TELKOMNIKA, Vol.17, No.4, August 2019, pp.1785~1795
ISSN: 1693-6930, accredited First Grade by Kemenristekdikti, Decree No: 21/E/KPT/2018
DOI: 10.12928/TELKOMNIKA.v17i4.12033

Design and implementation of single bit error correction linear block code system based on FPGA

Abdullah Mohammed A. Hamdoon, Zaid Ghanim Mohammed*, Emad A. Mohammed
Computer Eng. Technology Department of Technical Eng. College
Northern Technical University, Mosul, Iraq
*Corresponding author, e-mail: zaid_gh@ntu.edu.iq

Abstract

Linear block code (LBC) is an error detection and correction code that is widely used in communication systems. In this paper a special type of LBC called Hamming code was implemented and debugged using FPGA kit with integrated software environments ISE for simulation and tests the results of the hardware system. The implemented system has the ability to correct single bit error and detect two bits error. The data segments length was considered to give high reliability to the system and make an aggregation between the speed of processing and the hardware ability to be implemented. An adaptive length of input data has been consider, up to 248 bits of information can be handled using Spartan 3E500 with 43% as a maximum slices utilization. Input/output data buses in FPGA have been customized to meet the requirements where 34% of input/output resources have been used as maximum ratio. The overall hardware design can be considerable to give an optimum hardware size for the suitable information rate.

Keywords*: double bit error detection, FPGA, Hamming code, single bit error correction*

1. Introduction

Error detection and correction is widely used in many application fields especially in communication systems, satellite and space communications, network communications, cellular telephone networks, and any other of digital data communication. In addition, it is used in computing applications, data compression, and system coding. In noisy communication system the data transmission from transmitter to receiver suffer from errors. To overcome this problem and get error free data, there are number of error detection and correction techniques can be used. Linear block code (LBC) is one of the most common used error detection and correction methods. Hamming Code is a special case of LBC error detection and correction codes which is used to detect single or double bit errors and correct single bit errors that occur within data when it is transmitted from one device to another [1]. In traditional communication systems, the implemented hardware of error detection and correction part is designed to deals with fixed number of information data (bits) and has no ability to be reprogramed easily to meet other requirements of different communication system. To overcome this problem a flexible hardware system can be used such as Field programmable Gated Array (FPGA) [2-9].

Hamming code system based on FPGA is utilized in this work. Many sub-systems can implemented to consist the overall system hardware, each sub-systems run with its own program and need to be executed correctly, as well as, whenever data is stored or transmitted, there are chances that at least one or more bits will be an incorrect value. The transmission systems are exposing to get bits error values in either the instruction or data causing undesirable crashes or other system failures. Therefore, utilizing Hamming Code inside an embedded system is considered with high priority in modern industrial fields [1-19].

Many research works on designing and implementation Hamming code with embedded system with FPGA kits have been introduced. Hamming code as hardware system using VHDL or Virlog as hardware description language is proposed to demonstrate and analyze the performance of the system. Most of these works have limitation on the number of data bits and hardware resources [7-19]. Literary review for the most important research involved this approaches with their limitation can be summarized:

- ECC application on embedded systems is designed for memory error detection and correction it is realized a 64-bit ECC controller has been proposed in [10].

Received January 31, 2019; Revised April 12, 2019; Accepted May 2, 2019

- 3D Parity check code with data stored in memory system with maximum data bits about 128 bits has been proposed in [11].
- CRC as a hardware system has been implemenated with PIC microcontroller as in [12].
- Demonstrate implementation of Hamming code as a hardware system with simple 8 bit of data length has been explained in [13].
- Design a Hamming code system based on IP core with 56 data bits for the transmission data of communication system is demonstrated in [14].
- Hamming code architecture which is used as an alternative error detection code scheme in Controller Area Network technology, the implementation done on set of inputs from 8-bit to 64-bit of data frames has been explained in [15].
- A code rate of (15, 11) algorithm for design Hamming code as hardware system is utilized in [16], while a code rate of (7, 4) algorithm is used in with Verilog as hardware description languge [17].

From previous works, which are intended in this research to present an essential error detection technique that can deal with adaptive information data bits length up to 248 bits is introduced. This research organize as first introduces Hamming code theory, then shows hardware design of Hamming algorithm with decoding and encoding of it, finally the obtained result is demonstrated and discussed with conclusion.

2. Single bit Error Correction (Hamming code) Theory

The key to the Single bit error correction is the use of extra parity bits to allow the identification of a single error. Two methods (even parity, odd parity) for generating redundancy bits that Hamming code need it. Number of redundancy bits are generated (Check bits) is calculate according to (1). This redundancy bits are depends on the number of information data bits [7-8-13-14].

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

where k is the data bits length, and r is the bits to find the check bits that will add to data.

