ISSN: 2348-4098
VOLUE 02 ISSUE 06 JULY 2014

IMPROVEMENT OF THE ORTHOGONAL CODE CONVOLUTION CAPABILITIES USING FPGA IMPLEMENTATION

[1]ANITA YADAV, [2]SHEETAL GANGWAR

[1, 2]Masters of Technology Scholar, Department of Electronics & Communication Engineering, Jayoti Vidyapeeth Women's University, Jaipur- INDIA

ABSTRACT

In this paper, FPGA implementation of orthogonal code convolution is presented by employing Xilinx and Modelsim softwares. In digital communication system, convolution coding is preferred for the channel coding as it facilitates a better error correction as comparison to block coding which does not require memory. Among other techniques such as Cyclic Redundancy and Solomon Codes; orthogonal coding is one of the codes which can detect errors and correct corrupted data in an efficient way. When data is stored, compressed, or communicated through a media such as cable or air, sources of noise and other parameters such as EMI, crosstalk, and distance can considerably affect the reliability of these data. Error detection and correction techniques are therefore required. Orthogonal Code is one of the codes that can detect errors and correct corrupted data. An n-bit orthogonal code has n/2 1s and n/2 0s. In a previous work these properties have been exploited to detect and correct errors. The technique was implemented experimentally using Field Programmable Gate Arrays (FPGA). The results show that the proposed technique improves the detection capabilities of the orthogonal code by approximately 50%, resulting in 99.9% error detection, and corrects as predicted up to (n/4-1) bits of error in the received impaired code with bandwidth efficiency. The transmitter does not have to send the parity bit since the parity bit is known to be always zero. Therefore, if there is a transmission error, the receiver will be able to detect it by generating a parity bit at the receiving end.

1. INTRODUCTION

Information and communication technology has brought enormous changes to our life and turned out to be one of the basic building blocks of modern society. Day by day, there is an increasing demand of network capacity due to the use of internet and real time transmission of voice and picture. To

ISSN: 2348-4098
VOLUE 02 ISSUE 06 JULY 2014

fulfill these requirements data transmission at high bit rates is essential for various aspects such as video, high-quality audio and mobile integrated service digital network (ISDN).

However, the data transmitted at high bit rates over mobile radio channels, leads to inter symbol interference (ISI). The significant factors which cause the reliability of digital data communication are the transmission medium i.e. cable or air, sources of noise and some others like electromagnetic interface, crosstalk and distance. To overcome this problem, error correction coding is a solution for the best possible communication. The main advantage of using coding is the efficiency of the channels use becomes higher as comparison to the case when code is not used. Therefore, error detection and correction techniques are needed which can detect errors such as the Cyclic Redundancy Check and others which can detect as well as correct errors such as Solomon Codes [1-3]. Our objective in this paper is to enhance the error control capabilities of orthogonal codes by means of Field Programmable Gate Array (FPGA) implementation. The CRC generation has many advantages over simple sum techniques or parity check. This coding is binary valued and with equal number of 1‟s and 0‟s. All orthogonal codes can generate zero parity bits as n-bit orthogonal code has n/2 1‟s and n/2 0‟s. In simple there are n/2 positions where 1‟s and 0‟s differ and hence, each antipodal code can also generate a zero parity bit [5]. It is noted that with this method, the transmitter does not have to send the parity bit since the parity bit is known to be always zero.

Therefore, if there is a transmission error, the receiver will be able to detect it by generating a parity bit at the receiving end. In this paper, the FPGA implementation of orthogonal code convolution is presented by employing Xilinx and Modelsim softwares; in section second and third, the theory of orthogonal coding and design approach are presented. The simulated results and analysis are discussed in section fourth. Finally, section fifth concludes the paper.

2. ORTHOGONAL CODES

Orthogonal codes are consists of equal number of 1‟s and 0‟s e.g. n-bit orthogonal code consist n/2 1‟s and n/2 0‟. Meaning, there are n/2 positions where 1‟s and 0‟s differ. In this way, all

ISSN: 2348-4098
VOLUE 02 ISSUE 06 JULY 2014

orthogonal codes generate zero parity bits. An illustration of 16-bit orthogonal code is shown in figure 1. Zero parity bits. The concept is illustrated by means of an 8- bit orthogonal code as shown in Fig.1. It has 8-orthogonal codes and 8-antipodal codes for a total of 16-biorthogonal codes. Antipodal codes are just the inverse of orthogonal codes; they are also orthogonal among themselves.

