Vol. 08, No. 03, pp. 22-37, September 2015

DESIGN OF HAMMING CODE FOR 64 BIT SINGLE ERROR DETECTION AND CORRECTION USING VHDL

Adham Hadi Saleh

Assistant Lecturer, Department of Electronic engineering, College of Engineering, University of Diyala adham.hadi@yahoo.com (Received: 16/2/2014; Accepted: 21/9/2014)

ABSTRACT: - Hamming code is an efficient error detection and correction technique which can be used to detect single and burst errors, and correct errors. In communication system information data transferred from source to destination by channel, which may be corrupted due to a noise. So to find original information we use Hamming code.

In this paper, we have described how we can generate 7 redundancy bit for 64 bit information data. These redundancy bits are to be interspersed at the bit positions (n = 1, 2, 4, 8, 16, 32 and 64) of the original data bits, so to transmit 64 bit information data we need 7 redundancy bit generated by even parity check method to make 71 bit data string. At the destination receiver point, we receive 71 bit data, this receives data may be corrupted due to noise. In Hamming technique the receiver will decided 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. 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

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 combination 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, DRAM), mobile communication (cellular telephones, wireless, 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

23

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).

Suppose, we want to transmit 64 information data bit is redundancy bits using even parity method. After generating redundancy bits, add these bits to 64 bit information data for making 71 bit data string for transmission at source end. How we can generate 7 redundancy bits for 64 bit information data for making 71 bit data string for transmission at source end by using even parity method will be discussed in details at communication with even parity method. At destination receiver receives 71 bit data string from channel and check it, is it corrupted or not? If it is corrupted then the receiver find the error location according to parity check method correct it.

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 singlebit error. By contrast, the simple parity code cannot correct errors, and can only detect an odd

24

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):

$$2^{r} \ge m + r + 1 \tag{1}$$

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. 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, 13, 13) 14)

3.1 Hamming Encoder

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:

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

25

DESIGN OF HAMMING CODE FOR 64 BIT SINGLE ERROR DETECTION AND CORRECTION USING VHDL $R(2) = D1 \oplus D3 \oplus D4 \oplus D6 \oplus D7 \oplus D10 \oplus D11 \oplus D13 \oplus D14 \oplus D17 \oplus D18 \oplus D21 \oplus D22 \oplus D25 \oplus$ $D26 \oplus D28 \oplus D29 \oplus D32 \oplus D33 \oplus D36 \oplus D37 \oplus D40 \oplus D41 \oplus D44 \oplus D45 \oplus D48 \oplus D49 \oplus D52 \oplus$ $D53 \oplus D56 \oplus D57 \oplus D59 \oplus D60 \oplus D63 \oplus D64.$ (3)

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

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

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

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

 $R(64) = D58 \oplus D59 \oplus D60 \oplus D61 \oplus D62 \oplus D63 \oplus D64.$ (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 input bit, according to hamming code with even parity redundancy the transmitted data will 0101 0101 1010 101" which equal in Hexadecimal "25AA5555AAAAAAA55" as explained in Figure (1).

The simulation of Hamming code generation code for VHDL code for source end as shown in Figure (3) and Figure(4). The schematic circuit diagram is shown in Figure (5), and Hamming encoder design status is shown in Table (1).

3.2 Hamming Decoder

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 (6).

The detection and correction of a single error bit by Hamming decoder is done by VHDL code written in Xilinx ISE 10.1 project navigator window as shown in Figure (7).

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, the receiver correct, that error bit by replacing zero by one and one by zero. To produce the actual transmitted data.

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 errors, as shown in Figure (11) and Figure (12). Where Figure (13) show schematic circuit diagram of Hamming decoder. The design status of Hamming decoder is shown in Table (2).

4. CONCLUSION

As a conclusion, Hamming code error detection and correction with even parity check method can be design using 64 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.