Summarize table for the redundant (check bits length opposite to the data bits length) as shown in Table 1. The operation of hamming code extended can be summarize as following:

a. Mark all bit positions that are powers of two as parity bits (positions 1, 2, 4, 8, 16, 32, 64, etc.).
b. All other bit positions are for the data to be encoded (positions 3, 5, 6, 7, 9, 10, 11, 12, 13, 14, 15, 17, etc.).
c. The last bit is added for Parity bit.
d. Each parity bit calculates the parity for some of the bits in the code word. The position of the parity bit determines the sequence of bits that it alternately checks and skips.
   - Position 1: check 1 bit, skip 1 bit, check 1 bit, skip 1 bit, etc. (1, 3, 5, 7, 9, 11, 13, 15,...)
   - Position 2: check 2 bits, skip 2 bits, check 2 bits, skip 2 bits, etc. (2, 3, 6, 7, 10, 11, 14, 15, ...)
   - Position 4: check 4 bits, skip 4 bits, check 4 bits, skip 4 bits, etc. (4, 5, 6, 7, 12, 13, 14, 15, 20, 21, 22, 23, ...)
   - Position 8: check 8 bits, skip 8 bits, check 8 bits, skip 8 bits, etc. (8-15, 24-31, 40-47, ...) and so on for the position 16 and 32 …etc.
e. Set a Parity bit to 1 if the total number of ones in the positions it checks is odd (XOR operation between all bits) . Set a parity bit to 0 if the total number of ones in the positions it checks is even.
f. To test packet data received, the XOR is applied on all bits to determine if there is any error in Parity bit; Parity bit is extracted; same algorithm is applied on rest bits to generate hamming code.
g. If hamming code is zero and Parity bit is zero, then there are no error in received packet data. If hamming code is not zero and Parity bit is one, then there is one error in packet data and correction is capable by invert the bit location that pointed by hamming code value. If hamming code is not zero and Parity bit is zero, then there are two errors or even errors in packet data and cannot correction.

Table 2 explains the data packet is (10001000) and check bit is unknown so it is calculated. Table 3 add the check bits to the data with test is that it free from error. In Table 4, an error is add to the data packet D4 bit which become D6 bit after combine it with check bits.

Table 1. Check Bits Opposite Data Bits Length

Data length Bits (k)	Check bit length (r)
2-4	3
5-11	4
11-26	5

Table 2. First Stage Encoding the Data

	D1	D2	D3	D4	D5	D6	D7	D8					Parity bit	Total Bits
Data	1	0	0	0	1	0	0	0						k=8 bit
Check Bits Location	P1	P2		P4				P8						r=4 bit
	D1	D2	D3	D4	D5	D6	D7	D8	D9	D10	D11	D12	D13	k+r
over all bits	?	?	1	?	0	0	0	?	1	0	0	0		Pb=?
redundant P1	?		1		0		0		1		0			P1=xor(?10010)=0
redundant P2		?	1			0	0			0	0			P2=xor(?10000)=1
redundant P4				?	0	0	0					0		P4=xor(?0000)=0
redundant P8								?	1	0	0	0		P8=xor(?1000)=1

Table 3. Appling Check Bits (Decoding)

	D1	D2	D3	D4	D5	D6	D7	D8					Parity bit	Total Bits
Data	1	0	0	0	1	0	0	0						
Check Bits Location	P1	P2		P4				P8						
	D1	D2	D3	D4	D5	D6	D7	D8	D9	D10	D11	D12	D13	k+r
over all bits	0	1	1	0	0	0	0	1	1	0	0	0	0	xor(011000011000)=0
redundant P1	0		1		0		0		1		0			xor(010010)=0
redundant P2		1	1			0	0			0	0			xor(110000)=0
redundant P4				0	0	0	0					0		xor(00000)=0
redundant P8								1	1	0	0	0		xor(11000)=0

Table 4. Test Data with Bit Error

	D1	D2	D3	D4	D5	D6	D7	D8					Parity bit	Total Bits
Data	1	0	0	1	1	0	0	0						
Check Bits Location	P1	P2		P4				P8						
	D1	D2	D3	D4	D5	D6	D7	D8	D9	D10	D11	D12	D13	
over all bits	0	1	1	0	0	0	1	1	1	0	0	0	1	xor(011000111000)=1
redundant P1	0		1		0		1		1		0			xor(010110)=1
redundant P2		1	1			0	1			0	0			xor(100100)=1
redundant P4				0	0	0	1					0		xor(00010)=1
redundant P8								1	1	0	0	0		xor(11000)=0
														D7 is the error bit

3. Hardware Design of Hamming Algorithm

The Hamming code is a system that encoding data depending on its contents which means that Data Packet Encoding is changed as the data information changed. The information's data can be extract form data packet needed by removing the hamming bits. The error detection and correction can be done by applying the same algorithm at encoding. The hamming code uses for detecting errors in data packet and correct one bit only. In many systems, the data's information size is very large, and to create Data Packet Encoding needs many clock pulses to input data information to Hamming Encoding System at many sequence slice input, and the output data packet encoding is divided into a sequence slices output to input them to a next system processing. The customizing data bus in\out width is used to control on the width of input/output data bus. The constants parameters are used to increase or decrease width of input/output data bus as need. If a data bus slice is large, the data is input/output fast to/from Hamming Encoding/Decoding System, but the large data size exhausting the hardware input/output pins. If a data bus size is small, the data input/output processing is slow, and needed less hardware pins to input and output data. The selection of data information size uses to control on size of hardware used and cost of all design [20-25].

The encoding/decoding hamming system design is configured via a parameters of constant data. These parameters are determins the following:

a. The data bus input Slice to system (data_width_slice_in constant).
b. The data bus output Slice from a system (data_width_slice_out constant).
c. The size of data Information processing in system (information_bits constant).
d. The size of hamming code depends on size of data information (ham_bits constant).

