IJERMT

International Journal Of Engineering Research & Management Technology

ISSN: 2348-4039

Email: editor@ijermt.org March 2014 Volume-1, Issue-2 Website: ijermt.org

# Hamming Code for Error Detection and Correction using VHDL

**Himanshu Sharma**
**M.Tech Scholar**

**Amit Kumar**
**Assistant professor**

**Department of Electronics and Communication**
**SRM University NCR campus.**

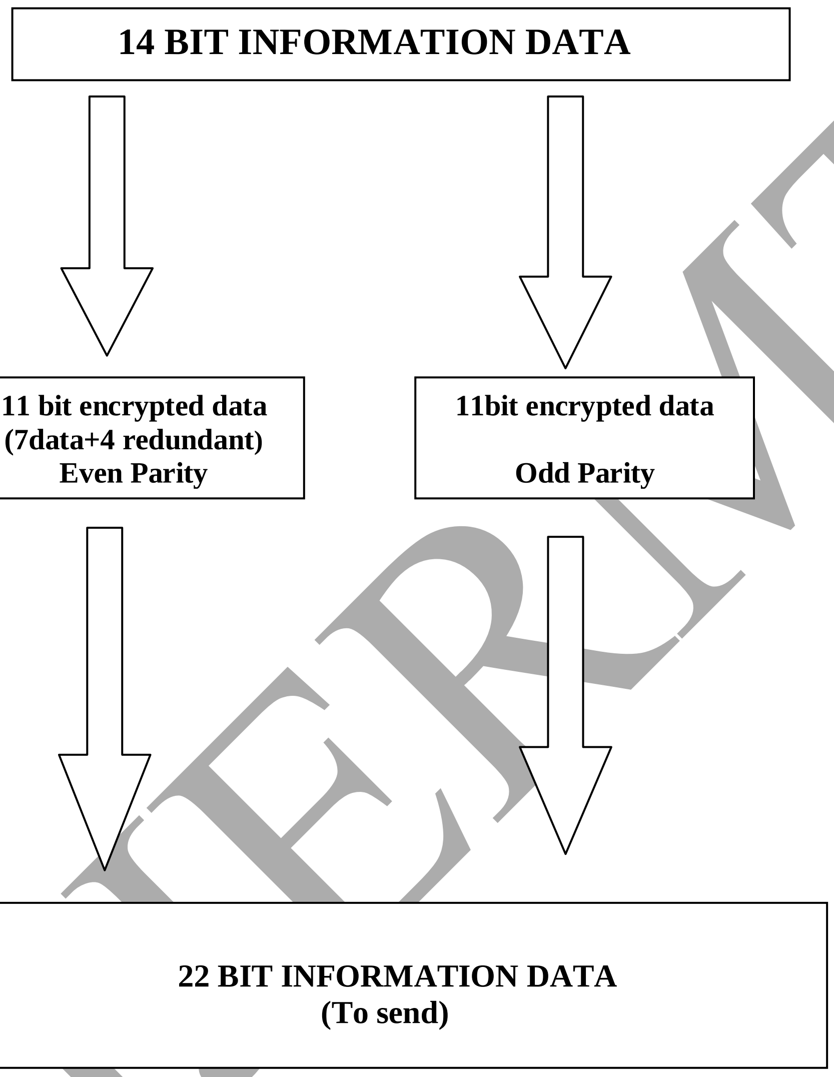
**ABSTRACT**
Hamming code for error detection and correction methodology is used for error free communication in communication systems.
In communication system information data is transferred from source to destination through channel. In between, source and destination, data may be corrupted due to any type of noise. To find original information we use Hamming code error detection and correction technique.
In hamming code error detection and correction technique, to get error free data at destination, we encrypt information data according to even and odd parity method before transmission of information at source end.
In this project 14 bit Information data is divided into two groups of 7bit data. After this division each group is encrypted using four Redundancy bit.
In this way 14bit information data at sender end is now converted into two groups of 11bit data (with redundant bits). Both groups are joined to make total 22 bit Information data.
At Receiver end 22 bit information data in divided into two groups of 11 bit data and each group is detected and corrected using even and odd parity check method.
After correction both group are joined together to Receive 22bit original information data.
Here, Xilinx ISE 10.1 Simulator has been used for simulating VHDL Code. Xilinx ISE 10.1 S is a simulator which is used for simulating HDL language and schematic circuit diagram. Here we have used Xilinx simulator to simulate VHDL code for transmitter and receiver.

**Keywords –** Hamming code, Odd parity check method, even parity check method, Redundancy bits, VHDL language, Xilinx ISE 10.1 Simulator.

## 1. INTRODUCTION

In communication system, a secure data transmission from transmitter to receiver is very major issue. For error free transmission there are number of technologies. One of them is hamming code method. Hamming code works on the parity check method. Parities are of two type first even parity and second odd parity .Here we use both even parity and odd parity method to encrypt data before transmission.
In this paper 14 bit Information data is divided into two groups of 7bit data. After this division each group is encrypted using four Redundancy bits.
In this way 14bit information data at sender end is now converted into two groups of 11bit data (with redundant bits). One group is encrypted using even parity and another by odd parity.
Suppose, we want to transmit 14 bit information data bit as "1101001". For this 14 bit information data we need, two set of 4 redundancy bit and these are "0101" and "1010"by using even and odd parity method respectably. After generating redundancy bits, add these bits to 7 bit information data for making two group of 11 bit data string for transmission at source end. Now both these group are joined to make a total of 22 bit information data.

At destination receiver receives 22 bit data string from channel and divide it into two group of 11 bit data. Each group is detected for error using even parity and odd parity check method. If this data string is corrupted then receiver find the error location according to parity check method and correct that bit.


14 BIT INFORMATION DATA
11 bit encrypted data (7data+4 redundant) Even Parity
11bit encrypted data Odd Parity
22 BIT INFORMATION DATA (To send)

**DATA AT SENDER END**

## 2. 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.
Thus, reliable communication is possible when the Hamming distance between the transmitted and received bit patterns is less than or equal to one. 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 error. 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.