5. REFERENCES

- Joanne Gomes and B. K. Mishra "DOUBLE ERROR CORRECTING LONG CODE" International Journal of Computer Networks & Communications (IJCNC) Vol.2, No.5, September 2010.
- 2) en.wikipedia.org/wiki/Hamming code.
- 3) Moon, Todd K. "Error Correction Coding", New Jersey, John Wiley & Sons, (2005).
- Simon Tam "Single Error Correction and Double Error Detection" XILINIX, XAPP645 (v2.2) August 9, 2006.
- 5) Brajesh Kumar Gupta, M. Tech Scholar and Associate prof. Rashmi Sinha " Novel Hamming code for error correction and detection of higher data bits using VHDL" International Journal of Scientific & Engineering Research, Volume 4, Issue 4, April, 2013.
- 6) Patrick Faraj, Jochen Leibrich and Werner Rosenkranz "Coding Gain of Basic FEC Block-Codes in the Presence of ASE Noise" University of Kiel, Chair for Communication, Kaiserstr. 2, 24143 Kiel, Germany, IEEE, 2003.
- Jatinder Singh and Jaget Singh "A Comparative study of Error Detection and Correction Coding Techniques" Second International Conference on Advanced Computing & Communication Technologies, IEEE, 2012.
- Behrouz A. Forouzan, "Data communication and networking" 3nd edition McGraw Hill publication.
- 9) Brajesh Kumar Gupta, M. Tech Scholar and Associate prof. Rashmi Sinha "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 (0975 – 8887) Volume 35– No.13, December 2011.
- 10) Mr. Brajesh Kumar Gupta and Prof. Rajeshwar Lal Dua "COMMUNICATION BY 31 BIT HAMMING CODE TRANSCEIVER WITH EVEN PARITY AND ODD PARITY CHECK METHOD BY USING VHDL" International Journal Of Computational Engineering Research / Vol. 2 | Issue No.2 |278-288, Mar-Apr 2012

- 11) U. K. Kumar, and B. S. Umashankar "Improved Hamming Code for Error Detection and Correction" IEEE, 2007.
- 12) Davit Mirzoyan "Fault-Tolerant Memories in FPGA based Embedded Systems" M.SC thesis, Dept. of Information and Communication Technology, Royal Institute of Technology (KTH), Sweden, Stockholm, June 2009.
- 13) Brajesh Kumar Gupta Rajeshwar Lal Dua "Various Methodologies used for 25 Bit Information Data String Communication through Hamming Code" International Journal of Applied Information Systems (IJAIS) – ISSN : 2249-0868 Foundation of Computer Science FCS, New York, USA Volume 2– No.2, May 2012.
- 14) S. Balochi, T. Arslani, and A. Stoicaiii "Efficient Error Correcting Codes for On-Chip DRAM Applications for Space Missions" IEEE, 2005.

		HAMMINGENCO	DE64 Project	Status		
Project File:	HAMN	IINGENCODE64.ise	Current Stat	e:	Synthesized	
Module Name:	hamer	nc	• Erro	rs:	No Errors	
Target Device:	xc3s2	00-4ft256	• War	nings:	No Warnings	
Product Version:	ISE 10).1 - WebPACK	Rou	ting Results:		
Design Goal:	Balan	ced	• Timi	ing Constraints:		
Design Strategy:	Xilinx	Default (unlocked)	 Final 	I Timing Score:		
	HA	AMMINGENCODE64 Pa	rtition Summa	ry		E
No partition information was		AMMINGENCODE64 Pa	rtition Summa	ry		E
No partition information was	s found.	AMMINGENCODE64 Pa			•	E
No partition information was	s found.			ues)	ilable	
Logic Utilization	s found.	ce Utilization Summary (ues)	ilable 1920	Ð
	s found.	ce Utilization Summary (estimated val	ues)		[-] Utilization

Table (1): Hamming encoder design status.

Table (2): Hamming decoder design status.

	HAMMINGdec	oder64 Project S	Status		
Project File:	HAMMINGdecoder64.ise	Current Stat	e:	Synthesized	
Module Name:	hamdec	• Erro	irs:	No Errors	
Target Device:	xc3s200-4ft256	• War	nings:	68 Warnings	
Product Version:	ISE 10.1 - WebPACK	Rou	ting Results:		
Design Goal:	Balanced	• Timi	ing Constraints:		
Design Strategy:	Xilinx Default (unlocked)	• Fina	I Timing Score:		
	Device Utilization Summa	ry (estimated val	ues)		[-]
Logic Utilization	Device Utilization Summa Used		,	ilable	[-] Utilization
Logic Utilization Number of Slices			,	ilable 1920	
Number of Slices			,		Utilization
~		135	,	1920	Utilization 7%

