Improvement of Error Correction and Detection Using Orthogonal Code Convolution

1Vijaya K. Shendre  
M.Tech Scholar, Department of Electronics Engineering  
Wainganga College of Engineering & Management  
Nagpur, India  
kittu.shendre@gmail.com

2Rahul Nawkhare  
Asst. prof. Department of Electronics Engineering  
Wainganga College of Engineering & Management  
Nagpur. India  
rahulnawkhare@gmail.com

3Jayant Mankar  
Asst. prof., Department of Electronics Engineering  
SRMCEW, Nagpur, India  
sushjay@rediffmail.com

4Yuwraj V. Waghmare  
Asst. prof. Department of Electronics Engineering  
SRMCEW, Nagpur, India  
yuwraj.w@gmail.com

5Niraj Telrandhe  
Asst. Prof. Department of CSE  
GNIET, Nagpur, India  
niraj.telrandhe@gmail.com

Abstract: Digital communication system, convolution coding is preferred for the channel coding as it facilitates a better error correction as comparison to block coding. 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. In this project we propose a high efficient combined error detection and correction technique based on the Orthogonal Codes Convolution, Closest Match, and vertical parity. This method will experimentally simulate using Xilinx software. The propose technique will detects 99.99% of the errors and corrects as predicted up to (n/2-1) bits of errors in the received impaired n-bit code.

Keywords: Error detection and correction, XilinlXISE13.1i,Orthogonal Code Convolution.

I. INTRODUCTION

In communication system, 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. 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. Some of those techniques can only detect errors, such as the Cyclic Redundancy Check (CRC) [8-6]; others are designed to detect as well as correct errors, such as Solomon Codes [10, 5], Hamming Codes [4], and Orthogonal Codes Convolution (OCC) [7, 3]. However, the existing techniques cannot achieve high efficiency in error detection and correction as well as meet bandwidth requirements, especially with the increase in the quantity of data transmitted.

II. ORTHOGONAL CODES

Orthogonal codes are binary valued, and they have equal number of 1’s and 0’s. An n-bit orthogonal code has n/2 1’s and n/2 0’s; i.e., there are n/2 positions where 1’s and 0’s differ [7,3]. Therefore, all orthogonal codes will generate zero parity bits. The concept is illustrated by 16-bit orthogonal codes as shown in Fig. 1. It has 16 orthogonal codes and 16 antipodal codes for a total of 16 bi-orthogonal codes. The inverse of orthogonal codes is antipodal code; they have the same properties.
To improve the detection and correction capabilities of the OCCM technique, the advance method, OCCMP(orthogonal code convolution with Closest Match and vertical parity) is proposed. This mechanism offers a decision process in error correction, where the incoming impaired orthogonal code is examined for correlation with the codes stored in a look-up table, for a possible match.

**TABLE I**

<table>
<thead>
<tr>
<th>Code length(n)</th>
<th>Detection Capability (nc)</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>71.88%</td>
</tr>
<tr>
<td>16</td>
<td>80.35%</td>
</tr>
<tr>
<td>32</td>
<td>86.01%</td>
</tr>
<tr>
<td>64</td>
<td>90.07%</td>
</tr>
</tbody>
</table>

Before transmission, a m-bit data set is mapped into a unique n-bit orthogonal code. For example, a 5-bit data set is represented by a unique 16-bit orthogonal code, which is transmitted without the parity bit. Based on code correlation, the received data is decoded. It can be done by setting a threshold between two orthogonal codes. The following equation is used for setting threshold:

$$d_{th} = \frac{n}{4}$$  \hspace{1cm} (1)

Where $n$ is the code length and $d_{th}$ is the threshold, which is midway between two orthogonal codes. Therefore, for the 16-bit orthogonal code (Fig. 2), we have $d_{th} = 16/4 = 4$. This mechanism offers a decision process in error correction, where the incoming impaired orthogonal code is examined for correlation with the codes stored in a look-up table, for a possible match.