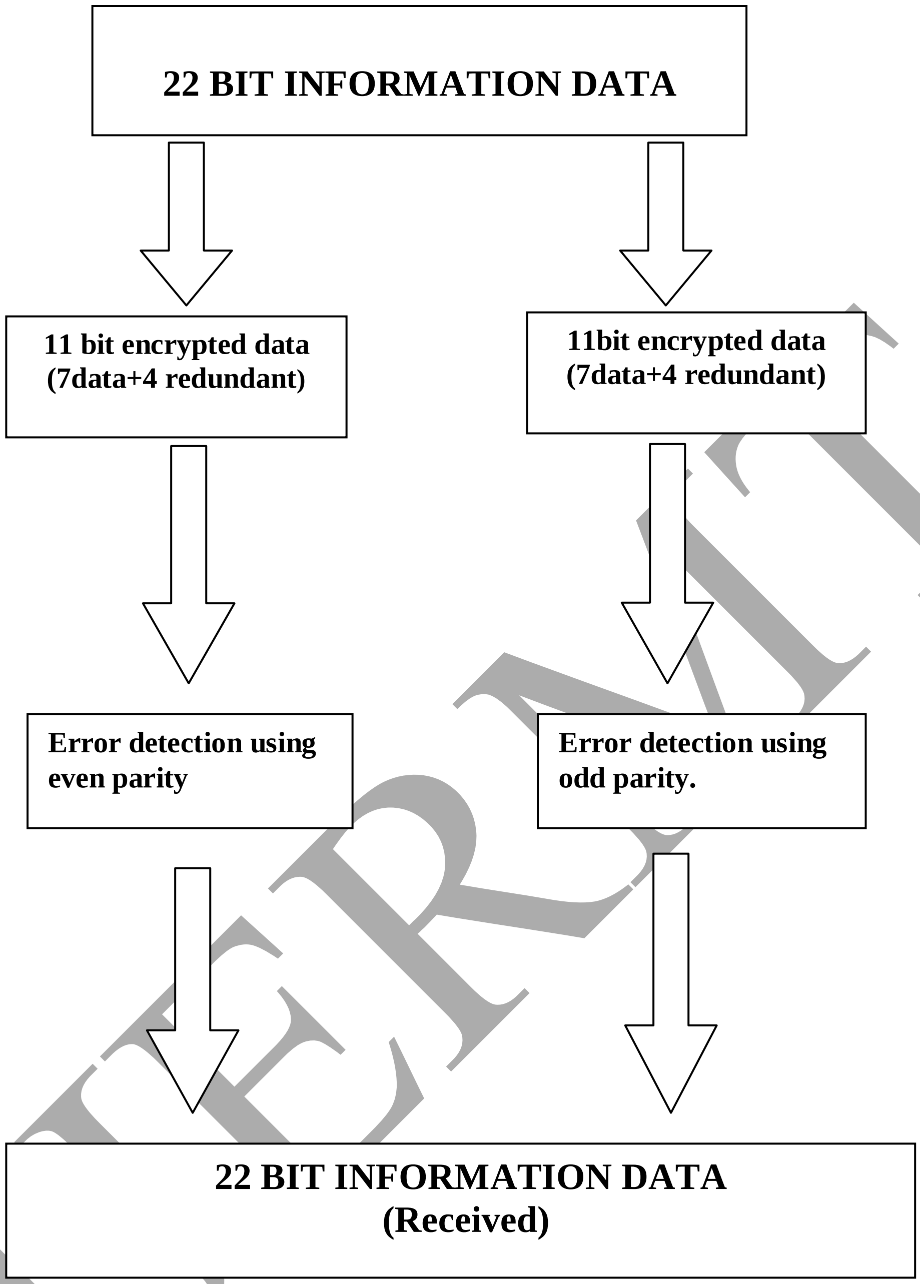
Hamming code method works only on two methods (even parity, odd parity) for generating redundancy bit. In hamming code method for generating the number of redundancy bit given formula is used .The number of redundancy depends on the number of information data bits.

Formula for generating redundancy bit ----
2^ r >= D + r + 1 ------------------------------------------ (1) Here r = number of redundancy bit
D = number of information data bit
Using above formula the number of redundancy bits for 7 bit of input data are 4 bits
(4 for each group).

**2.1 Redundancy**
To detect or correct the error we have to use some extra bits.
These extra bits are called redundancy bits. We add these redundant bits to the information data at the source end and remove at destination end. Presence of redundant bits 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.

## 3. COMMUNICATION WITH EVEN PARITY CHECK METHOD
Communication system needs two main parts one is source for sending data and another is destination for receiving transmitted data. Even parity check method count the number of one`s if number of one`s are even add zero (0) else add one.

### 3.1 Source Section With Even Parity and odd parity Method (Encryption)
In this paper, 14 bit information data string is divided into two groups of 7bit data each. Transmitting 7 bit information data requires minimum 4 redundant bits.
Suppose , these redundancy bits are r(1),r(2),,r(4),r(8 ). To find the value of redundant bits, here even parity check method is used. The value of redundant bits can be found by XORING of different location of information data for different redundant bits.
The property of XOR gate is that if number of one`s are even in input it shows the output zero else it shows output one. Using this property we can easily find the value for a particular redundant bit.
For odd parity bits the even parity bits are just reversed and the logic is same as that of even parity.


22 BIT INFORMATION DATA
11 bit encrypted data (7data+4 redundant)
11bit encrypted data (7data+4 redundant)
Error detection using even parity
Error detection using odd parity.
22 BIT INFORMATION DATA (Received)

**DATA AT RECEIVER END**

Suppose, 7 bit information data is “110100”. Before transmission of information data bits we need to 4 redundant bit. Calculation for redundancy bit r(1) , By XORING input bit address given below.

r (1) = 1, 2,4,5,7

r(2) = 1,3,4,6,7

r(4) = 2,3,4
r(8) = 5,6,7
For calculating the redundant bits r(1), r(2),r(4),r(8) count the number of one`s at the given above locations.
To calculate r(1) using even parity method, check locations 2,4,5,7 in data "110100" the number of 1's at these locations are odd . So the value of r (1) = 1. Similarly to calculate r(2) check bit positions 1,3,4,6,7 the number of 1's are odd so r(2) =0.
In this way r (2) and r(8) are calculated. r (2) =0 and r(8)=0.
Therefore 11 bit encrypted data using even parity is "11001001101".
Remaining 11 bit encrypted data of second group using odd parity is "11011000110".
Hence total 22bit encrypted data at sender end is "1100100110111011000110".