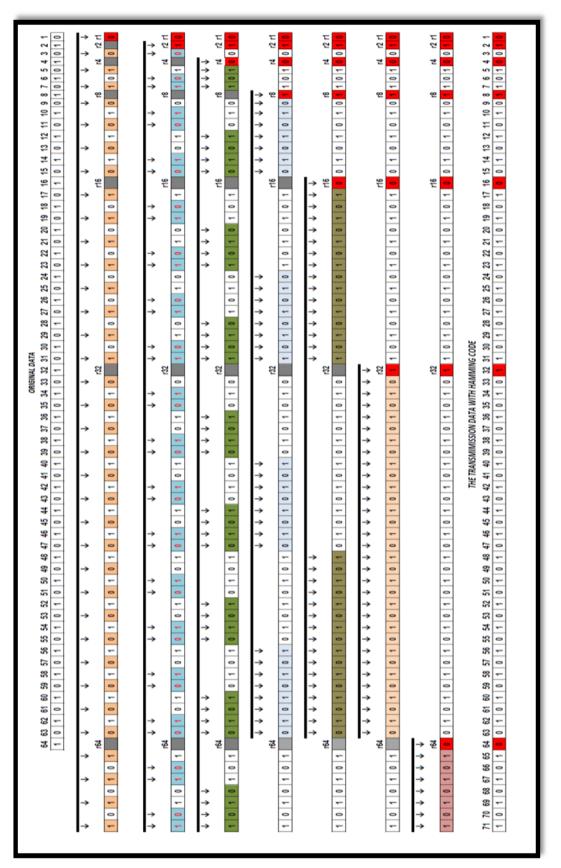


Figure (1): Hamming Code Generation for 64 Bits.

Xilinx - ISE - F-VIAMMINGENCODE64VIAMMINGENCODE64.ise - [HAMMING64ENCODER.vhd]	זר
	161)
	(a) (-
	_
	^
Sources for: Implementation 2 Library IEEE;	
□ HAMMINGENCODE64 3 use IEEE.STD_LOGIC_1164.ALL; □ #x3:22044256 4 use IEEE.STD_LOGIC_ARITH.ALL;	
Stocoverze HAMMIN 5 use IEEE.STD_LOGIC_UNSIGNED.ALL;	
Construct And Longiand	
7 Uncomment the following library declaration if instantiating	
8 any Xilinx primitives in this code.	
9library UNISIM; 10use UNISIM.VComponents.all;	
12 ENTITY hamenc IS	
13 PORT(datain: IN BIT_VECTOR(1 TO 64); d1 d2 d3	
A hamout : OUT BIT_VECTOR(1 TO 71)); d1 d2 d3 +REDUNDANCYd71	
mg Souri Files Bis Snape Libre 15 END namenc; 16 ARCHITECTURE ver2 OF hamenc IS	
Processes X 17 SIGNAL p1, p2, p4, p8, p16, p32, p64 : BIT;check bits	
Processes for hameno - ver2 A 18 BEGIN	
AddExistingSource 19 20generate check bits	
generate 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)	
E View Design Surmay	1
# 3 Design Utilities 23 p2 <= datain(1) XOR datain(3) XOR datain(4) XOR datain(6) XOR datain(7) XOR datain(10) XOR datain(11) XOR datain(13) XOR datain(14)	4)
# 24 datain(22) XOR datain(25) XOR datain(26) XOR datain(28) XOR datain(29) XOR datain(32) XOR datain(33) XOR datain(36) XO	
COSynthesize -XST ZS datain(48) XOR datain(49) XOR datain(52) XOR datain(53) XOR datain(56) XOR datain(57) XOR datain(59) xor datain(60) XOR datain	n
W (Wimplement Design 27 v4 (= datain(2) VOD datain(2) VOD datain(4) VOD datain(9) VOD datain(0) VOD datain(10) VOD datain(11) VOD datain(15) VOD datain(15)	1 -
Annual Programming Fair	, v
C C	2
🖳 rucesses 🛛 What's New in ISE Design Suite 10.1 🔹 Design Summary 🔹 HAMMING64ENCODER. vhd	
X Started : "Launching Design Summary".	^
Started : "Launching ISE Text Editor to edit HAMMING64ENCODER.vhd".	
	V
	1
🚊 📋 Console 🛛 🙆 Errors 🚺 Warnings 🛛 🔂 Tcl Shell 🛛 🙀 Find in Files	
CAPS NUM SCRU LN 1 COL1 W	IDL ,
🐉 Start 🔰 🖄 Moosoft Word - ppa 👔 http://en.eazel.com/ 📴 wink - ise - Fil/HAMM 🎽 untited - Part 👌 AR 🔆 😨 🕀 🕸 👧 🕸 🗐 🖉 10:	
🖞 Start 🔰 🖞 Wooselt Word - ppa 👌 http://en.eacek.com/ 📴 xinx - 152 - Fil (MAM 👌 untited - Part 🔰 AR I 🙀 🕻 🖯 😣 🔍 🖓 🖉 🙂 10:	مي 0.