**TABLE II**

<table>
<thead>
<tr>
<th>Code length(n)</th>
<th>Correction Capability (nc)</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>1</td>
</tr>
<tr>
<td>16</td>
<td>3</td>
</tr>
<tr>
<td>32</td>
<td>7</td>
</tr>
<tr>
<td>64</td>
<td>15</td>
</tr>
</tbody>
</table>

**Fig. 3:** Illustration of 16-bit OCC encoding and decoding

### III. METHODOLOGY

#### A. Design methodology:

To improve the detection and correction capabilities of the OCCM technique, the advance method, OCCMP(orthogonal code convolution with Closest Match and vertical parity) is proposed. This
technique allows the transmission of several successive codes (16 codes in this paper) followed by their vertical parity byte. As the orthogonal codes have property of closure, the parity byte, which is the exclusive sum (XOR) of Sixteen orthogonal codes, is also an orthogonal code, and thus, the same technique used to detect and correct errors of each code in the block of data and in the parity byte. After a byte is received, the system checks and corrects errors if the code is corrupted using the Closest Match technique. After checking and correcting the received 17 bytes by receiver, the new vertical parity byte is calculated corresponding to the 16 data bytes and compares the new byte to the received parity byte in order to detect and correct more errors.

**TABLE III**

<table>
<thead>
<tr>
<th>Code length(n)</th>
<th>Detection Capability (nc)</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>93.57%</td>
</tr>
<tr>
<td>16</td>
<td>99.95%</td>
</tr>
<tr>
<td>32</td>
<td>99.99%</td>
</tr>
<tr>
<td>64</td>
<td>99.99%</td>
</tr>
</tbody>
</table>

**B. Transmitter**

The transmitter consists of four blocks: an encoder, a memory, a vertical parity generator and a shift register. A 5-bit data is encoded by an encoder to 16-bit orthogonal code using the look-up table (Fig. 2). After encoding, that 16-bit orthogonal code is stored in a 17 by 16 memory and a parity byte is calculated representing the columns; for each column, a parity bit is calculated. A shift register is used to convert these parallel 272 bits in serial data and then that data is transmitted in serial form.

Fig. 4 shows the block diagram of the transmitter. The input signal is parallel 5-bit data and the output is serial codes that take 272 periods of clock to transmit all the information.

![Fig:4 Block Diagram of Transmitter](image)

**C. Receiver**

As shown in Fig. 5, when a block of 16-bit code is received by receiver, it first converts it into parallel form by serial to parallel converter. XOR in the bit of data is carried out to detect the error and the Closest Match technique is used to correct a one bit error or multi-bit errors in the received code by comparing that received code with each code in the look-up table. By using counter the number of 1’s in the resulting signal is calculated. If that number is any number, except zero, shows an error or errors in the received code and the closest match code is used to correct it if only one bit error is detected. If there is an error in more than one bit, the received code is retransmitted. The indication for the retransmission is given by the flag. After matching the received sixteen codes, a new parity byte (Pc) is calculated using these matching codes and “XOR” performance is conducted between the new and corrected received parity byte (Pr). The 1’s in the resultant signal show the column with an odd number of error or errors. The location of the error or errors is easily found by using flag which will become on for the block of corrupted code. Decoding is performed after correcting the error or errors, and these 256 corrected bits are decoded to Sixteen 5-bit data in order.
III. IMPLEMENTATION AND RESULTS

The system has been going to implement using the software Xilinx ISE 9.2i. For error detection, in a previous work [2], it was shown that if an orthogonal code changes to another orthogonal code, it is not possible to detect the problem. However, with OCCMP, when one block in the package changes to another orthogonal code, the error can be detected using the parity byte. Let’s assume that the number of blocks in the memory is k (including the parity byte). When more than one block in this memory change to other orthogonal codes and there is an even number of errors in one column, the errors cannot be detected.

The number of combinations that cannot be detected is given as follows:

\[
\begin{align*}
  n_i &= \begin{cases} 
  C_{k-1}^i \cdot C_{2^{n-1}}^{i/2} & \text{for } i \text{ even} \\
  C_{k-1}^i \cdot C_{2^{n-1}}^{i-1} & \text{otherwise}
  \end{cases}
\end{align*}
\]

where \( i=2, 3, \ldots, k \) is the number of corrupted blocks, and \( n_i \) is the amount of combinations that cannot be detected.