The size of data packet encoding is determined by adding three parameters, the size of Data Information Size, the Hamming Code Size and the number of parity bit (packet_bits constant). The design can be implemented for deference implementation with select different parameters and monitor the features of system design implementation of FPGA device, and then select the suitable parameters for the design.

3.1. Customizing Input/Output Data Bus Size

The data bus size at input and output can be changed by two parameters (data_width_slice_in and data_width_slice_out constant). The size can be changed from one bit (serial transfer) to any size suitable for the system input and output data. For example if data input to the system is 17 bit, then the data_width_slice_in is programmed to number 17. If the size of data bus of the system receive data from our system is 23 bit, then the data_width_slice_out is programmed to number 23. By using this method the system can be connected to any system data bus by programming the data bus size as the size of the system connected.

3.2. Encoding Design

If the new data information is applied to the encoding system, the system begin input data and a Data Packet Encoding is produced. As shown in Figure 1 the hardware signals with block diagram of the encoding of the designed system that if the new_data_flag signal is zero and new_info_packet signal is one to initialize the controls, flags, and counters signals to initialize values at rising edge of first clock, then the new_data_flag signal is one to begin input part of Information Data Signal. The Data Bus Input (data_in port) is load to low byte of data_info_new signal at second clock until the whole Information Data Signal inputs on it. The size of Data Bus Input Size (data_width_slice_in constant) and Information Data Size (information_bits constant) are determined as constants definition in a design. The number of clocks needing to input all Information Data signal bits is determined by subtraction the Data Bus Input Size from Information Data Size at each clock input data. The last input bits that result from remainder of subtraction are input from the low Data Bus Input bits, as result, the excess bits at high Data Bus Input bits are ignored. For each data bits input are collected in Information Data Signal. The size of this signal is determined by Information Data Parameter.

The Information Data Signal is reconstruction by insert the location of hamming bits, then adding one bit for parity after last bit. The location of hamming bits in the encoding signal are 1, 2, 4, 8,…, 2n , where n is the Hamming Code Size (ham_bits constant). The size of Encoding Signal (data_encode signal) is produced by addition of three parameters, Hamming Code Size, Information Data Parameter, and the number for parity bit. The Encoding Signal is

changed to the Data Packet Encoding Signal (data_packet_out signal) by calculated the values of hamming bits in Encoding Signal and parity bit. Each hamming bit has its equation for calculating its value. For example hamming bit (1) value is calculated by Exclusive OR (XOR) of the locations (3, 5, 7 ,9,…). The parity bit is calculated by Exclusive OR (XOR) for all bits of Data Packet Encoding Signal and stored at last bit of it.

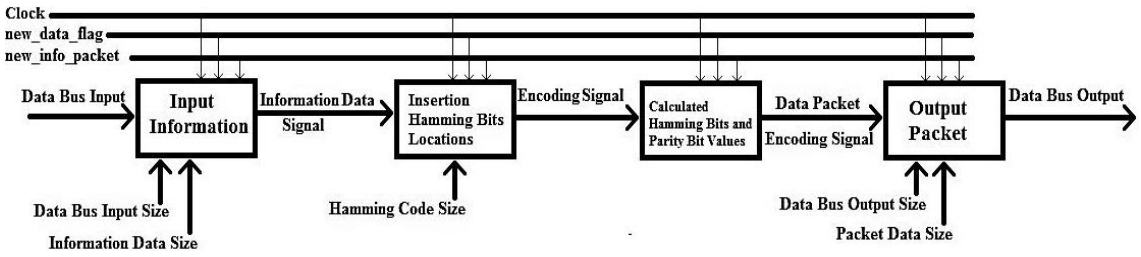

Figure 1. Block diagram of encoding system

After the Data Packet Encoding Signal built, its output to the next system. The width of Data Bus Output (data_out_packet port) is determine by the Data Bus Output Size (data_width_slice_out). The number of clocks needing to output all bits of the Data Packet Encoding Signal is determined by subtraction The Data Bus Output Size (data_width_slice_out constant) from the Packet Data Size (packet_bits constant). The last output bits that produced from remainder of subtraction are only updated at the Data Bus Output and others bits are zero, as result, the excess bits are ignored from received system. To repeat the Encoding operation, the new_data_flag signal must be zero to Reset the values of counters, flags, and control signal that needed to input new information.

3.3. Decoding Design

If the new data Packet is applied to the decoding system, the system begin input data and a Data Information Encoding is produced. As shown in Figure 2, the hardware signals with block diagram of the decoding of the designed system that if the new_data_flag signal is zero and new_info_packet signal is zero to initialize the controls, flags, and counters Signals to initialize values at rising edge of first clock pulse, then the new_data_flag signal is one to begin input part of Data Packet Encoding Signal. The Data Bus Input (data_in port) is load low byte of data_info_new signal at second clock until the whole Data Packet Encoding Signal inputs on it. The size of Data Bus Input Size (data_width_slice_in constant) and Information Data Size (information_bits constant) are defiened as constants definition in a design. The number of clock pulses needing to input all Data Packet Encoding Signal bits is determined by subtracting the Data Bus Input Size from the Packet Data Size. The last input bits that result from remainder of subtraction are input from the low Data Bus Input bits, as result, the excess bits at high Data Bus Input bits are ignored. For each data bits input are collected in Packet Data signal. The size of this signal is defined by Packet Data Size.