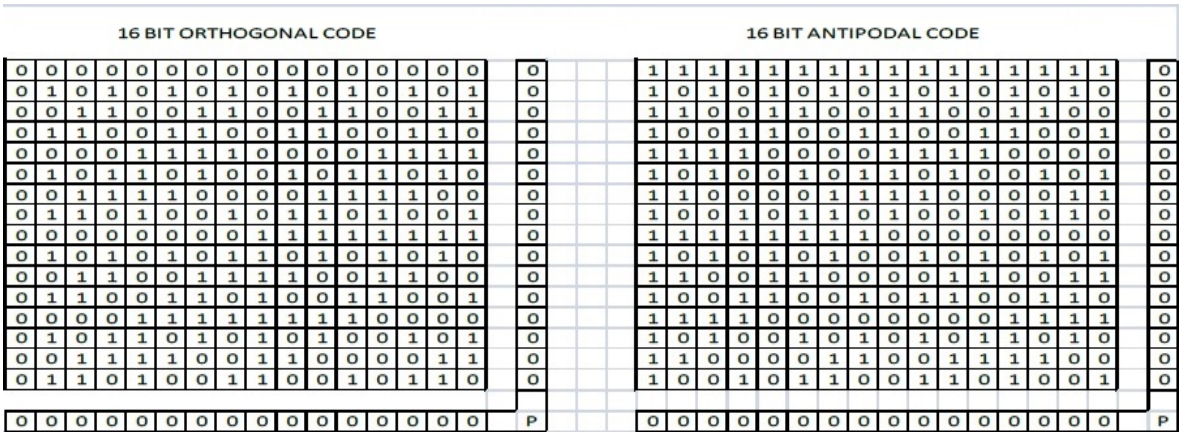


Figure 1: A 16-bit orthogonal code has 16 orthogonal codes and 16-antipodal codes for a total of 32 bi-orthogonal codes

It is comprised of 16-orthogonal codes and 16- antipodal codes (just the inverse of orthogonal codes) for a total of 32 bi-orthogonal codes as depicted in figure 2. The advantage with this approach is that transmitter does not need to send the parity bit as parity bit is known to be always zero. In this way, if error exists during, the receiver can detect by generating a parity bit at the receiving end. In orthogonal coding, a k-bit data set is mapped into a unique n-bit before transmission. Here, we have considered a 5-bit data set which is can be represented by a unique 16-bit orthogonal code and transmitted without the parity bit. After receiving the data, it is decoded based on code correlation by setting a threshold midway between two orthogonal codes. The threshold midway is represented as

$$dth = n/4$$

Where n is the code length and dth is the threshold midway between two orthogonal codes. According to above equation, for 16-bit orthogonal coding, threshold midway is 4 between two orthogonal codes. This approach offers a decision process, where the incoming

ISSN: 2348-4098
VOLUE 02 ISSUE 06 JULY 2014

impaired orthogonal code is examined for correlation with the neighbouring codes for a possible match. It is noted that the acceptance criterion for a valid code is that an n-bit comparison must yield a good autocorrelation value; otherwise, a false detection will occur. Where R(x, y) is the auto correlation function, n is the code length, dth is the threshold defined in (1). Since the threshold (dth) is in between two valid codes, an additional 1-bit offset is added to (2) for reliable detection. The average number of errors that can be corrected by means of this process can be estimated by combining (1) and (2), yielding,(3).

In (3), t is the number of errors that can be corrected by means of an n-bit orthogonal code. For example, a single error-correcting orthogonal code can be constructed by means of an 8-bit orthogonal code (n = 8). Similarly, a three-error correcting orthogonal code can be constructed by means of a 16-bit orthogonal code (n = 16), and so on. Table-1 below shows a few orthogonal codes and the corresponding error correcting capabilities:

