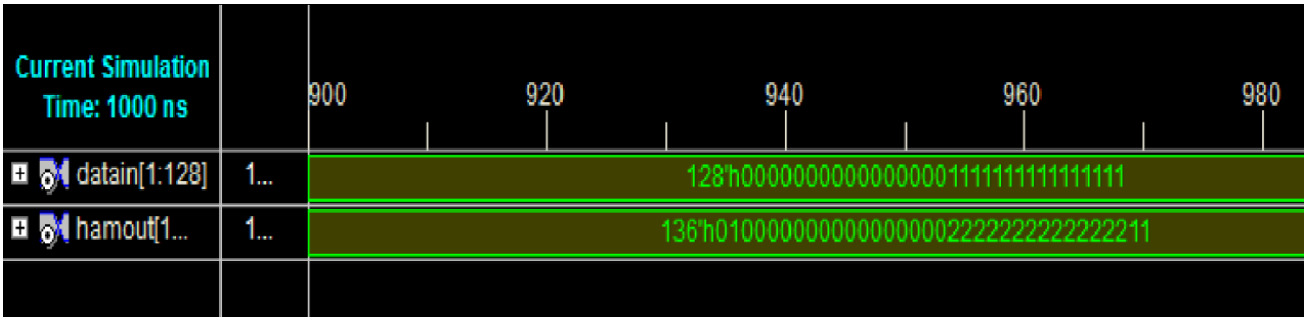

Figure 6: Hamming Code Generation for 128 Bits in Hexadecimal Form

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

HAMMINGENCODE64 Project Status			
Project File:	HAMMINGENCODE64.ise	Current State:	Synthesized
Module Name:	hamenc	• Errors:	No Errors
Target Device:	xc3s200-4ft256	• Warnings:	No Warnings
Product Version:	ISE 10.1 - WebPACK	• Routing Results:	
Design Goal:	Balanced	• Timing Constraints:	
Design Strategy:	Xilinx Default (unlocked)	• Final Timing Score:	

HAMMINGENCODE64 Partition Summary	[-]
No partition information was found.	

Device Utilization Summary (estimated values)			[-]
Logic Utilization	Used	Available	Utilization
Number of Slices	28	1920	1%
Number of 4 input LUTs	50	3840	1%
Number of bonded IOBs	135	173	78%

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

HAMMINGFINAL128 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 2015 19:17:23

HAMMINGFINAL128 Partition Summary
No partition information was found.

Device Utilization Summary (estimated values)			
Logic Utilization	Used	Available	Utilization
Number of Slice LUTs	73	19200	0%
Number of fully used Bit Slices	0	73	0%
Number of bonded IOBs	264	400	66%

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.