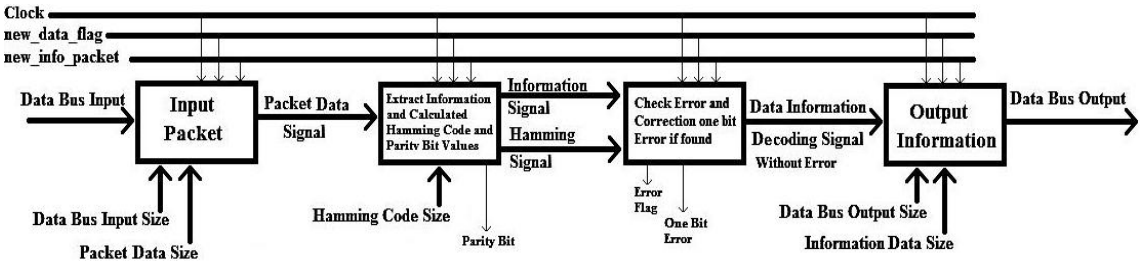

Figure 2. Block diagram of decoding system

The Data Information Signal (data_information_read signal) is extracted from Data Packet Encoding Signal by canceled the Hamming code bits locations and values and parity bit. The Data Information Signal size new depending on Information Data Size constant. While extraction, the Hamming Code Signal (ham_test signal) is calculated from Data Packet Encoding Signal by applying hamming code algorithm. The parity bit (parity_bit signal) is calculated, if zero there is no parity error in packet, else parity error is found.

If Hamming Code Signal value's is not equal to zero and less than packet_bits constant and parity bit equal to zero, then the one bit error correction is applied and both one_error_bit and error flag signals are ones. If Hamming Code Signal value's is not equal to zero and less than packet_bits constant and parity bit equals to one, then the one bit error correction cannot applied and one_error_bit signal equal to zero and error_flag signal is equal to one. If Hamming Code Signal value's is not equal to zero and large than packet_bits, then the one bit error correction cannot applied and one_error_bit signal equal to zero and error_flag signal is equal to one. If Hamming Code Signal value's is equal to zero and parity bit equal one, then the one bit error correction can not applied and one_error_bit signal equal zero and error_flag signal is equal are one. If Hamming Code Signal value's is equal to zero and parity bit equal to zero, then the one bit error correction is applied and one_error_bit and error_flag signals are equal to zeros. Finally, the Data Packet Signal that not has error or one bit correction is done on it can be decoding to extract Data Information Signal from it.

After the Data Information Signal build its output to the next system. The width of Data Bus Output (data_out_info port) is determine by the Data Bus Output Size (data_width_slice_out). The number of clocks needing to output all bits of the Data Information Signal is determined by subtraction the Data Bus Output Size (data_width_slice_out constant) from the Information Data Size (info_bits constant). The last output bits that produced from remainder of subtraction are only updated at the Data Bus Output and others bits are zeros, as result, the excess bits are ignored from received system. To repeat the Encoding operation, the new_data_flag signal must be Reset to reset the values of counters, flags, and control signal that needed to input new packet.

4. Results of the Designed System

Figures 3 and 4 shows how can input information data and output packet data with hamming encoding. The information data input value is "1AA00"h as example. The following steps describe the operations work:

a. The new_data_flag is equal to zero and new_info_packet is equal to one to initialize the controls, flags, and counters Signals at rising edge of first clock pulse.
b. The new_data_flag is one and first byte is input from data_in port and stored at low byte of data_info_new Signal at rising edge of clock. The values of input data at each rising edge in sequentially are "00" h,"aa" h, and "01"h. The two counters (info_packet_in_count and info_packet_location_in_count) are used to input data; the info_packet_in_count Signal counter is used to determine number of clock pulses needing to input all data; Its value is subtracted from data_bus_in_size parameter at each clock pulse; when its value less than data_bus_in_size parameter or equal to zero, its Reset to zero and input remaining bits from data_in port if found. info_packet_location_in_count Signal is used to determine the storing location in data_info_new Signal; when its value one, the data_in port value's is stored in low location depending on size of data_bus_in_size; then its increment until all data input on data_info_new Signal.
c. The remaining bytes are input from data_in port and stored at data_info_new Signal at each clock pulse until the remainder bits less than data_bus_in_size. In this case the remainder bits only input and stored in high bits of data_info_new Signal. The value of data_info_new signal is equal "1AA00"h.
d. When data completes input, the info_packet_read signal is one.
e. The next clock is used to encoding information by adding the location of hamming bits and parity bit with zero values. The data_encode Signal is produced and done_encode signal is one. The value of data_encode Signal is equal "352000"h.
f. The next clock uses to calculate the hamming bits values by using hamming code algorithm and stored result in data_packet_out port. The parity bit is calculated and the final value of data_packet_out Signal is equal "75208B"h. The value of ham_code Signal is equal "0f"h.

g. The next clock begins checking request_info_packet_out input port, if its value equal to zero, the output is not ready, if its value equal to one the data is begin output directly in a number of stages. The low bits that their size equal data_bus_in_size parameter are applied on output. The values of output data at each rising edge in sequentially are "8b" h,"20" h, and "75"h. The two counters (info_packet_out_count and info_packet_location_out_count) are used to output all data. The two counters are operate as the two counters at the input data and for the output data.
h. When all data output, info_packet_write Signal is one.