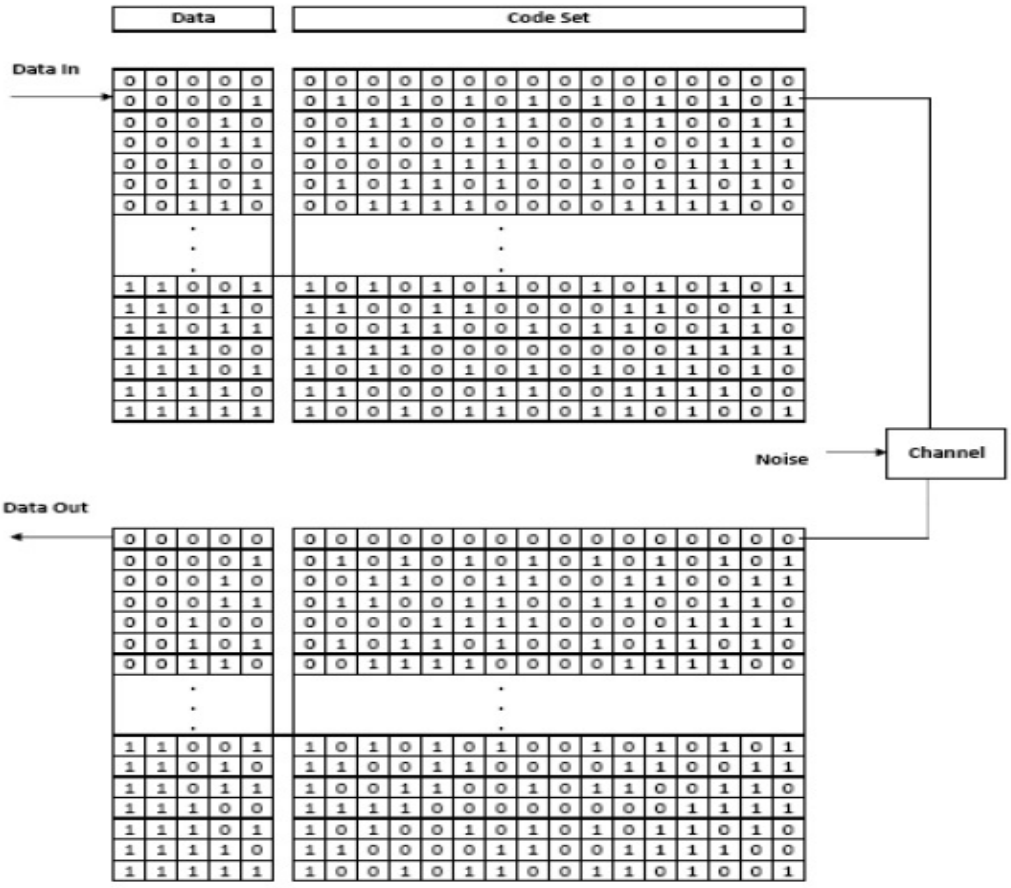


Figure 2: Encoding and Decoding Process

ISSN: 2348-4098
VOLUE 02 ISSUE 06 JULY 2014

TABLE I: Orthogonal Codes and the Corresponding Chip Error Control Capabilities

n	t
8	1
16	3
32	7
64	15

3. DESIGN APPROACH

Our design approach is based on the comparison between the received code and all the orthogonal code combinations stored in a look up table; which has two major components such as a transmitter and a receiver. The first component (transmitter) consists of two blocks such as encoder and shift register which is shown in figure 3.

3.1 DESIGN METHODOLOGY

Since there is an equal number of 1's and 0's, each orthogonal code will generate a zero parity bit. If the data has been corrupted during the transmission the receiver can detect errors by generating the parity bit for the received code and if it is not zero then the data is corrupted. However the parity bit doesn't change for an even number of errors, hence the receiver can only detect errors 2^N /2 combinations of the received code. Therefore detection percentage is 50%. Our approach is not to use the parity generation method to detect the errors, but a simple technique based on the comparison between the received code and all the orthogonal code combinations stored in a look up table. The technique which involves a transmitter and receiver is described below.

3.2 TRANSMITTER

The transmitter includes two blocks: an encoder and a shift register. The encoder encodes a k-bit data set to n=2^k-1 bits of the orthogonal code and the shift register transforms this code to a serial data in order to be transmitted as shown in Fig.3. For example, 5-bit data is encoded to 16-bit orthogonal code according to the lookup table shown in Fig.2. The generated orthogonal code is then transmitted serially using a shift register with the rising edge of the clock.

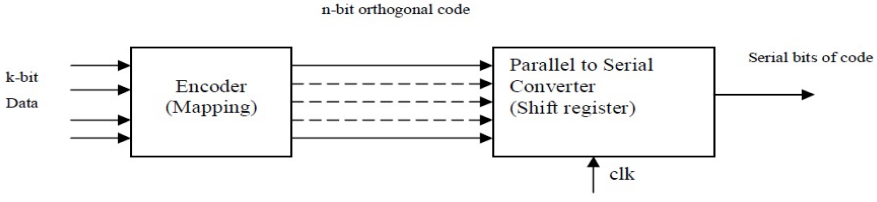


Fig. 3. Block diagram of the transmitter.

ISSN: 2348-4098
VOLUE 02 ISSUE 06 JULY 2014

3.3 RECEIVER

The received code is processed through the sequential steps, as shown in Fig.4. The incoming serial bits are converted into n-bit parallel codes.This is done by counting the number of ones in the signal resulting from 'XOR' operation between the received code and each combination of the orthogonal codes in the lookup table. A counter is used to count the number of ones in the resulting n-bit signal and also searches for the minimum count. However a value rather than zero shows an error in the received code. The orthogonal code in the lookup table which is associated with the minimum count is the closest match for the corrupted received code. The matched orthogonal code in the lookup table is the corrected code, which is then decoded to k-bit data. The receiver is able to correct up to (n/4)-1 bits in the received impaired code. However, if the minimum count is associated with more than one combination of orthogonal code then a signal, REQ, goes high.

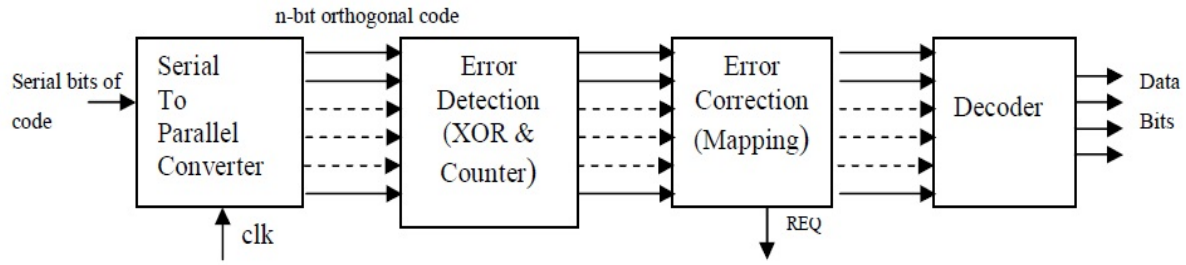


Fig. 4. Block diagram of the receiver.