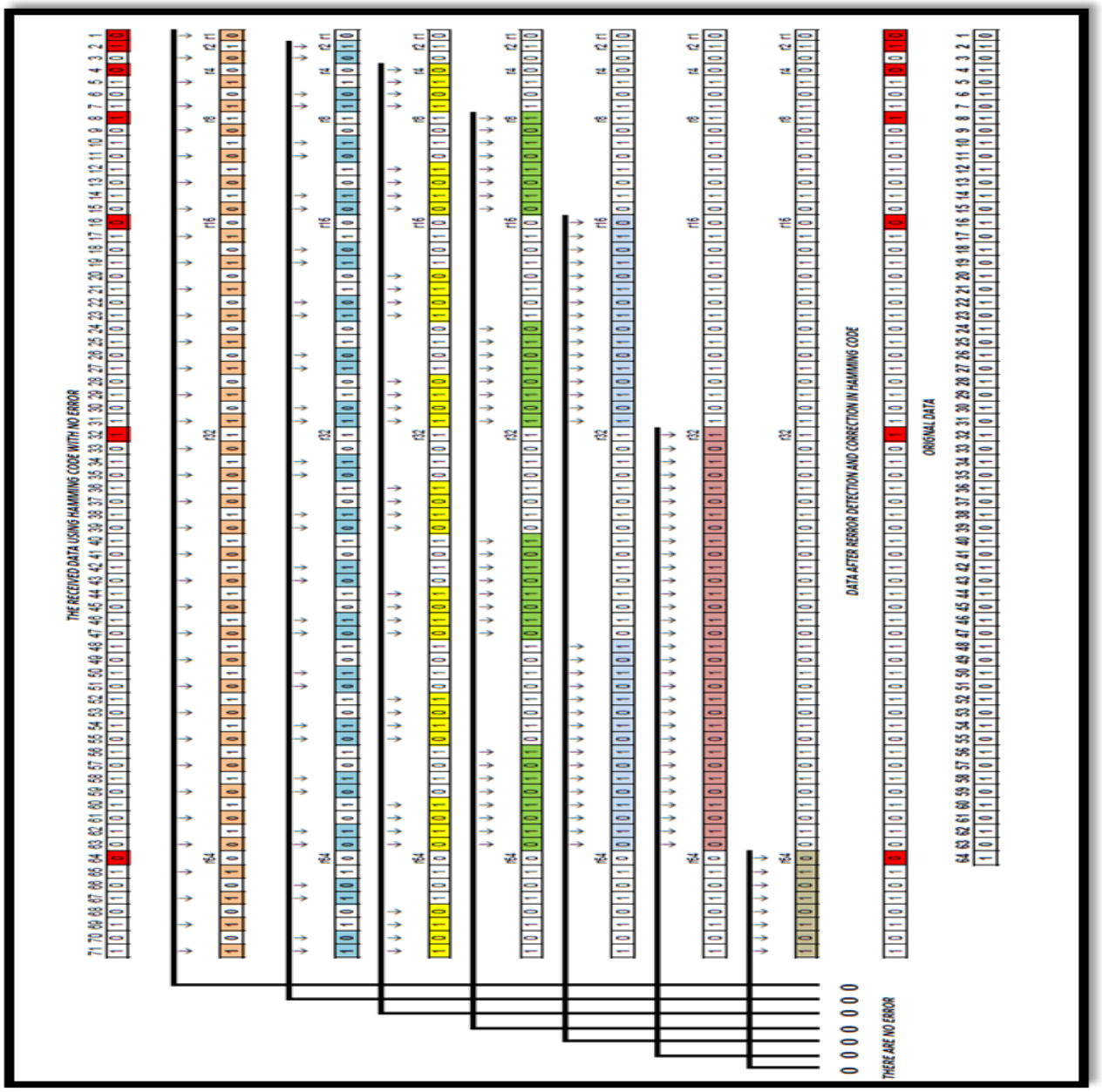

Figure 7: Hamming Code Detection Method for 64 Bits with no Error State

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.