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*h555555555555555555555555555555555555
🗄 😽 hamout(1:71)	Ĩ	71 h25A45555AAAAAAAA55

Figure (3) Hamming Code Generation for 64 Bits in Hexadecimal Form

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%01010101010101010101010101010101010101
🗄 😽 hamout(1:71)	7	71%0100101101010101010101010101010101010

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

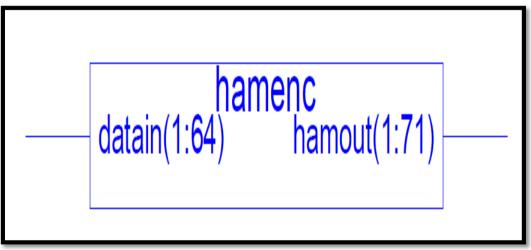


Figure (5): Schematic Circuit Diagram of Hamming Encode.

	71 70 60 68 67 66 64 65 62 61 60 56 56 55 56 55 56 56 59 59 48 47 49 54 44 32 24 10 38 35 36 35 34 33 22 31 30 29 26 27 26 25 24 23 22 120 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0
	レンシン シンシン シンシン シンシン シンシン シンシン シンシン シンシン
	レレレレレレレレレレ R4 10110110100101010101010101010101010101
	レンレンレンレンレンレンレンレンレンレンレン ext 101101101101101101010101010101010101101
	↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓ K4 101101101101101101101101101101101101101
0 0 0 0 0 0 0 0 1 THERE ARE NO ERROR	DATA AFTER RESROR DETECTION AND CORRECTION IN HAMMING CODE 1 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 0
	ORIGNAL DATA 64 65 62 61 60 39 58 75 55 54 53 52 51 50 49 48 47 46 45 44 45 42 41 40 39 38 37 38 35 43 32 31 30 28 37 28 25 22 23 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1