4. CODING TECHNIQUE & RESULTS

4.1 VHDL CODE FOR TRANSMITTER

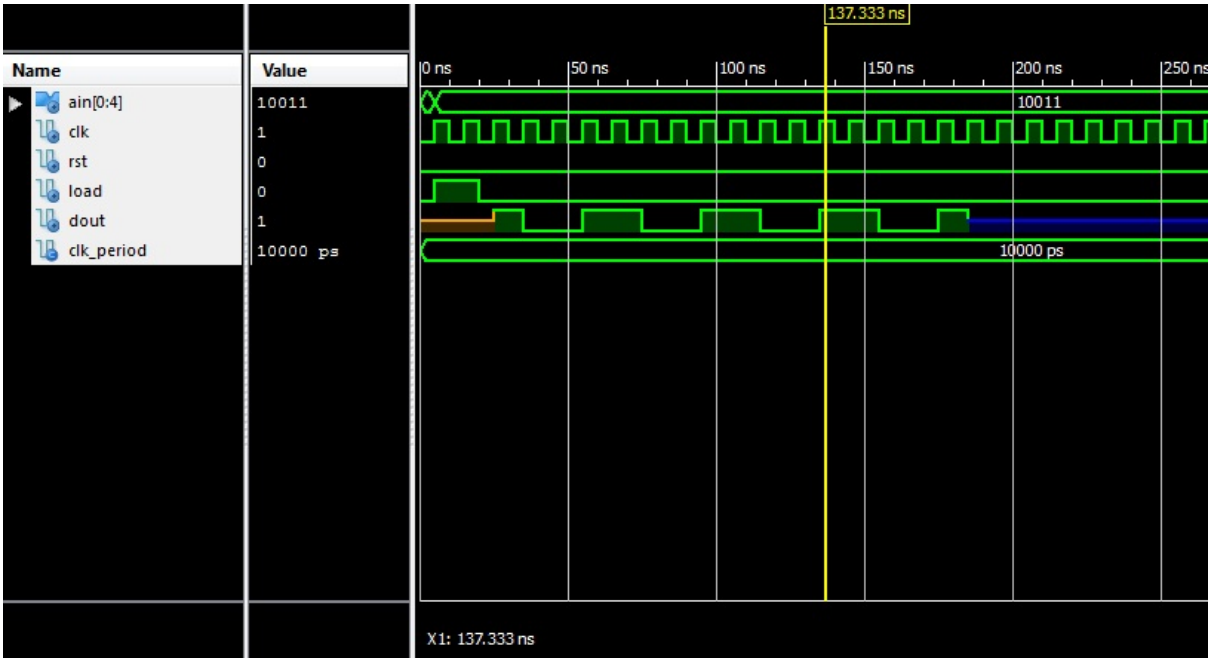
```
LIBRARY IEEE;

USE IEEE.STD_LOGIC_1164.ALL;

ENTITY TRANSMITTER16 IS

PORT(AIN:IN STD_LOGIC_VECTOR(0 TO 4);

                CLK,RST,LOAD:IN STD_LOGIC;
```

ISSN: 2348-4098
VOLUE 02 ISSUE 06 JULY 2014

```
                DOUT:OUT STD_LOGIC);
        END TRANSMITTER16;
ARCHITECTURE BEHAV OF TRANSMITTER16 IS
SIGNAL DATA_OUT:STD_LOGIC_VECTOR(0 TO 15);
COMPONENT BIT16_ENCODER
        PORT(AIN:IN STD_LOGIC_VECTOR(0 TO 4);
                DATA_OUT:OUT STD_LOGIC_VECTOR(0 TO 15));
END COMPONENT;
COMPONENT PISO16
        PORT(B:IN STD_LOGIC_VECTOR(0 TO 15);
                CLK,RST,LOAD:IN STD_LOGIC;
          DOUT:OUT STD_LOGIC);
END COMPONENT;
BEGIN
CHIP1:BIT16_ENCODER
PORT MAP(AIN,DATA_OUT);
CHIP2:PISO16
PORT MAP(DATA_OUT,CLK,RST,LOAD,DOUT);
END BEHAV;
```

ISSN: 2348-4098
VOLUE 02 ISSUE 06 JULY 2014

4.1.1 SIMILATION RESULT OF TRANSMITTER



4.2 ENCODER (MAPPING)

An encoder is a device, circuit, transducer, software program, algorithm or person that converts information from one format or code to another, for the purposes of standardization, speed, secrecy, security or compressions.