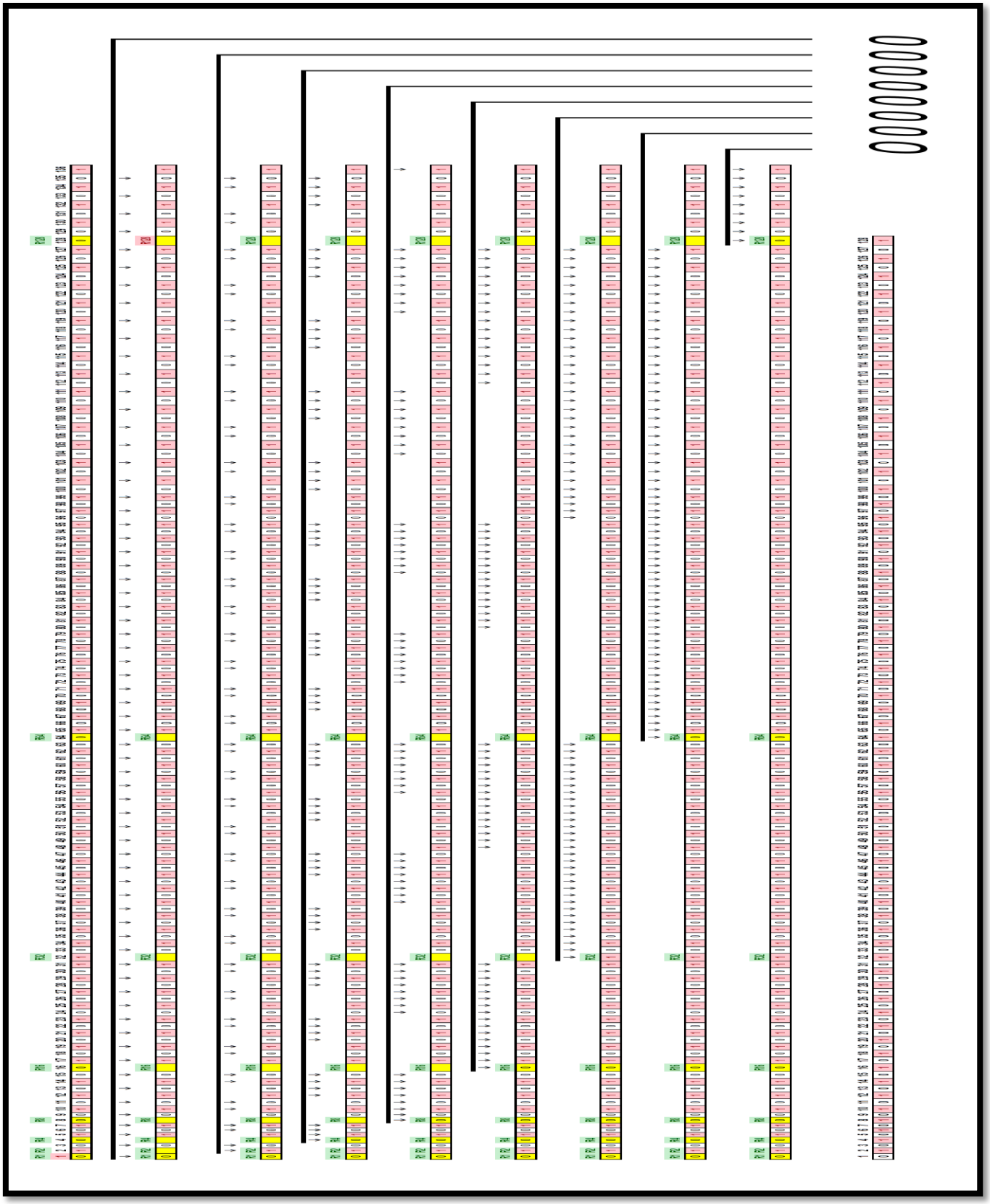

Figure 8: Hamming Code Detection Method for 128 Bits with no Error State

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 0101 0100 1010 1010 1010 1011 0101 0101 0101 0101 0101 0101 0101 0101 1010 101" which equal in Hexadecimal "25AA5555AAAAAAAA55", where 'ded means detection error' and 'ne means no error' is as shown in Figure 9 and Figure 10.

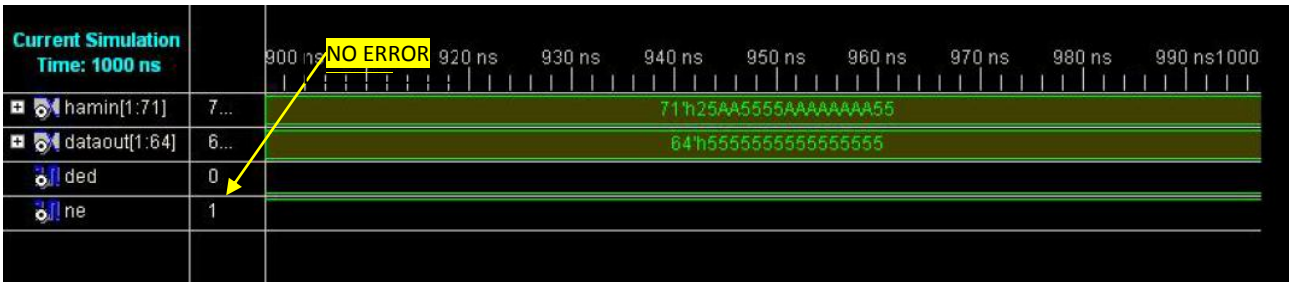

Figure 9: Hamming of (64,7) code with a Single Error in Hexadecimal Form (With no error state)

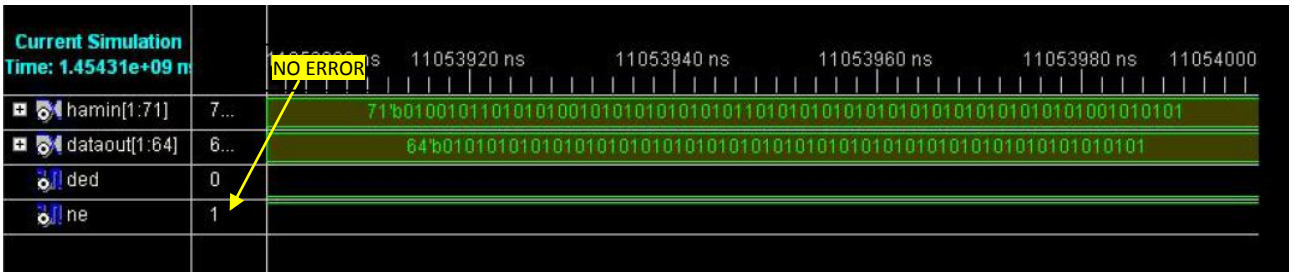

Figure 10: Hamming of (64,7) code with a Single Error in Binary Form (With no error state)