Figure (6): Hamming Code Detection Method for 64 Bits with no Error State

			
		AMMINGdecoder64V4AMMINGdecoder64.ise - [yt.vhd*]	
🔉 File Edit View Project Source Pr			(_)(ð) (×)
8 🗅 🤌 🗟 🕼 🕼 🖓 🕼	X 🔊	@ 🖉 🕫 🖉 🗰 🖉 🔊 🕾 🖪 💷 🗁 🖗 🕺 🗛 😹 📧 🔛 😒 🗑 😰 🖉 🖗 🕱 🕱 🛱 🖉	
ie e E 2 E 2 4 %	**	色 效	
Sources ×	2	library IEEE,	
Sources for: Implementation	3	use IEEE.STD LOGIC 1164 ALL;	
HAMMINGdecoder64	4	use IEEE.STD LOGIC ARITH.ALL;	
🖃 🛄 xc3s200-4ft256	5	use IEEE STD_LOGIC_UNSIGNED ALL;	
hamdec - ver1 (vt.vhd)	6		
	7		
	8		
	9 10	ENTITY handed IS	
	10	PORT(hamin : IN BIT_VECTOR(1 TO 71); HAMMING DECODER VHDL CODE dataout : OUT BIT_VECTOR(1 TO 64);	
	12	datable : OUT STD LOGIC);diagnostic outputs	
	13	END handec:	
	14	ARCHITECTURE ver1 OF handec IS	
	15	BEGIN	
🕫 Souri 👔 Files 🚕 Snape 👔 Libra	16	PROCESS (hamin)	
	17	VARIABLE syndrome : BIT_VECTOR(7 DOWNTO 1);	
Processes ×	18	BEGIN	
Processes for: hamdec - ver1	19	generate syndrome bits	
Add Existing Source	20	<pre>syndrome(1) := hamin(1) XOR hamin(3) XOR hamin(5) XOR hamin(7) XOR hamin(9) XOR hamin(11) XOR hamin(13) XOR hamin(15) XOR syndrome(2) := hamin(2) XOR hamin(3) XOR hamin(6) XOR hamin(7) XOR hamin(10) XOR hamin(14) XOR hamin(14) XOR hamin(15) XOR</pre>	
Create New Source	21	syndrome(z) := hamin(z) XOR hamin(s) XOR hamin(s) XOR hamin(r) XOR hamin(z) XOR ham	
🕞 🗵 🛛 🗐 📃 📃	23	syndrome(5) - namin(3) XOR hamin(5) XOR hamin(10) XOR hamin(11) XOR hamin(12) XOR hamin(13) XOR hamin(13) XOR hamin(15) XOR hami	
🖶 😼 Design Utilities	24	syndrome(5) := hamin(16) XOR hamin(17) XOR hamin(18) XOR hamin(19) XOR hamin(20) XOR hamin(21) XOR hamin(22) XOR hamin(23)	
🕀 🐲 User Constraints	2.5	syndrome(6) := hamin(32) XOR hamin(33) XOR hamin(34) XOR hamin(35) XOR hamin(36) XOR hamin(37) XOR hamin(38) XOR hamin(39)	
🕀 🔁 🔥 Synthesize - XST	26	syndrome(7) := hamin(64) XOR hamin(65) XOR hamin(66) XOR hamin(67) XOR hamin(68) XOR hamin(69) XOR hamin(70) XOR hamin(71);	
Implement Design	27		
Benerate Programming File	28	IF (syndrome = "0000000") THENno errors	~
<	< 20		>
[©] ∰ Processes	ST Deci	ian Summary 🔯 yi.vhd"	
× Started : "Launching	Design	Summary".	<u> </u>
Started : "Launching	ISE Tex	xt Editor to edit vt.vhd".	
boaroca i baanoning	100 107		
			>
🗴 🗐 Console 🛛 🕜 Errors 🔥 W	/amings		×
📔 📋 Console 🛛 🔞 Errors 🔔 W	varnings	🚾 Tcl Shell 🙀 Find in Files	
		CAPS NUM SCRU IN 11 Co	d 46 VHDL
🛃 start 🔜 👘 Xinx - ISE - F		- · · · · · · · · · · · · · · · · · · ·	all 10:33
	-American		

Figure (7): Hamming Decoder in VHDL Using ISE 10.1.

	900 ns/810 n s 510 ns 930 ns 940 ns 950 ns 960 ns 970 ns 980 ns 990 ns1000
7	711254455555444444455
6 🖌	64'h555555555555555
0	
1	
	7 6 🖌

Figure (8): Hamming Code Error Detection and Correction for a Single Error in Hexadecimal Form (With no error state).

Current Simulation ime: 1.45431e+09 n		1105390 <mark>NO</mark> s 11053920 ns 11053940 ns 11053960 ns 11053980 ns 11054000
🛚 😽 hamin[1:71]	7	71%0100101101010101010101010101010101010
🗄 🚮 dataout[1:64]	6	64%01010101010101010101010101010101010101
oli ded	0	
one 💦	1	

Figure (9): Hamming Code Error Detection and Correction for a Single Error in Binary Form (With no Error state).

	71 70 69 68 67 66 65 64 63 62 61 60 59 58 57 56 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0	THE RECEIVED DATA USING HAMMING CODE WITH FRACR 64 63 62 61 60 59 58 57 56 55 54 53 22 51 50 49 48 47 46 45 44 43 42 41 40 39 38 57 36 53 31 30 29 28 72 65 24 33 22 11 00 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 10 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0
	+ 1 0	1 1
$ \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1$		L L L L L L L L L L L L L L L L L L L
1 1	r64 1 0 1 0	1 1
1 1	10 1 0 1 0	1 1
1 1	0 1 0	1 1
1 1	n64 0 1 0	I I
DATA AFTER RERROR DETECTION AND CORRECTION IN HAMMING CODE 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0	↓ ↓ ↓ 1 0	1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0
1 0 1	0 0 0 0 0 1 1	DATA AFTER RERIOR DETECTION AND CORRECTION IN HAMMING CODE
0 1 0	1011011	0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1
	10	0 1 0

Figure (10): Hamming Code Detection Method for 64 Bits with Error State.

Current Simulation Time: 1000 ns		Detect Error Error position 900 no 910 ns 920 ns 930 ns 940 ns 985 ns 960 ns 970 ns 980 ns 990 ns1000
■ 🚮 hamin[1:71]	7*	71'h35AA55555AAAAAAAA55
■ 😽 dataout(1:64)	6	64%5555555555555555
oll ded	1	
o, i ne	0	

Figure (11): Hamming Decoder for a Single Error with Error Received Data (Error at Third Bit) in Hexadecimal Form.