The conversion of input data into orthogonal coder is given below:-

5 bit Input data 16 bit Orthogonal Code

Selected input data =>Output data of Encoder

"00000" => "0000000000000000";

"00001" => "0101010101010101";

"00010" => "0011001100110011";

"00011" => "0110011001100110";

"00100" => "0000111100001111";

"00101" => "0101101001011010";

"00110" => "0011110000111100";

"00111" => "0110100101101001";

"01000" => "0000000011111111";

ISSN: 2348-4098
VOLUE 02 ISSUE 06 JULY 2014

"01001" => "0101010110101010";

"01010" => "0011001111001100";

"01011" => "0110011010011001";

"01100" => "0000111111110000";

"01101" => "0101101010100101";

"01110" => "0011110011000011";

"01111" => "0110100110010110";

"10000" => "1111111111111111";

"10001" => "1010101010101010";

"10010" => "1100110011001100";

"10011" => "1001100110011001";

"10100" => "1111000011110000";

"10101" => "1010010110100101";

"10110" => "1100001111000011";

"10111" => "1001011010010110";

"11000" => "1111111100000000";

"11001" => "1010101001010101";

"11010" => "1100110000110011";

"11011" => "1001100101100110";

"11100" => "1111000000001111";

"11101" => "1010010101011010";

"11110" => "1100001100111100";

"11111" => "1001011001101001";

ISSN: 2348-4098
VOLUE 02 ISSUE 06 JULY 2014

VHDL CODE FOR ENCODER

```
LIBRARY IEEE;

USE IEEE.STD_LOGIC_1164.ALL;

ENTITY BIT16_ENCODER IS

PORT(AIN:IN STD_LOGIC_VECTOR(0 TO 4);

                DATA_OUT:OUT STD_LOGIC_VECTOR(0 TO 15));

                END BIT16_ENCODER;

ARCHITECTURE BEHAV OF BIT16_ENCODER IS

BEGIN

PROCESS(AIN)

        BEGIN

CASE AIN IS

            WHEN "00000" => DATA_OUT <= "0000000000000000";

              WHEN "00001" => DATA_OUT <= "0101010101010101";

              WHEN "00010" => DATA_OUT <= "0011001100110011";

              WHEN "00011" => DATA_OUT <= "0110011001100110";

              WHEN "00100" => DATA_OUT <= "0000111100001111";

              WHEN "00101" => DATA_OUT <= "0101101001011010";

              WHEN "00110" => DATA_OUT <= "0011110000111100";

              WHEN "00111" => DATA_OUT <= "0110100101101001";

              WHEN "01000" => DATA_OUT <= "0000000011111111";

              WHEN "01001" => DATA_OUT <= "0101010110101010";
```

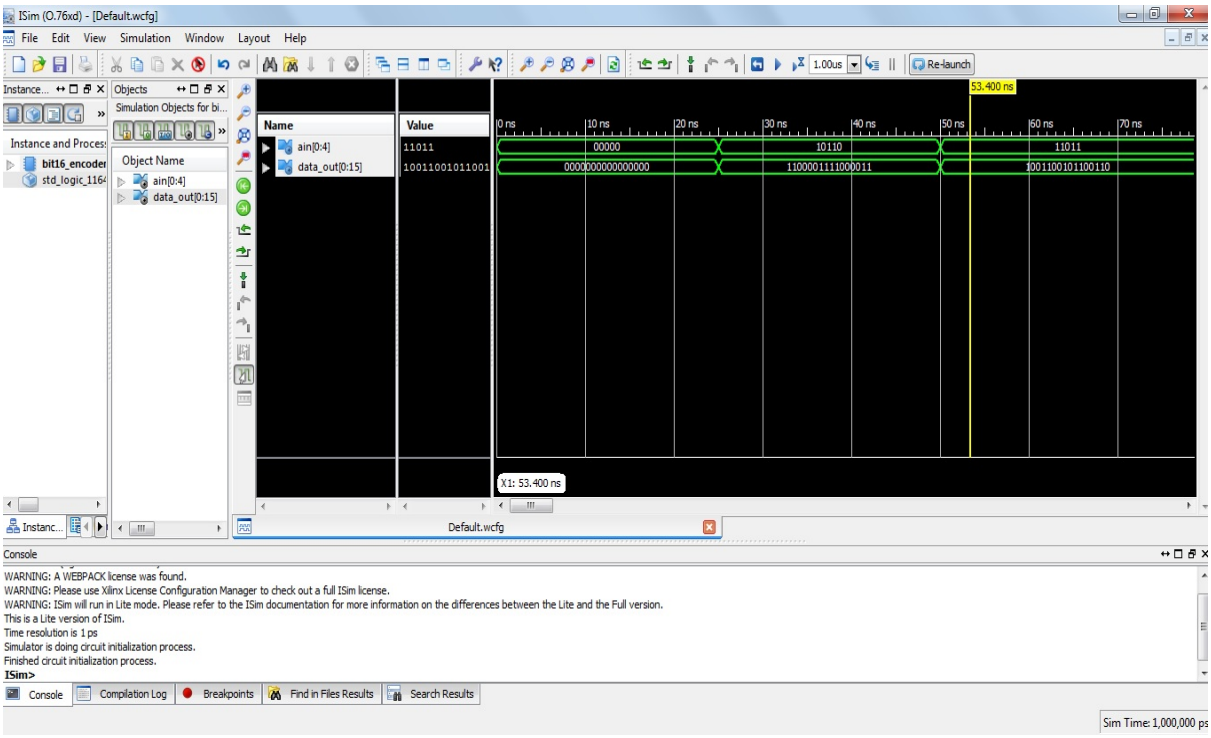
ISSN: 2348-4098
VOLUE 02 ISSUE 06 JULY 2014