Suppose, transmitter of source end transmit data is "0100 1011 0101 0100 1010 1010 1010 1011 0101 0101 0101 0101 0101 0101 0101 0101 1010 101" which equal in Hexadecimal "25AA5555AAAAAAAA55" and at destination receiver received error data is "0110 1011 0101 0100 1010 1010 1010 1011 0101 0101 0101 0101 0101 0101 0101 0101 1010 101" which equal in Hexadecimal "35AA5555AAAAAAAA55", Hamming decoder at first detect the error location by even parity checking method and corrected it as shown in Figure 11.

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.

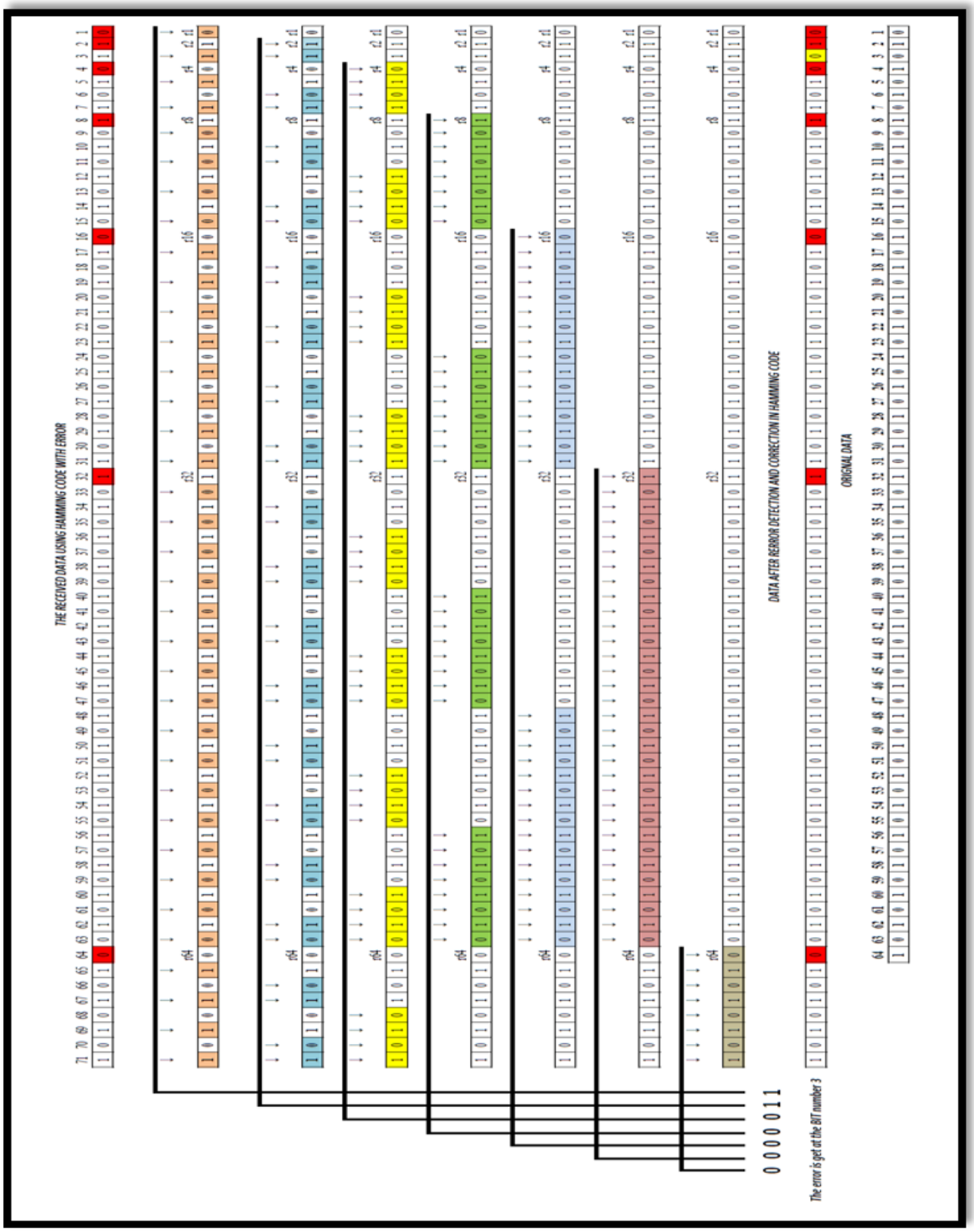

Figure 11: Hamming Code Detection Method for 64 Bits with Error State

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.

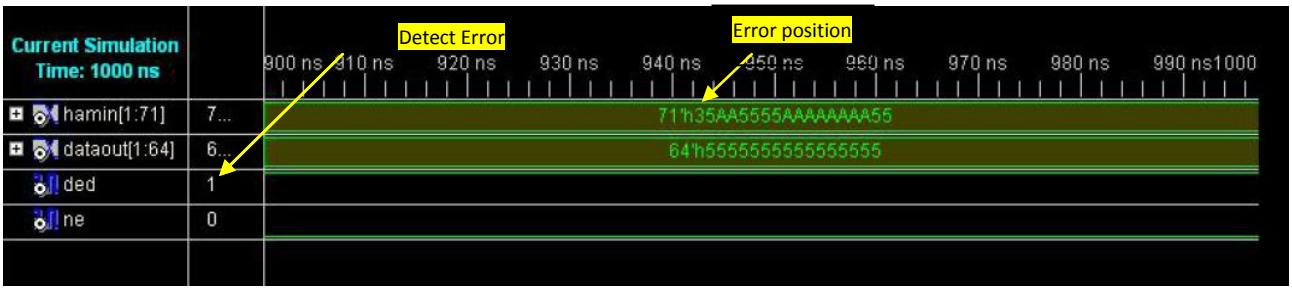

Figure 12: Hamming (64,7) Decoder for a Single Error with Error Received Data (Error at Third Bit) in Hexadecimal Form

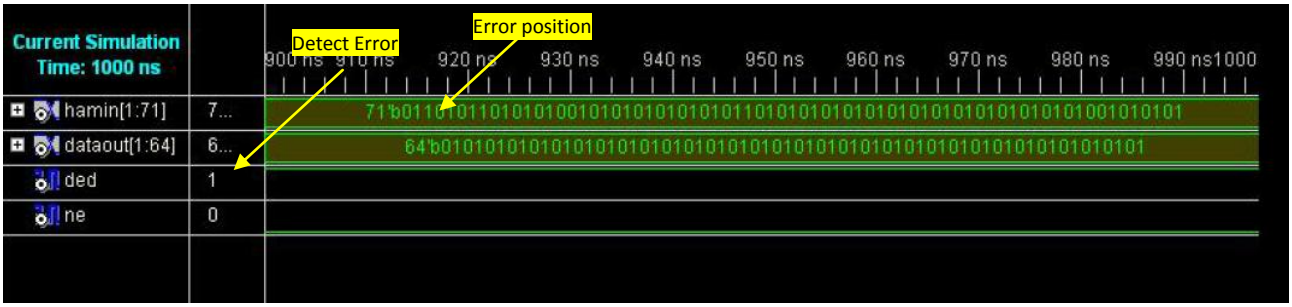

Figure 13: Hamming (64,7) Decoder for a Single Error with Error Received Data (Error at Third Bit) in Binary Form

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.