**3.2 Destination Section With Even and odd Parity Method (Detection and correction)**

The Received data at receiver end is given below

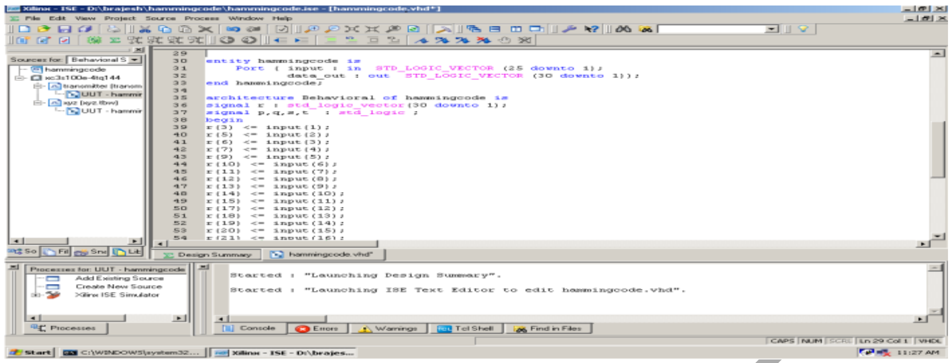
| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 1 |
| **D(7)** | **D(6)** | **D(5)** | **r(8)** | **D(4)** | **D(3)** | **D(2)** | **r(4)** | **D(1)** | **r(2)** | **r(1)** |
| **11** | **10** | **9** | **8** | **7** | **6** | **5** | **4** | **3** | **2** | **1** |

Suppose due to noise present in the channel D(4) is in Error so the corrupted data is shown below

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 0 | 0 | **0** | 0 | 0 | 1 | 1 | 0 | 1 |
| **D(7)** | **D(6)** | **D(5)** | **r(8)** | **D(4)** | **D(3)** | **D(2)** | **r(4)** | **D(1)** | **r(2)** | **r(1)** |
| **11** | **10** | **9** | **8** | **7** | **6** | **5** | **4** | **3** | **2** | **1** |

**Error detection using even parity**
The error is detected by Xor logic in the following manner.
Process is same as done at sender end r(1) = r(1) xor D(1) xor D(2) xor D(4) xor D(5) xor D(7)
r(1) = 1 xor 1 xor 0 xor 0 xor 0 xor 1 = **1**
r(2) = r(2) xor D(1) xor D(3) xorD(4) xor D(6) xor D(7)
r(2) = 0 xor 1 xor 0 xor 0 xor 1 xor 1= **1**
r(4) = r(4) xor D(2) xor D(3) xor D(4)
r(4) = 1 xor 0 xor 0 xor 0 = **1**
r(8) = r(8) xor D(5) xor D(6) xor D(7)
r(8) = 0 xor 0 xor(1) xor(1) =**0**
The parity bit r(2) is as calculated above is coming as '1' whereas in original data it is '0' therefore error is detected in the received data and the position of error bit is r(8)r(4)r(2)r(1) => 1110 => 7th bit in received data.
Hence position of error bit is detected and is corrected.
**Error detection using odd parity**
In odd parity method count the number of one`s in a given string, if number of one`s are even add zero else add one to encrypt information data.
Remaining procedure for error detection using odd parity for second group is same as above.

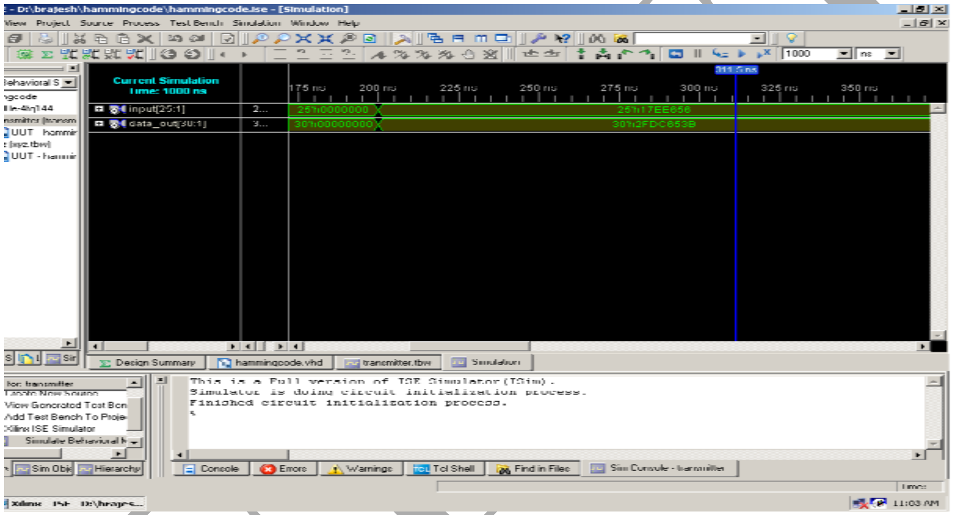
**VHDL code for even parity and odd parity check method at source**



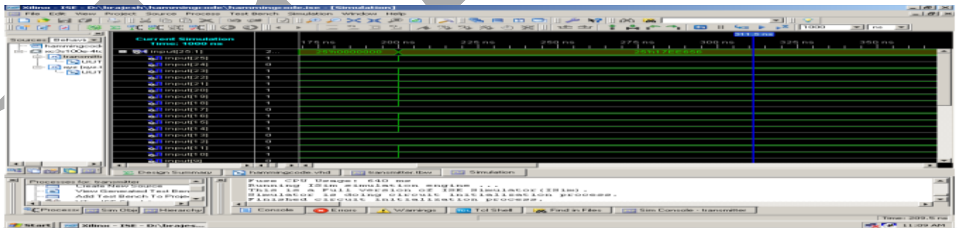
```
entity hammingcode is
    Port ( input : in  STD_LOGIC_VECTOR (25 downto 1);
           data_out : out  STD_LOGIC_VECTOR (30 downto 1));
end hammingcode;

architecture Behavioral of hammingcode is
signal r : std_logic_vector(30 downto 1);
signal p,q,s,t  : std_logic ;
begin
r(3)  <=  input(1);
r(5)  <=  input(2);
r(6)  <=  input(3);
r(7)  <=  input(4);
r(9)  <=  input(5);
r(10)  <=  input(6);
r(11)  <=  input(7);
r(12)  <=  input(8);
r(13)  <=  input(9);
r(14)  <=  input(10);
r(15)  <=  input(11);
r(17)  <=  input(12);
r(18)  <=  input(13);
r(19)  <=  input(14);
r(20)  <=  input(15);
r(21)  <=  input(16);
```

```
Started : "Launching Design Summary".
Started : "Launching ISE Text Editor to edit hammingcode.vhd".
```