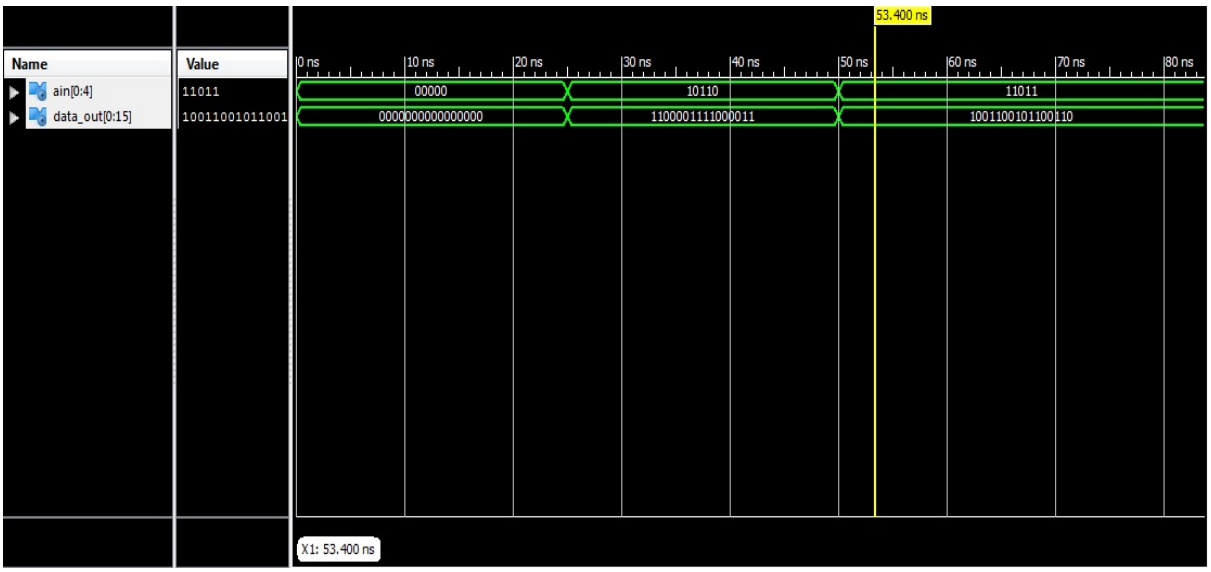
```
WHEN "01010" => DATA_OUT <= "0011001111001100";

WHEN "01011" => DATA_OUT <= "0110011010011001";

WHEN "01100" => DATA_OUT <= "0000111111110000";

WHEN "01101" => DATA_OUT <= "0101101010100101";

WHEN "01110" => DATA_OUT <= "0011110011000011";

WHEN "01111" => DATA_OUT <= "0110100110010110";

WHEN "10000" => DATA_OUT <= "1111111111111111";

WHEN "10001" => DATA_OUT <= "1010101010101010";

WHEN "10010" => DATA_OUT <= "1100110011001100";

WHEN "10011" => DATA_OUT <= "1001100110011001";

WHEN "10100" => DATA_OUT <= "1111000011110000";

WHEN "10101" => DATA_OUT <= "1010010110100101";

WHEN "10110" => DATA_OUT <= "1100001111000011";

WHEN "10111" => DATA_OUT <= "1001011010010110";

WHEN "11000" => DATA_OUT <= "1111111100000000";

WHEN "11001" => DATA_OUT <= "1010101001010101";

WHEN "11010" => DATA_OUT <= "1100110000110011";

WHEN "11011" => DATA_OUT <= "1001100101100110";

WHEN "11100" => DATA_OUT <= "1111000000001111";

WHEN "11101" => DATA_OUT <= "1010010101011010";

WHEN "11110" => DATA_OUT <= "1100001100111100";

WHEN "11111" => DATA_OUT <= "1001011001101001";
```

ISSN: 2348-4098
VOLUE 02 ISSUE 06 JULY 2014

```
                WHEN OTHERS  => DATA_OUT <= "----------------";

END CASE;

        END PROCESS;

END BEHAV;
```

SIMILATION RESULT OF ENCODER





ISSN: 2348-4098
VOLUE 02 ISSUE 06 JULY 2014

VHDL CODE FOR RECEIVER

```
LIBRARY IEEE;

USE IEEE.STD_LOGIC_1164.ALL;

ENTITY RECEIVER16 IS

PORT(DIN:IN STD_LOGIC;

                CLK,RST:IN STD_LOGIC;

                AOUT:OUT STD_LOGIC_VECTOR(0 TO 4));

END RECEIVER16;

ARCHITECTURE BEHAV OF RECEIVER16 IS

SIGNAL Z,ZO:STD_LOGIC_VECTOR(0 TO 15);

COMPONENT SIPO16

        PORT(DIN:IN STD_LOGIC;

                CLK,RST:IN STD_LOGIC;

                Z:OUT STD_LOGIC_VECTOR(0 TO 15));

END COMPONENT;

COMPONENT ERRORDET_CORR16

        PORT(Z:IN STD_LOGIC_VECTOR(0 TO 15);

                ZO:OUT STD_LOGIC_VECTOR(0 TO 15));

END COMPONENT;

COMPONENT DECODER16

PORT(ZO:IN STD_LOGIC_VECTOR(0 TO 15);

                AOUT:OUT STD_LOGIC_VECTOR(0 TO 4));
```