The total number of combinations that cannot be detected for \( k \) blocks of \( n \)-bit code is \( N \) given by:

\[
N = \sum_{i=2}^{k} n_i
\]

For 17 block of 16 bit code

\[
N = \sum_{i=2}^{17} n_i
\]

\[
= 17C2 \cdot 31C1 + 17C4 \cdot (31C1)^2 + 17C6 \cdot (31C1)^3 + 17C8 \cdot (31C1)^4 + 17C10 \cdot (31C1)^5 + 17C12 \cdot (31C1)^6 + 17C14 \cdot (31C1)^7 + 17C16 \cdot (31C1)^8 + 17C18 \cdot (31C1)^9 + 17C20 \cdot (31C1)^{10} + 17C22 \cdot (31C1)^{11} + 17C24 \cdot (31C1)^{12} + 17C26 \cdot (31C1)^{13} + 17C28 \cdot (31C1)^{14} + 17C30 \cdot (31C1)^{15} + 17C32 \cdot (31C1)^{16}
\]

\[
= 215556182.9 \times 10^6
\]

The detection rate for \( k \) block of \( n \)-bit code is

\[
\frac{2^k \cdot n - N}{2^k \cdot n} \times 100\%
\]

Detection rate for OCCMP-16

\[
((2^{17} \cdot 16 \cdot 215556182.9 \times 10^6)/2^{17} \cdot 16) \times 100\%
\]

\[
= 99.99\%
\]

A. Encoder Simulation

We have done RTL simulation of encoder to ensure the proper working of standalone module. The input data is encoded to 16-bit orthogonal code with the rising edge of the clock signal. The 16-bit orthogonal code is outputted through a signal “data_o”. For example, the 5-bit data “00001” is encoded to “0101010101010101” 16-bit orthogonal code, this is shown in figure 5.1
B. Transmitter

After simulating encoder and parallel to shift register, the transmitter is simulated and results are shown in figure 5.4. The transmitter reset, using the reset signal “reset”. This resets the transmitter to the default value “00000”. 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. For example, the 5-bit data “00001” is encoded to “0101010101010101” 16-bit orthogonal code. The generated orthogonal code is then transmitted serially using a shift register with the rising edge of the clock. The bits are outputted through a signal “s_out”.

C. Receiver

The receiver is simulated with its components such as serial to parallel converter and counter; memory, comparator decoder etc. and results are shown in figure 5.7. The receiver reset, using the reset signal “reset”. This resets the receiver to the default value “00000”. For the conditions “0101010101010101” it gives the value “00001”.

D. Transceiver

Figure 5.8 shows the interconnection of transmitter and receiver for the input condition “11110”. 

---

**Fig 6: Simulated result of encoder**

**Fig 7: Simulated result of Transmitter**

**Fig 8: Simulated Result of Receiver**

**Fig 9: Simulated Result of Transceiver**
IV. CONCLUSION

In this work, we are going to develop and implement a real-time high-efficiency technique to detect and correct errors in digital communication. The experimental results will show that the proposed technique can detect 99.99% of errors and correct up to \((n/2-1)\)-bit of errors for \(n\)-bit orthogonal codes.

<table>
<thead>
<tr>
<th>Technique</th>
<th>Detection Rate</th>
<th>Correction Capability</th>
</tr>
</thead>
<tbody>
<tr>
<td>OCC-8</td>
<td>71.88%</td>
<td>1</td>
</tr>
<tr>
<td>OCCM-8</td>
<td>93.57%</td>
<td>1</td>
</tr>
<tr>
<td>CRC-32</td>
<td>99.99%</td>
<td>0</td>
</tr>
<tr>
<td>OCCMP-8</td>
<td>99.99%</td>
<td>3</td>
</tr>
<tr>
<td>OCCMP-16</td>
<td>99.99%</td>
<td>7</td>
</tr>
</tbody>
</table>

V. REFERENCES