Current Simulation Time: 1000 ns		Detect Error Error position 900 ns 940 ns 950 ns 960 ns 970 ns 980 ns 990 ns1000
🗄 😽 hamin[1:71]	7	71%0110101101010101010101010101010101010
🗉 😽 dataout(1:64)	6	64%01010101010101010101010101010101010101
ol ded	1	
o ne	0	

Figure (12): Hamming Decoder for a Single Error with Error Received Data (Error at Third Bit) in Binary Form.

 hamin(1:71)	dataou	ut(1:64)	
ha	mdec	ded	
		ne	

Figure (13): Schematic Circuit Diagram of Hamming Decoder.

تصميم شفرة (Hamming) لـ 64 بت لاكتشاف ومعالجة الخطأ المفرد باستخدام لغة

(VHDL)

أدهم هادي صالح

مدرس مساعد، قسم الهندسة الالكترونية، كلية الهندسة، جامعة ديالي

الخلاصة :-

شفرة (Hamming) هي تقنية لاكتشاف الخطأ وتصحيحه والتي يمكن أن تستخدم لكشف خطأ واحدة أو عدة أخطاء، أن قابلية هذه التقنية في الكشف عن الخطأ الذي يحدث لبت واحد ومعالجته تتم بكفاءة عالية و التي يمكن أن تحدث عندما يتم نقل البيانات الثنائية من جهاز إلى آخر. في أنظمة الاتصالات يتم نقل البيانات من المرسل إلى المستلم خلال الوسط الناقل ، والتي قد تعرض للتلف بسبب الضوضاء. تعمل شفرة (Hamming) على العثور على المعلومات خلال الوسط الناقل ، والتي قد تعرض للتلف بسبب الضوضاء. تعمل شفرة (Hamming) على العثور على المعلومات الأصلية من خلال الالتي المتشاف ، والتي قد تعرض للتلف بسبب الضوضاء. تعمل شفرة (Hamming) على العثور على المعلومات الأصلية من خلال المسلم الخطأ ومعالجته. هذا البحث، يصف كيف يمكننا توليد 7 بت أضافية و أضافتها للبيانات الأصلية ذات ال 64 بت. هذه البتات الاضافية سنتخلل المواقع (ن = 1، 2، 4، 8، 16، 22، 64) مع بتات البيانات الأصلي. يتم نقل البيانات والمعلومات (64 بت مع 7) لجعل سلسلة البيانات 17 بت بتقنية اكتشاف المزدوجات الثنائية عند المرسل. عند نقطة الاستلام، قد تصل البيانات ذات 71 بت بتقنية اكتشاف المزدوجات الثنائية الأصلي. يتم نقل البيانات والمعلومات (64 بت مع 7) لجعل سلسلة البيانات 71 بت بتقنية اكتشاف المزدوجات الثنائية المرسل. عند نقطة الاستلام، قد تصل البيانات ذات 71 بت وجزء من البيانات تالفة بسبب الضوضاء. تتم عند المرسل. عند نقطة الاستلام، قد تصل البيانات ذات 71 بت وجزء من البيانات تالفة بسبب الضوضاء. تتم عند المرسل. عند نقطة الاستلام، قد تصل البيانات ذات 71 بت وجزء من البيانات أم لا، وفي حال الكشف عن الخطأ سيتم عند المرسل. علم موقع الخطأ و تصحيحه . هنا استخدم 10.1 SILINX ISE المحاكاة للحاكة وهو مترجم يستخدم لمحاكاة للعثور على موقع الخطأ و تصحيحه . هنا استخدم 10.1 Strong لمرسل والمستقبل لشفرة (VHDL ولوسر والمستقبل لشفرة (Hamming) و مومنجم يستخدم لمحاكاة عامر والمستقبل ولرسم مخطط الرسم البياني، في هذا البحث تم تصميم المرسل والمستقبل لشفرة (VHDL) وهي لغةً وصفية مالحائور الالكترونية.

الكلمات الرئيسية: شفرة (Hamming)، تصحيح الخطأ، اكتشاف الخطأ، طريقة اكتشاف الخطا بالمزدوجات الثنائية، بت التكرار، لغة XILINX ISE 10.1 ،VHDL المحاكي.