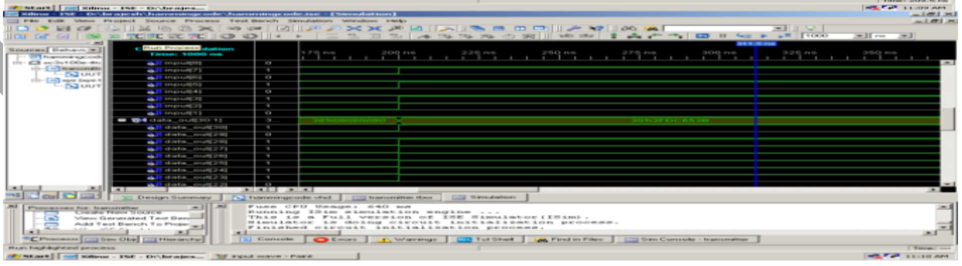
**Xilinx ISE 10.1 window shows Input output wave form in Hexadecimal format at source end**



```
This is a Full version of ISE Simulator(ISim).
Simulator is doing circuit initialization process.
Finished circuit initialization process.
```

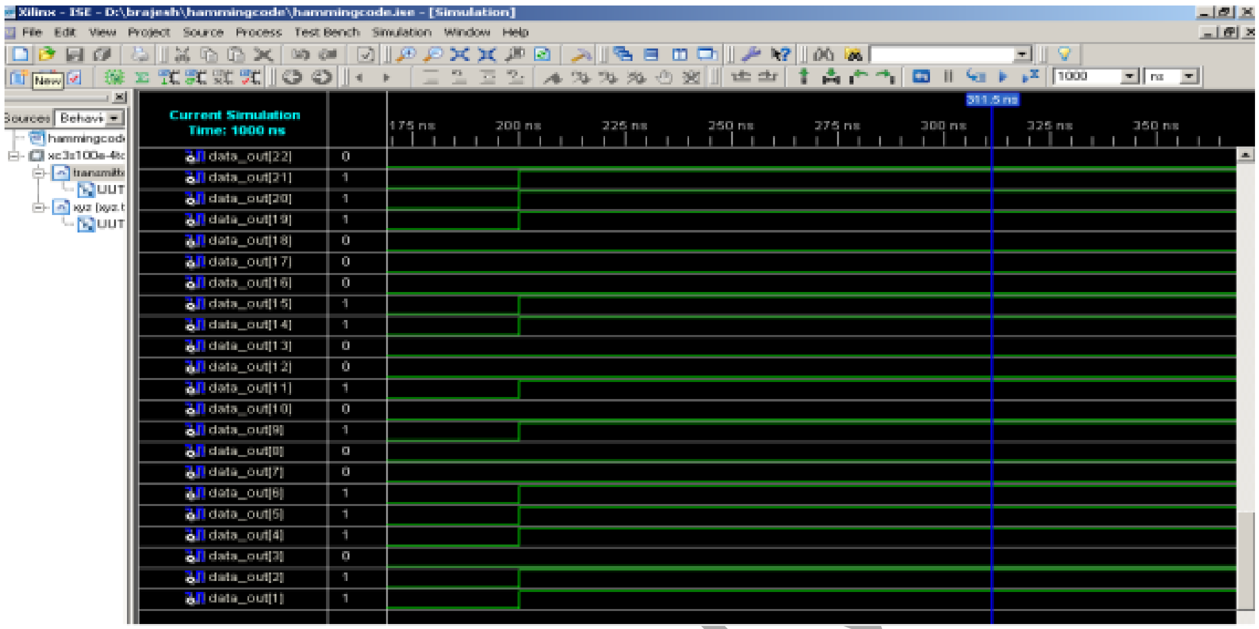


```
Fuse CPU Usage: 640 ms
Running ISim simulation engine ...
This is a Full version of ISE Simulator(ISim).
Simulator is doing circuit initialization process.
Finished circuit initialization process.
```