ISSN: 2348-4098
VOLUE 02 ISSUE 06 JULY 2014

END COMPONENT;

BEGIN

CHIP1:SIPO16

PORT MAP(DIN,CLK,RST,Z);

CHIP2:ERRORDET_CORR16

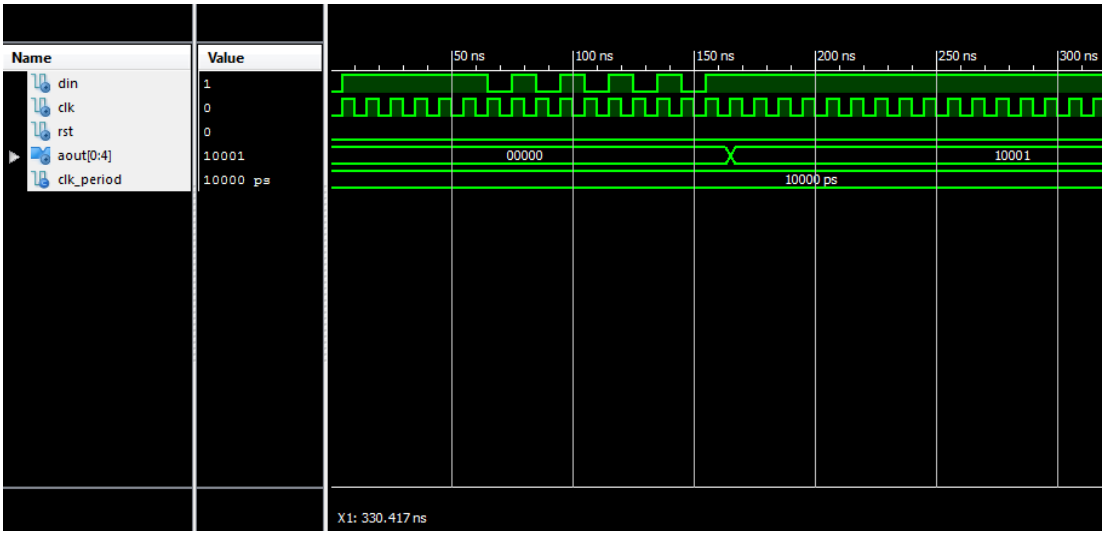
PORT MAP(Z,ZO);

CHIP3:DECODER16

PORT MAP(ZO,AOUT);

END BEHAV;

SIMULATION RESULT OF RECEIVER



VHDL CODE FOR DECODER

LIBRARY IEEE;

USE IEEE.STD_LOGIC_1164.ALL;

ENTITY DECODER16 IS

PORT(ZO:IN STD_LOGIC_VECTOR(0 TO 15);

ISSN: 2348-4098
VOLUE 02 ISSUE 06 JULY 2014

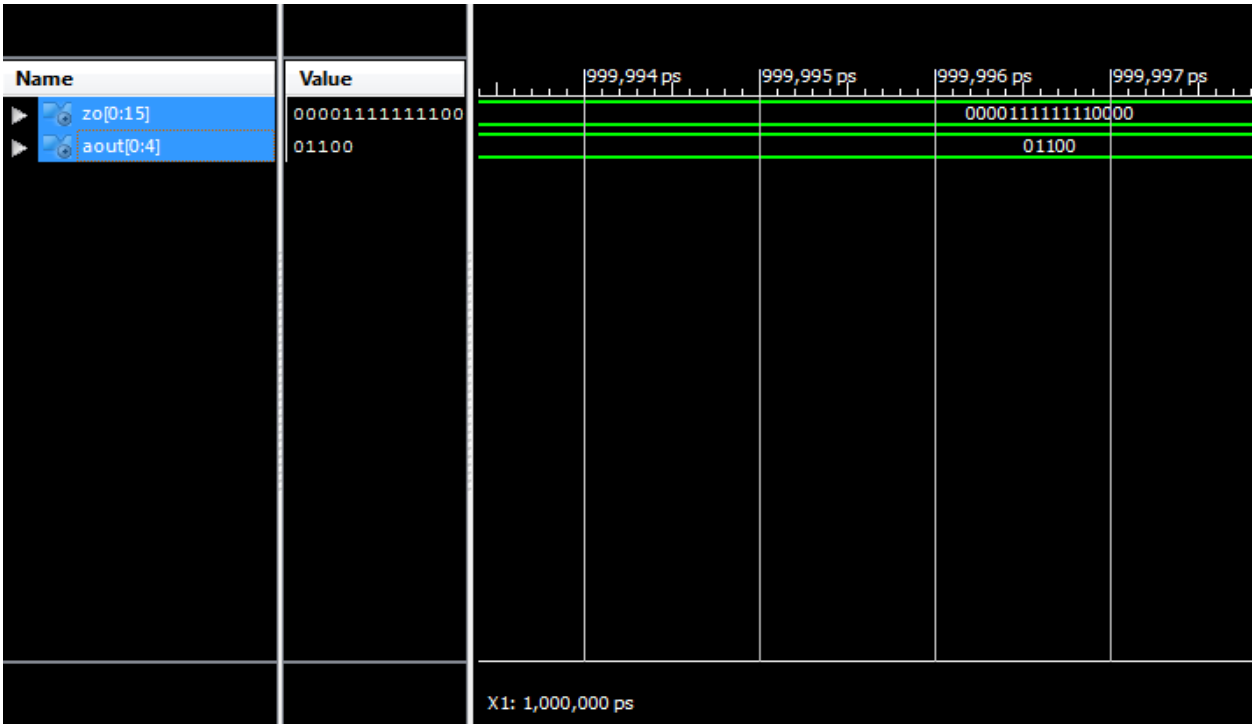
```
                AOUT:OUT STD_LOGIC_VECTOR(0 TO 4));
END DECODER16;
ARCHITECTURE BEHAV OF DECODER16 IS
BEGIN
PROCESS(ZO)
        BEGIN
CASE ZO IS
                WHEN "0000000000000000" => AOUT <="00000";
                WHEN "0101010101010101" => AOUT <="00001";
                WHEN "0011001100110011" => AOUT <="00010";
                WHEN "0110011001100110" => AOUT <="00011";
                WHEN "0000111100001111" => AOUT <="00100";
                WHEN "0101101001011010" => AOUT <="00101";
                WHEN "0011110000111100" => AOUT <="00110";
                WHEN "0110100101101001" => AOUT <="00111";
                WHEN "0000000011111111" => AOUT <="01000";
                WHEN "0101010110101010" => AOUT <="01001";
                WHEN "0011001111001100" => AOUT <="01010";
                WHEN "0110011010011001" => AOUT <="01011";
                WHEN "0000111111110000" => AOUT <="01100";
                WHEN "0101101010100101" => AOUT <="01101";
                WHEN "0011110011000011" => AOUT <="01110";
```