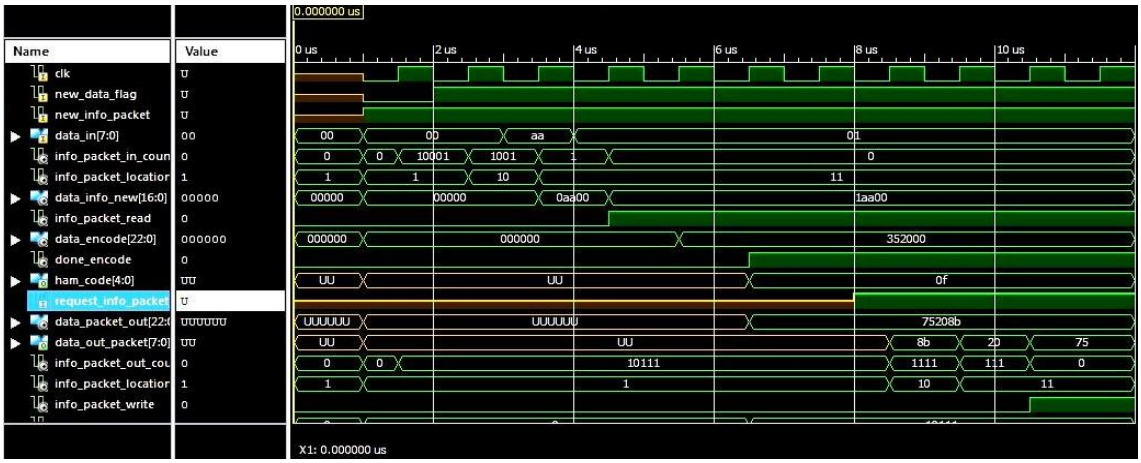

Figure 3. Test results on input information and output packet data for the encoding system

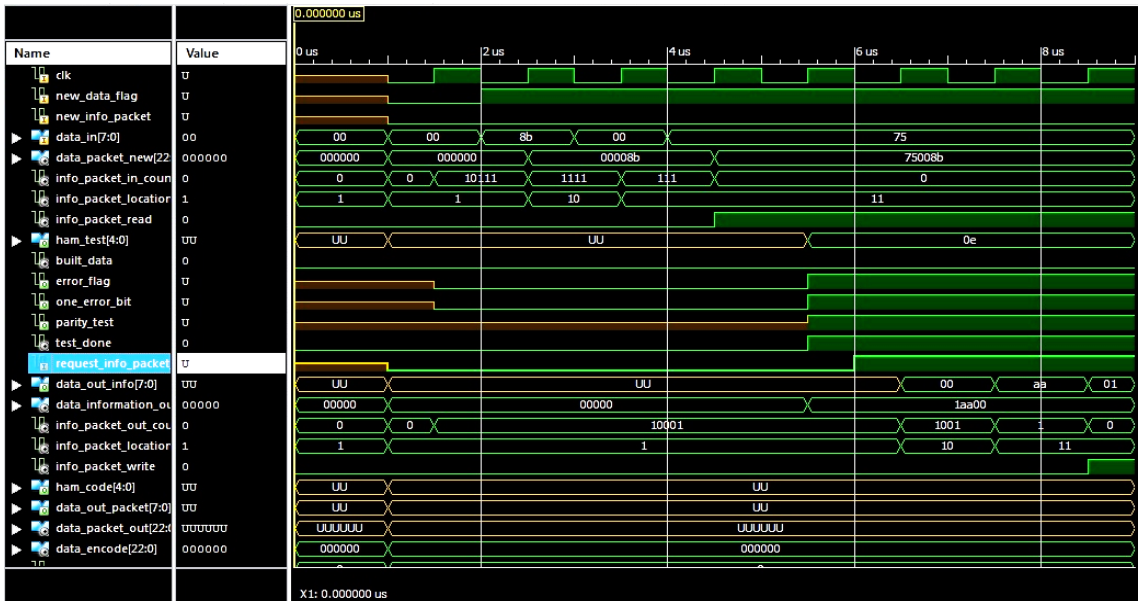

Figure 4. Test results the encoding system

Figures 5 and 6 show how to input packet data and output information data without hamming encoding. The packet data input value is "75208B"h as example. The following steps describe the operations work:

a. The new_data_flag is equal to zero and new_info_packet is equal to zero to Initialize the controls, flags, and counters signals at rising edge of first clock pulse.
b. The new_data_flag is equal to one and first byte is input from data_in port and stored at low byte of data_packet_new Signal at rising edge of clock pulse. The values of input data at

each rising edge in sequentially are "8b" h,"20" h, and "75"h. The two counters (info_packet_in_count and info_packet_location_in_count) are used to input data; the info_packet_in_count Signal counter is used to determine the number of clock pulses needed to input all data; Its value is subtracted from data_bus_in_size parameter at each clock pulse; when its value less than data_bus_in_size parameter or equal to zero, its Reset to zero and input remaining bits from data_in port if found. info_packet_location_in_count Signal is used to determine the storing location in data_packet_new Signal; when its value equal to one, the data_in port value's is stored in low bits locations depending on the size of data_bus_in_size; then its increment until all data input on data_packet_new Signal.