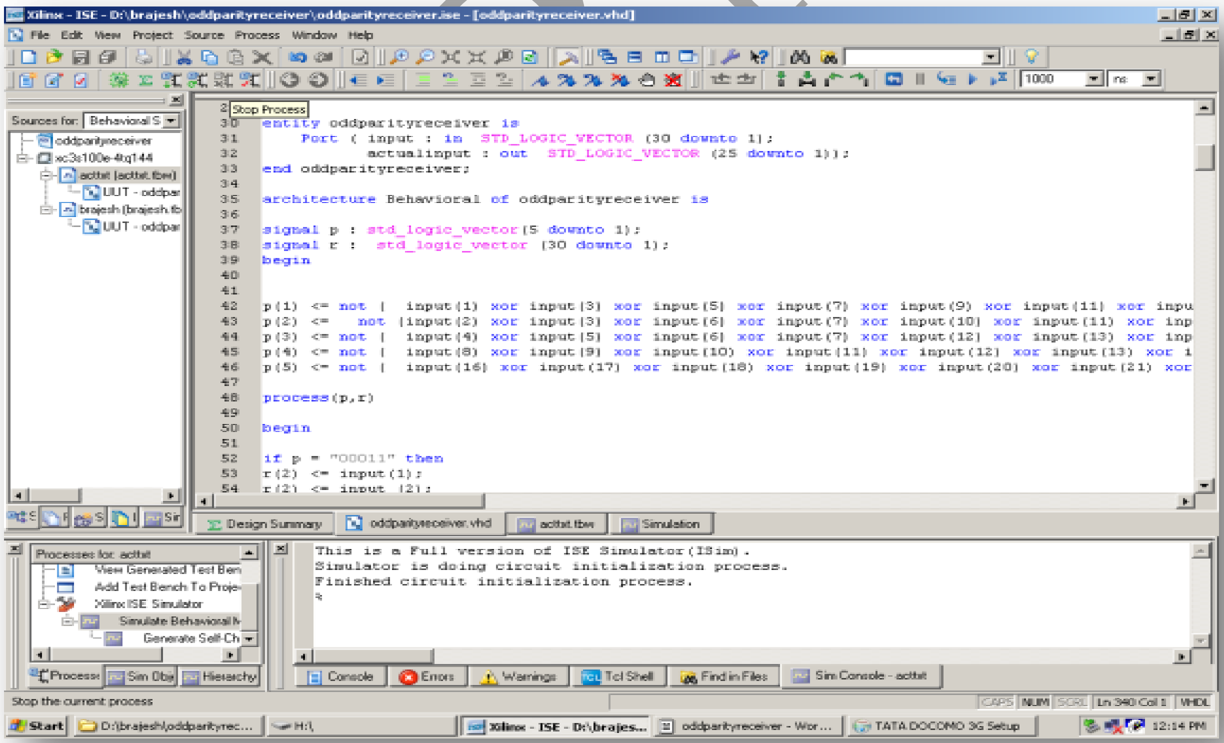


```
Fuse CPU Usage: 640 ms
Running ISim simulation engine ...
This is a Full version of ISE Simulator(ISim).
Simulator is doing circuit initialization process.
Finished circuit initialization process.
```

## Xilinx ISE 10.1 simulation windows shows input output wave form for source end in binary format.


Xilinx - ISE - D:\brajesh\hammingcode\hammingcode.ise - [Simulation]
File Edit View Project Source Process Test Bench Simulation Window Help
Current Simulation Time: 1000 ns
175 ns
200 ns
225 ns
250 ns
275 ns
300 ns
325 ns
350 ns
311.5 ns

```
data_out[22]  0
data_out[21]  1
data_out[20]  1
data_out[19]  1
data_out[18]  0
data_out[17]  0
data_out[16]  0
data_out[15]  1
data_out[14]  1
data_out[13]  0
data_out[12]  0
data_out[11]  1
data_out[10]  0
data_out[9]   1
data_out[8]   0
data_out[7]   0
data_out[6]   1
data_out[5]   1
data_out[4]   1
data_out[3]   0
data_out[2]   1
data_out[1]   1
```