ISSN: 2348-4098
VOLUE 02 ISSUE 06 JULY 2014

```
WHEN "0110100110010110" => AOUT <="01111";

WHEN "1111111111111111" => AOUT <="10000";

WHEN "1010101010101010" => AOUT <="10001";

WHEN "1100110011001100" => AOUT <="10010";

WHEN "1001100110011001" => AOUT <="10011";

WHEN "1111000011110000" => AOUT <="10100";

WHEN "1010010110100101" => AOUT <="10101";

WHEN "1100001111000011" => AOUT <="10110";

WHEN "1001011010010110" => AOUT <="10111";

 WHEN "1111111100000000" => AOUT <="11000";

WHEN "1010101001010101" => AOUT <="11001";

WHEN "1100110000110011" => AOUT <="11010";

WHEN "1001100101100110" => AOUT <="11011";

WHEN "1111000000001111" => AOUT <="11100";

WHEN "1010010101011010" => AOUT <="11101";

WHEN "1100001100111100" => AOUT <="11110";

WHEN "1001011001101001" => AOUT <="11111";

WHEN OTHERS => AOUT <= "ZZZZZ";

        END CASE;

END PROCESS;

END BEHAV;
```

ISSN: 2348-4098
VOLUE 02 ISSUE 06 JULY 2014

SIMULATION RESULT OF DECODER



5. CONCLUSION

The results of the orthogonal code implementation show that this technique improved the error detection from 50% to93% for 8-bit orthogonal code and 99.9% for 16-bit orthogonal code. The technique proposed can be applied to any encoding system used for digital transmission. Future work includes improvement of correction capasbilities of the orthogonal code and paralell implementation to speed up the data processing.

REFERENCES

[1] Baicheva, T., S. Dodunekov, and P. Kazakov, "Undetected error probability performance of cyclic redundancy-check codes of 16-bit redundancy," IEEE Proc. Comms., Vol. 147, No. 5, Oct. 2000, pp. 253- 256.

[2] A. Hokanin, H. Delic, S. Sarin, "Two dimensional CRC for efficient transmission of ATM Cells over CDMA," IEEE Communications Letters, Vol. 4, No. 4, April 2000, pp.131-133.

[3] Stylianakis V., Toptchiyski S, "A Reed-Solomon coding/decoding structure for an ADS modem," Electronics, Circuits and Systems,

[4] Stylianakis V ,Toptchiyski S, "A Reed-Solomon coding/decoding structure for an ADS modem", Electronics, Circuits and Systems, 1999. Proceedings of ICECS apos ; 99. The 6th

ISSN: 2348-4098
VOLUE 02 ISSUE 06 JULY 2014

IEEE International Conference , Volume 1, Issue , 1999, pp. 473 – 476.

[5] S. Faruque, "Broadband Communications Based on Code Division Parallel Access (CDPA)", The International Engineering Consortium (IEC), Annual Review of Communications, Vol. 57, ISBN: 1-931695- 28-8, Nov. 2004.

[6] Saleh Faruque , "Error Control Coding Based on Orthogonal Codes", Wireless Proceedings, Vol. 2, pp. 608-615, 2004.

[7] S. Faruque, A. Dhirde ,N. Kaabouch "Forward error control coding based on orthogonal code and its implementation using FPGA", 7th IASTED International Conference, Montreal, Quebec, Canada, May 30 – June 1.