c. The remaining bytes are input from data_in port and stored at data_packet_new Signal at each clock pulse until the remainder bits less than data_bus_in_size. In this case the remainder bits only input and stored in high bits of data_packet_new Signal. The value of data_packet_new signal equal "75208B"h.
d. When data completes input, the info_packet_read signal value is one.
e. The next clock pulse is used to decoding information by remove the values and locations of hamming bits and parity bit. Then, checking the error in the data_packet_new Signal. The information data is stored in data_out_information signal. The ham_test signal is produced by applying hamming algorithm on data_packet_new signal. The parity_bit signal is produced by applying XOR on all bits of packet. If the ham_test signal value is equal to zero The error_flag and the one_error_bit signals value is equal to zero, if one error found in packet data, the error_flag and the one_error_bit signals value is equal to one and Error correction is done on data_out_information signal, if ham_test signal value larger than value of packet_bits signal or parity bit is equal to one, then more than one error found in packet data, the error_flag signal value is one and the one_error_bit signal value is zero. At last, The test_done signal value is one. The value of ham_test Signal is equal "00"h. The value of parity_test Signal is equal to zero.
f. The next clock begins checking request_info_packet_out input port, if its value equal to zero, data not output, if its value one and test_done signal value is one, then the data begins output directly in a number of stages. The low bits that their size equal to data_bus_in_size constant are applied on output. The values of output data at each rising clock in sequentially are "00" h,"aa" h, and "01"h. The two counters (info_packet_out_count and info_packet_location_out_count) are used to output all data. The two counters are operate as the two counters at input data and for output data.

When all data output, info_packet_write Signal value is one.

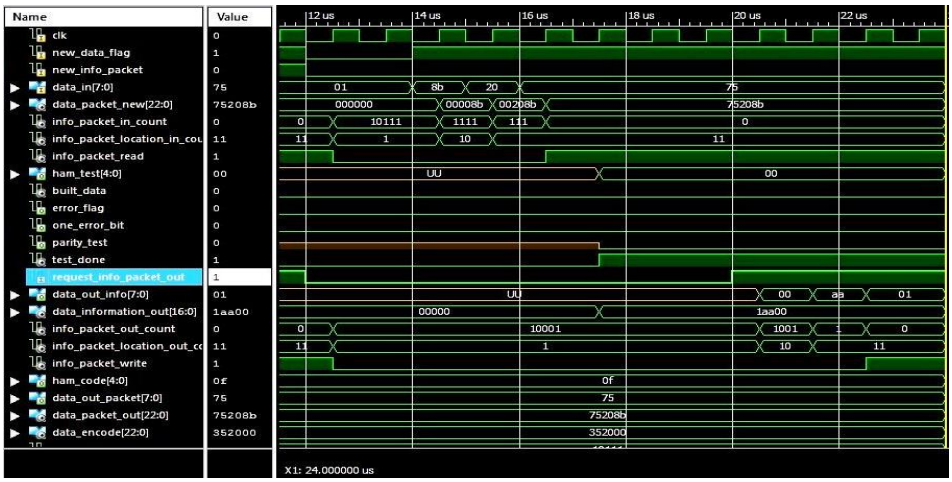

Figure 5. Results with test of the decoding system

Table 5 demonstrate the description of the signals used in simulation the designed system. Tables 6-8 show comparison of the hardware utilization of designed system of different bits of Hamming code data information 22 bits in Table 5, 107 bits in Table 6 and 248 bits in Table 7.