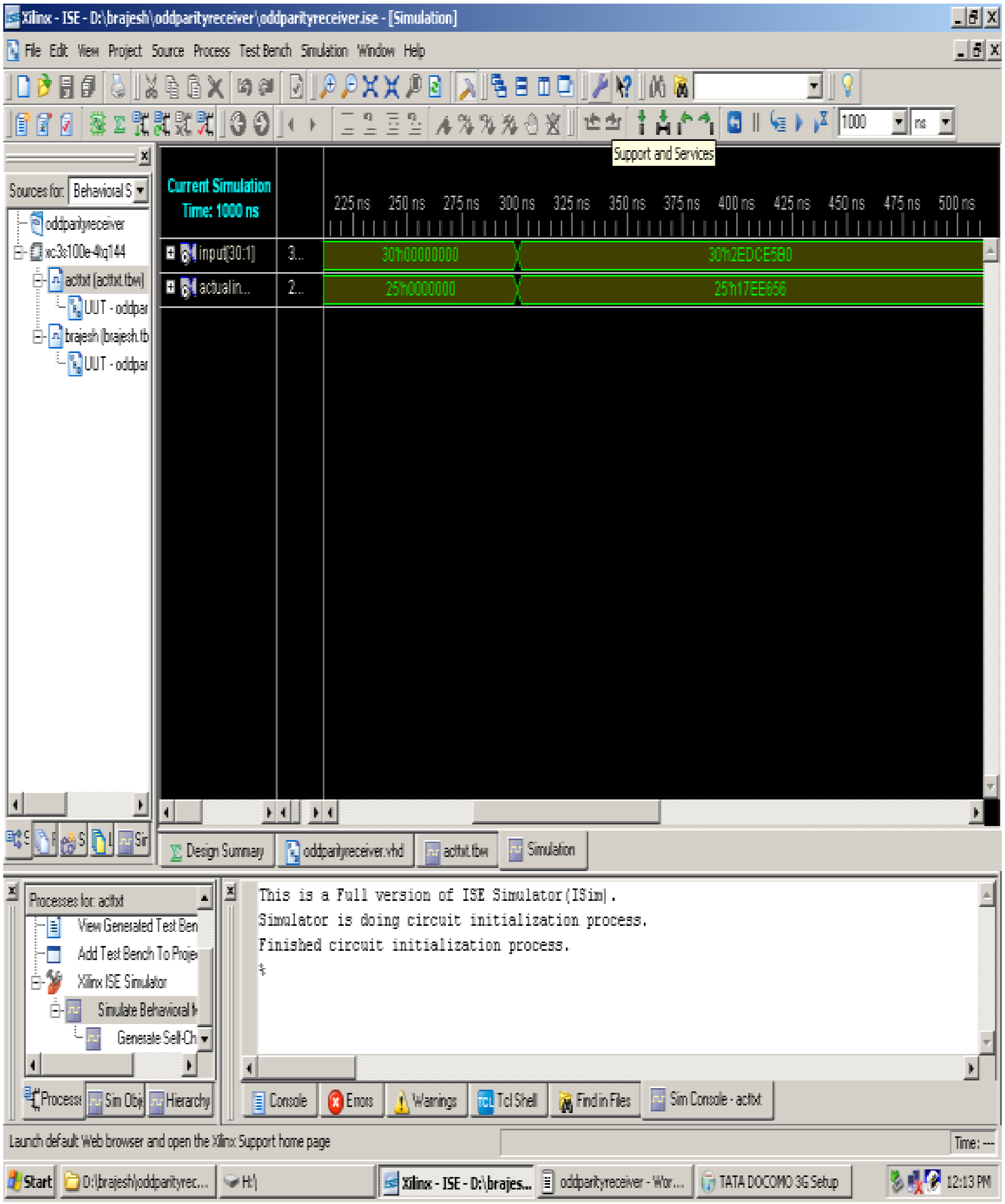

Xilinx - ISE - D:\brajesh\oddparityreceiver\oddparityreceiver.ise - [oddparityreceiver.vhd]
File Edit View Project Source Process Window Help
Stop Process
Design Summary
oddparityreceiver.vhd
actbt.tbw
Simulation
Console
Errors
Warnings
Tcl Shell
Find in Files
Sim Console - actbt

```
entity oddparityreceiver is
    Port ( input : in  STD_LOGIC_VECTOR (30 downto 1);
           actualinput : out  STD_LOGIC_VECTOR (25 downto 1));
end oddparityreceiver;

architecture Behavioral of oddparityreceiver is

signal p : std_logic_vector(5 downto 1);
signal r :  std_logic_vector (30 downto 1);
begin


p(1) <= not (  input(1) xor input(3) xor input(5) xor input(7) xor input(9) xor input(11) xor inpu
p(2) <=   not (input(2) xor input(3) xor input(6) xor input(7) xor input(10) xor input(11) xor inp
p(3) <= not (  input(4) xor input(5) xor input(6) xor input(7) xor input(12) xor input(13) xor inp
p(4) <= not (  input(8) xor input(9) xor input(10) xor input(11) xor input(12) xor input(13) xor i
p(5) <= not (  input(16) xor input(17) xor input(18) xor input(19) xor input(20) xor input(21) xor

process(p,r)

begin

if p = "00011" then
r(2) <= input(1);
r(2) <= input (2);
```

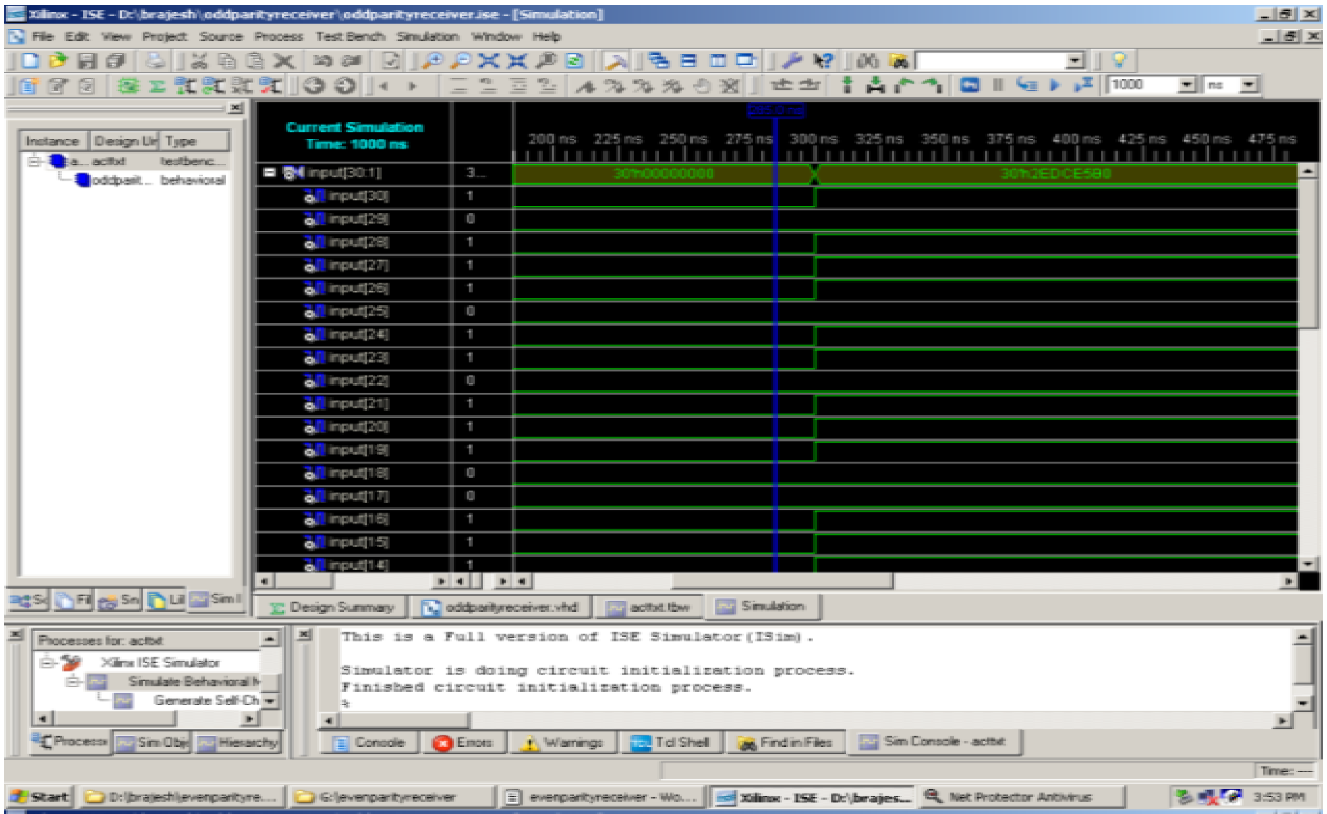
```
This is a Full version of ISE Simulator(ISim).
Simulator is doing circuit initialization process.
Finished circuit initialization process.
%
```