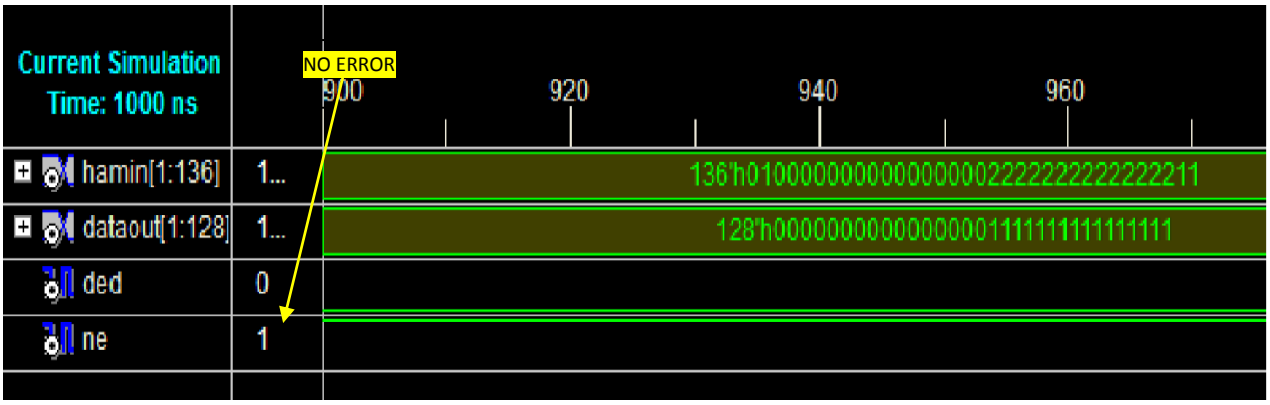

Figure 14: Hamming Code Error Detection and Correction for a (128,8) Code (With no error state)

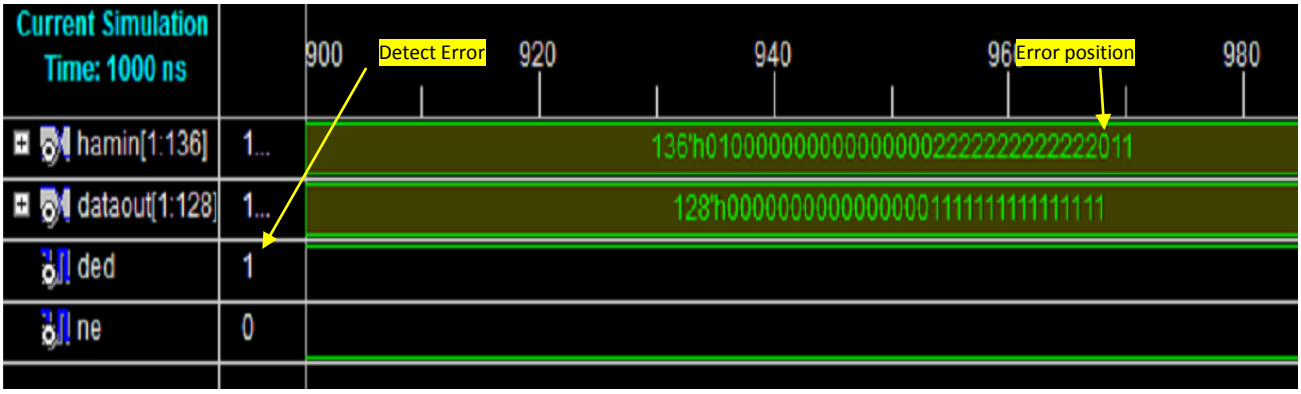

Figure 15: Hamming Code Error Detection and Correction for a (128, 8) Code (Error at the 127th Bit)

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

HAMMINGdecoder64 Project Status			
Project File:	HAMMINGdecoder64.ise	Current State:	Synthesized
Module Name:	hamdec	• Errors:	No Errors
Target Device:	xc3s200-4ft256	• Warnings:	68 Warnings
Product Version:	ISE 10.1 - WebPACK	• Routing Results:	
Design Goal:	Balanced	• Timing Constraints:	
Design Strategy:	Xilinx Default (unlocked)	• Final Timing Score:	

HAMMINGdecoder64 Partition Summary	[-]
No partition information was found.	

Device Utilization Summary (estimated values)			[-]
Logic Utilization	Used	Available	Utilization
Number of Slices	135	1920	7%
Number of 4 input LUTs	245	3840	6%
Number of bonded IOBs	137	173	79%
Number of GCLKs	1	8	12%

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

HAMMINGDECODER128 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 2015 21:24:55

HAMMINGDECODER128 Partition Summary
No partition information was found.

Device Utilization Summary (estimated values)			
Logic Utilization	Used	Available	Utilization
Number of Slice LUTs	424	19200	2%
Number of fully used Bit Slices	0	424	0%
Number of bonded IOBs	266	400	66%
Number of BUFG/BUFGCTRLs	1	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.
7. 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.