Table 5. Signals and Port Name Description of the Designed System

Signal or Port Name	Description
clk	it's a clock that using for synchronous all operations
new_data_flag	Its input control port using to enable input and output data when its value equal to one, and Reset controls, flags, and counters Signals when its value equal to zero.
new_info_packet	Its input control port to determine the current operation, if its value one, then the operation is input information data and encoding it to packet data with hamming code algorithm. If its value equal to zero, then the operation is input packet data and decoding it to information data with hamming code algorithm checking and extraction.
data_in	Its data input bus. Its width determine by the data_width_slice_in constant.
data_info_new	It's a signal to store all input information data from data_in port. Its width determine by the information_bits constant.
data_encode	It's a signal to store the encoding information data of data_info_new signal. It has the information data and hamming bits location with zero values. Its width determine by the information_bits constant plus ham_bits constant.
data_packet_new	It's a signal to store all input packet data from data_in port. Its width determine by the information_bits constant plus ham_bits constant.
data_packet_out	It's a signal to store the packet data signal that output. Its value is data_encode signal with calculate hamming bits value. Its width determine by the information_bits constant plus ham_bits constant
info_packet_read	Its signal that its value is one when data is complete read. At RESET its value is equal to zero.
done_encode	Its signal that its value is one when data is complete encoding. At RESET its value is zero.
ham_test	It's a signal that its value is produced from applying hamming code algorithm on data_packet_new signal. If its value equal to zero there are no error in packet and error_flag and one_error_bit signals are equal one, if its value not equal to zero
error_flag	It's a signal that its value is one when there is any error in data_packet_new signal.
one_error_bit	It's a signal that its value is one when there is one error in data_packet_new signal. If ham_test signal value large than size of packet_bits constant value then error cannot correction and one_error_bit value is equal to zero. If ham_test signal value less or equal size of packet_bits constant and parity_test equal zero, then one bit error correction is applied and one_error_bit is equal to one. If ham_test signal value less or equal size of packet_bits constant and parity_test equal one, then cannot apply error correction one_error_bit is equal to zero.
parity_test	It's a signal that its value is zero when there is no parity error in data_packet_new signal.
test_done	It's a signal that its value is one when the test is completed on data_packet_new signal.
info_packet_write	It's a signal that its value is one when data is complete output. At RESET its value is equal to zero.
request_info_packet_out	It's a input control port to give permission output data. If its value equal to zero, data not output. If its value one and packet is ready to output, its output data at data_out_packet port in number of stages depending on data_out_packet port size and data_packet_out signal size.
data_out_packet	Its data output bus. Its width determine by the data_width_slice_out constant.
info_packet_in_count	It's a counter signal uses to determine the number of stages needing to input all data (information or packet) from data_in port. Its value equal to the information_bits constant value if information data input or information_bits constant plus ham_bits constant value if packet data input. At RESET its value is equal information_bits if new_info_packet equal to one, else is equal information_bits plus ham_bits.
info_packet_location_in_count	It's a counter signal uses to determine locations of bits storing in data_info_new signal from data_in port. When its value is one, the input data from data_in port is stored at low bits locations of data_info_new signal or data_packet_new signal; if its value is two, the input data from data_in port is stored at next low bits locations of data_info_new signal or data_packet_new signal. The size of low bits locations depending on size of data_width_slice_in constant. At RESET its value is equal to one.
info_packet_out_count	It's a counter signal uses to determine the number of stages needing to output all data (information or packet) to data_out_packet port or data_out_info port. Its value equal to the information_bits constant value if information data output or information_bits constant plus ham_bits constant value if packet data output. At RESET its value is equal to information_bits plus ham_bits if new_info_packet equal one, else is equal information_bits.
info_packet_location_out_count	It's a counter signal uses to determine the locations of bits output from data_packet_out signal to data_out_packet port if packet output operation, or determine the locations of bits output from data_information_out signal to data_out_info port if information output operation. When its value is one, the low bits locations of data (data_packet_out or data_information_out signal) is output to output port (data_out_packet or data_out_info port) respectively; when its value is two, the next the low bits locations of data (data_packet_out or data_information_out signal) is output to output port (data_out_packet or data_out_info port) respectively;. The size of low bits locations depending on size of data_width_slice_out constant. At RESET its value is equal to one.

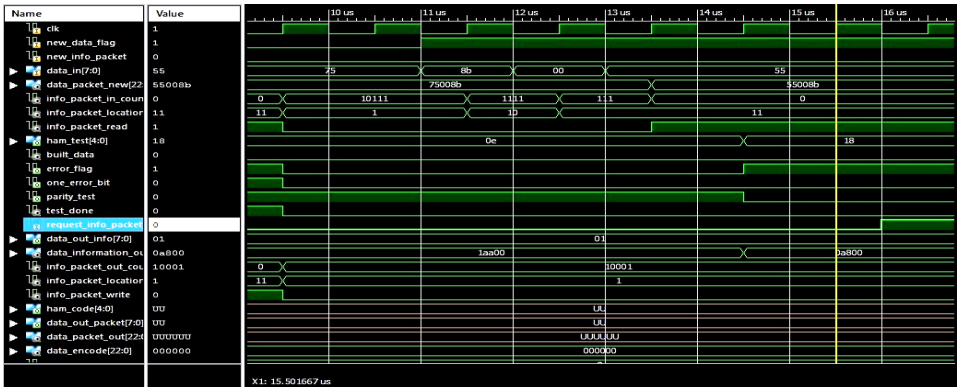

Figure 6. Results with test of the decoding system

Table 6. Utilization Summary of Designed System with Hamming Code of 22 Bit of Information Data

Device Utilization Summary			
Logic Utilization	Used	Available	Utilization
Number of Slice	158	4,656	3%
Number of Slice Flip Flops	163	9,312	1%
Number of 4 input LUTs	238	9,312	2%
Number of bonded IOBs	40	232	17%
Number of GCLK	1	24	4%

Table 7. Utilization Summary of Designed System with Hamming Code of 107 Bit of Information Data

Device Utilization Summary			
Logic Utilization	Used	Available	Utilization
Number of Slice	545	4,656	11%
Number of Slice Flip Flops	573	9,312	6%
Number of 4 input LUTs	883	9,312	9%
Number of bonded IOBs	78	232	33%
Number of GCLK	1	24	4%

Table 8. Utilization Summary of Designed System with Hamming Code of 248 Bit of Information Data

Device Utilization Summary			
Logic Utilization	Used	Available	Utilization
Number of Slice	2004	4,656	43%
Number of Slice Flip Flops	1525	9,312	16%
Number of 4 input LUTs	3700	9,312	39%
Number of bonded IOBs	80	232	34%
Number of GCLK	1	24	4%

5. Conclusion

In this paper, the implemented algorithm of Hamming Code has been designed to customized data information as embedded system. The implemented system has been achieved to select different parameters (22 bits), (107 bits) and (248 bits) of information data with hardware resource utilization about 3%, 11% and 43% respectively of FPGA slices and 17%, 33% and 34% respectively of input/output with ability to develop the designed system for any number of bits as data information, the obtained resulted was tested with (ISE) simulator and it met the required for the designed system. The end user need to understand the algorithm parameters of the designed system that will need it to modify and then the algorithm will be fined the error and correct it with one bits. The above results can be enhanced with advanced board of FPGA device.