## VHDL code for even parity and odd parity check at destination

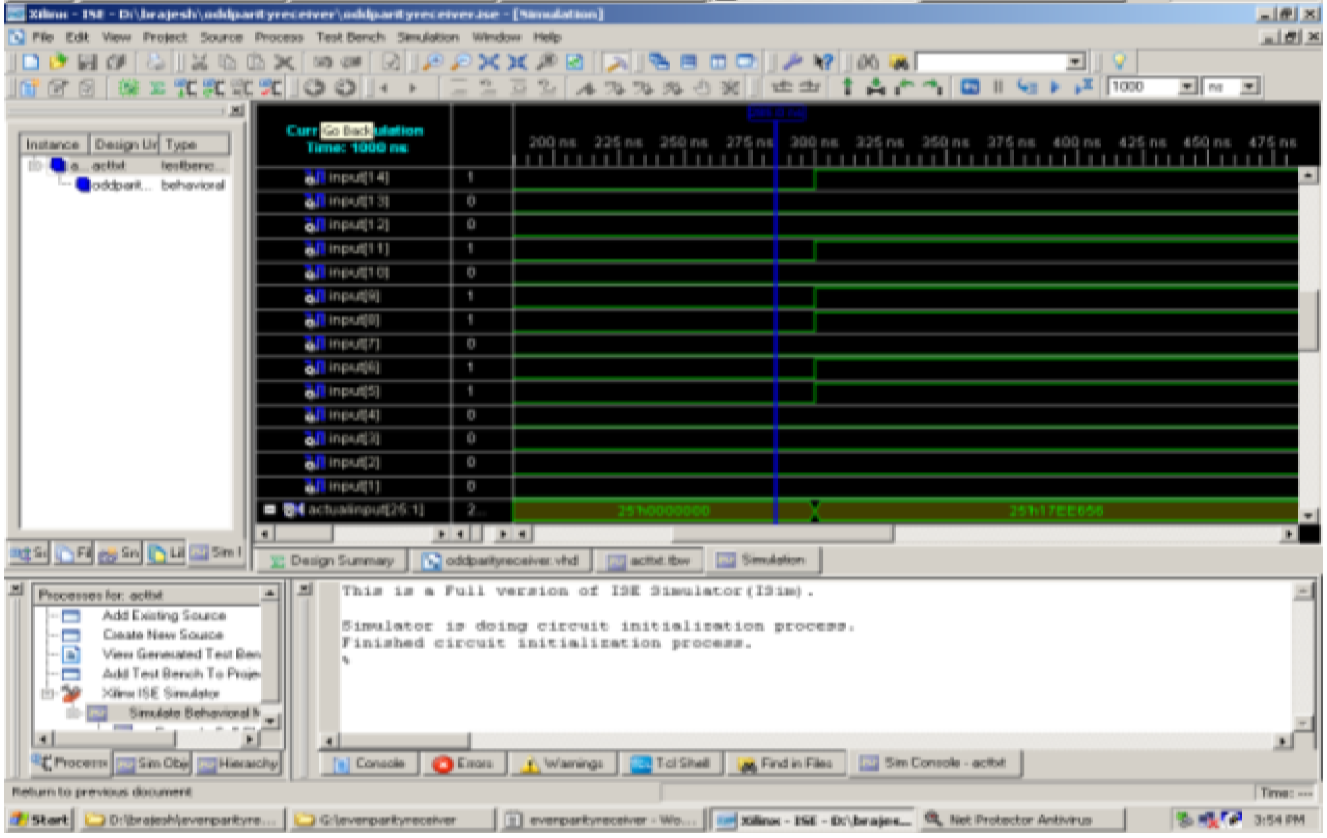
**Xilinx ISE 10.1 simulation window shows the input and output waveform at destination in hexadecimal format**



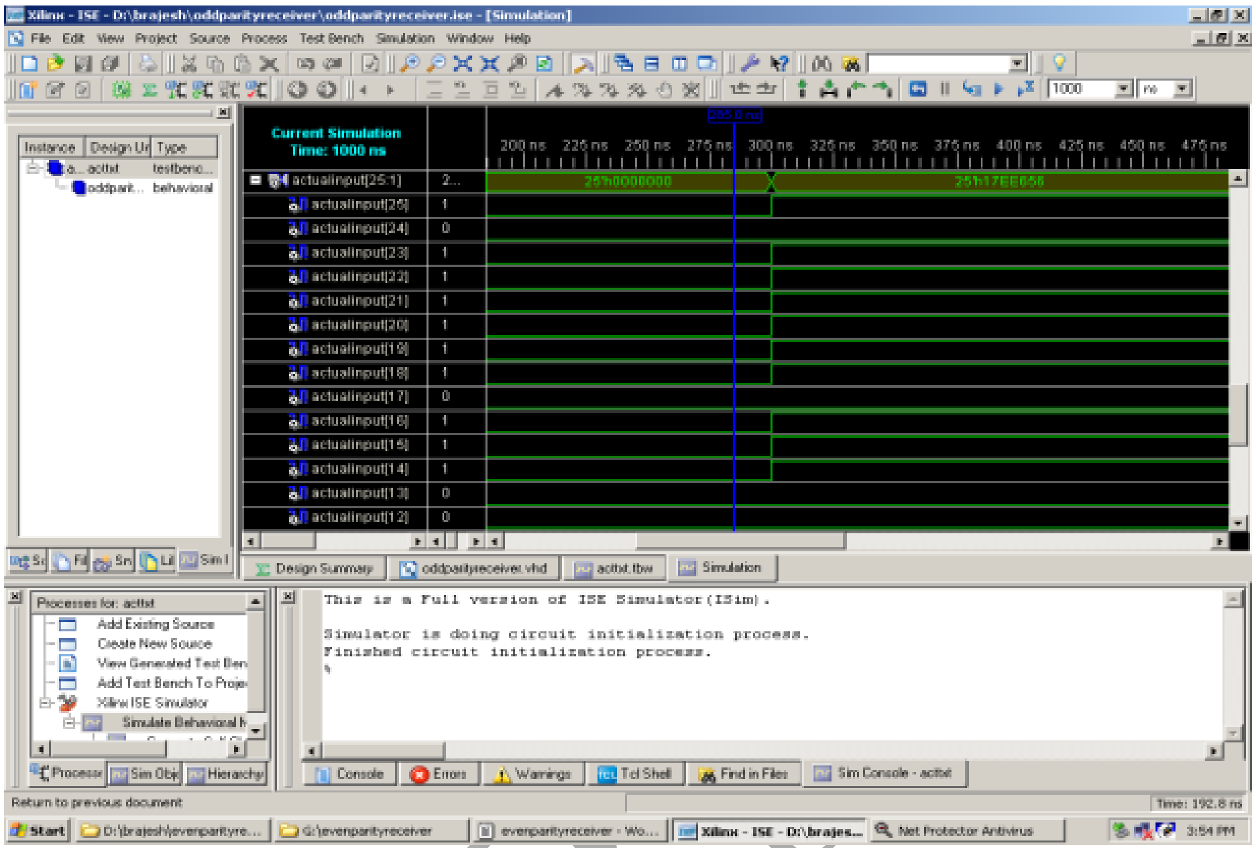
```
This is a Full version of ISE Simulator(ISim).
Simulator is doing circuit initialization process.
Finished circuit initialization process.
%
```



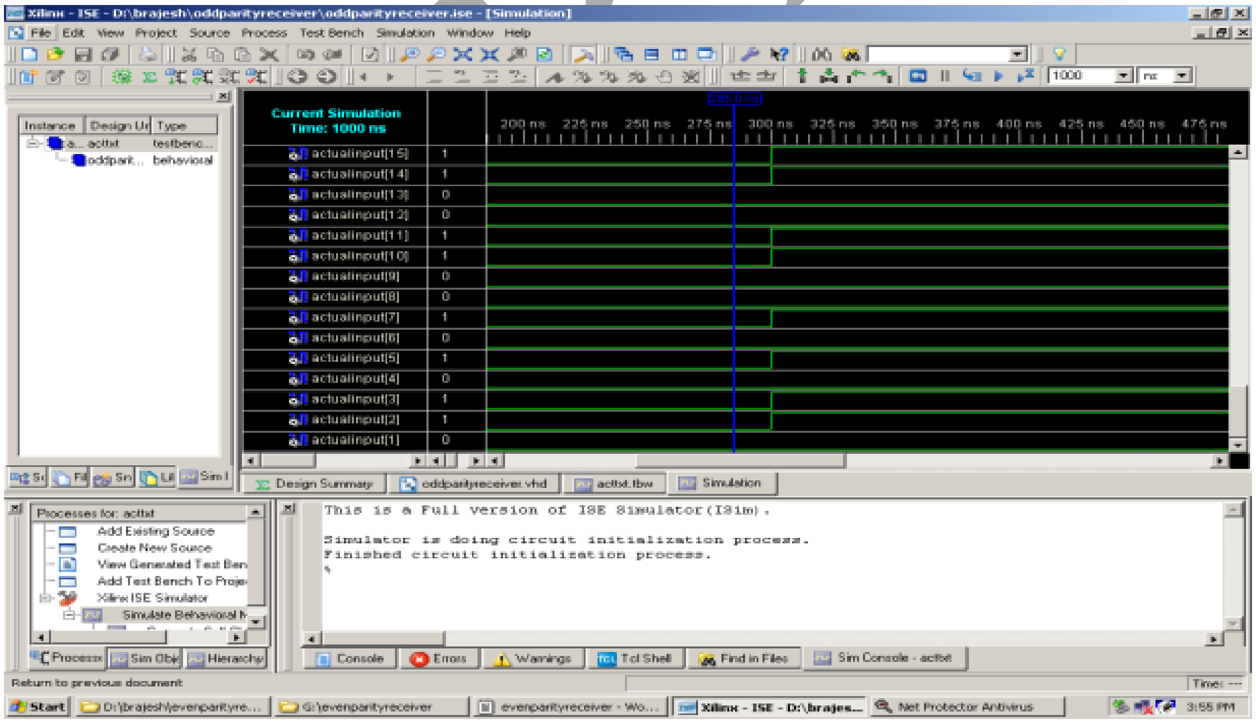
```
This is a Full version of ISE Simulator(ISim).

Simulator is doing circuit initialization process.
Finished circuit initialization process.
```



```
This is a Full version of ISE Simulator(ISim).

Simulator is doing circuit initialization process.
Finished circuit initialization process.
```



```
This is a Full version of ISE Simulator(ISim).

Simulator is doing circuit initialization process.
Finished circuit initialization process.
```



```
This is a Full version of ISE Simulator(ISim).

Simulator is doing circuit initialization process.
Finished circuit initialization process.
```

## 4. APPLICATION

An application of hamming code error detection and correction with even parity check and odd parity check method is that using this method there is no need to transmit data string again by transmitter at source end, if only single bit error is occurred by noisy channel because at destination end receiver can regenerate the original data string which was transmitted by transmitter at source end. 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 (ADSL, xDSL).

## 5. ADVANTAGES

Up to today, at destination end we were using two circuits for generating 22 bit actual information data which we want to transmit from source end. One for finding location of error bit and correcting that error bit, another for decrypt, received error free encrypted data.
Now we can use only one circuit for performing all operation at destination so that delay time could be reduced for regenerating 14 bit actual information data from received 22 bit encrypted corrupt data string. And the complexity of circuit configuration is reduced.

Speed of communication system also depends on the number of frame (combination of number of bit is called frame) that can be transmitted in a second. To increase the speed of communication system increases the number of frame per second or increase the number of bits in a frame. Here we have increased the frame size to increase the number of bits in a single frame.

## 6. CONCLUSION

The overall conclusion of this paper is that, speed of communication system can be increased by using these methodologies; we can transmit more combination of data (more information in a single frame).
The complexity of circuit is also reduced and we can use one IC in place of two IC`s for regenerating actual information data from encrypted corrupt received data at destination.

## 7. REFERENCES

1. Data communication and networking , Behrouz A.Forouzan , 2nd edition Tata McGrawHill publication.
2. Communication Networks, available at: http://www.pragsoft.com/books/CommNetwork.pdf
3. Xilinx ISE 8 Software Manuals and Help: http://www.eng.uwaterloo.ca/~tnaqvi/downloads/DOC/sd192/ISE8_1i_manuals.pdf
4. [Xilinx Training Course Listing, available at http://www.xilinx.com/training/xilinx-training- courses.pdf
5. ISE 10.1 Quick Start Tutorial, available at http://www.xilinx.com/itp/xilinx10/books/docs/qst/qst.pd f
6. VHDL (VHSIC hardware description language): http://en.wikipedia.org/wiki/VHDL
7. VHDL Tutorial, available at http://www.vhdl- online.de/tutorial/
   VHDL Designer' Guide, available at http://www.doulos.com/knowhow/vhdl_designers_guide/
8. A VHDL Primer, J. Bhasker , 3 rd edition PHI publication.
9. Digital Logic Design with VHDL , Stephen Brown & Zvonko Vranesic , 2 nd edition TMH publication