References

[1] Forouzan BA. Data Communication and Networking. Fourth edition. McGrawHill Ltd. 2007: 267-299.
[2] St. Onge LM, Areibi S. *VHDL for Digital Design*. Technical Report 2003-01P School of engineering, university of Guelph, Canada. 2003.
[3] Perry DL. VHDL Programming by Example. Fourth Edition. McGraw-Hill Ltd. 2002.
[4] Mohammed ZG, Hamdoon AMA, Aziz MS. *Scheduling lecturer system based on FPGA.* IEEE International Conference on Advances in Sustainable Engineering and Applications (ICASEA). 2018: 54-58.
[5] Peckol JK. Embedded Systems a Contemporary Design Tool. First Edition. 2007: 597-648.
[6] Bhattacharya R. Model & Platform Based Design of Embedded Systems. Texas A&M University College Station. TX 77843-3141. 2006.
[7] Moreira JC, Farrell PG. Essentials of Error-Control Coding. John Wiley & Sons Ltd. 2006: 41-77.
[8] Wang A, Kaabouch N. *FPGA based design of a novel enhanced error detection and correction technique*. 2008 IEEE International Conference on Electro/Information Technology. 2008; 3(5): 25–29.
[9] Lee H, Sung J, Kim E. *Reducing Power in Error Correcting Code using Genetic Algorithm*, Proceeding of World Academy of Science, Engineering and Technology. 2007; 21: 179-182.
[10] Yan C, Du L, Wang Z. Design of ECC Controller and its Validation Based on FPGA. *TELKOMNIKA Indonesian Journal of Electrical Engineering*. 2014; 12(10): 7253-7261.
[11] Tambatkar S, Menon SN, Sudarshan V, Vinodhini M, Murty NS. *Error Detection and Correction in Semiconductor Memories using 3D Parity Check Code with Hamming Code*. IEEE Int. Conf. on Communication and Signal Processing (ICCSP). 2017: 974–978.
[12] Ray J, Koopman P. *Efficient high Hamming distance CRCs for embedded networks*. Proc. Int. Conf. Dependable Syst. Networks. Philadelphia PA. 2006: 3–12.
[13] Hamzah SBA. Hamming Code Using Field Programmable Gate Array (FPGA). Doctoral dissertation. Universiti Teknikal Malaysia Melaka; 2014.
[14] Irudayaraj IR, Haroon PSA, JU, Bilagi SS. *Design and Verification of Improved Hamming Code (ECC) Using Verilog*. Proc. of ISETE Int. Conf. Bengaluru. 2017: 14–18. ISBN: 978-93-86291-63-9.
[15] Juan ROS, Jeong MW, Cha HW, Kim HS. *FPGA Implementation of Hamming Code for Increasing the Frame Rate of CAN Communication*. IEEE Asia Pacific Conf. Circuits Syst (APCCAS). 2017: 684–687.
[16] Zhang T, Ding Q. *Design of (15, 11) Hamming Code Encoding and Decoding System Based on FPGA*. IEEE Int. Conf. on Instrumentation, Measurement, Computer, Communication and Control (IMCCC). 2011: 704–707.
[17] Shep N, Bhagat PH. Implementation of Hamming Code Using VLSI. *Int. Journal of Eng. Trends Technol*. 2013; 4(2): 186–190.
[18] Tam S. Single Error Correction and Double Error Detection. *Xilinx Application Note*. 2006; 645: 1–12.
[19] Raha P, Vinodhini M, Murty NS. *Horizontal-Vertical Parity and Diagonal Hamming Based Soft Error Detection and Correction for Memories*. IEEE Int. Conf. on Computer Communication and Informatics (ICCCI-2017). 2017: 1-5.
[20] Xilinx, Inc. Spartan 3E Starter Kit Board. User Guide UG230 (v1.2). June 20, 2011.
[21] Xilinx, Inc. Spartan-3E FPGA Family: Functional Description. Data Sheet DS312-2(v4.2). Desember 14, 2018: 10-114.
[22] Lee EA, Seshia SA. Introduction to Embedded Systems a Cyber-Physical Systems Approach. Second Edition. MIT Press Ltd. 2017: 135-372.
[23] Xilinx, Inc. Using the ISE Design Tools for Spartan-3 FPGAs. Data Sheet XAPP473 (v1.1). May 23, 2005.
[24] Chu PP. FPGA Prototyping by VHDL Examples Xilinx Spartan-3 Version. John Wiley & Sons Ltd. 2008: 11-106. ISBN 978-0-470-18531-5.
[25] Haskell RE, Hanna DM. Introduction to Digital Design Using Digilent FPGA Boards Block Diagram/VHDL Examples. Rochester: LBE Books, LLC Ltd. 2009: 1-112