2021 9th International Conference on Information and Communication Technology (ICoICT) | 978-1-6654-0447-1/21/$31.00 ©2021 IEEE | DOI: 10.1109/ICoICT52021.2021.9527522

Modified Bit Parity Technique for Error Detection of 8 Bit Data

1st Fakhira Zulfira
School of Computing
Telkom University
Bandung, Indonesia
firafakhira25@gmail.com

2nd Hilal Hudan Nuha
HUMIC Engineering
Telkom University
Bandung, Indonesia
hilalnuha@telkomuniversity.ac.id

3rd Dodi Wisaksono Sudiharto
School of Computing
Telkom University
Bandung, Indonesia
dodiws@telkomuniversity.ac.id

4th Rio Guntur Utomo
HUMIC Engineering
Telkom University
Bandung, Indonesia
riogunturutomo@telkomuniversity.ac.id

Abstract—**Bit parity is often used as an error detection technique in sending digital data that has interruption during transmission. Error detection techniques allow correcting data to be error-free. Research that has been used using bit parity has been done but only detected even parity and odd parity where the process is very simple so that modifications can be made to detect other than even or odd parity. In this study a modification of the bit parity method was built in which each message received in the first 4 bits will be detected whether the bit value is a multiple of 4 and the remaining bits will be evenly detected. Error detection using the proposed parity bits results in a fairly good detection system and is more efficient if there are more bits in the message.**

Keywords—Error Detection Techniques, Parity Bit, transmission

I. Introduction

Error detection techniques are methods that allow reliable digital data transmission over unreliable communication channels. Error Detection due to noise or other types of interference during transmission from source to destination. Many communication channels are vulnerable to noise in the medium. Therefore, errors can occur during data transmission from source to destination. Error detection techniques enable the detection of such errors. In addition, error correction allows the recovery of the actual data in most cases into error-free data.

Many studies have conducted testing related to error detection techniques. In [1], a simulation application is designed that aims to describe how the error correction process is in the process of sending data in the form of numbers in binary numbers. Based on the test results, the data sent will be detected if an error occurs, then the application will correct any errors that have been detected. The results of the observation show that the error occurs at the time of sending data because of errors in the bits that are sent.

In [2], it is described that interference with Network on Chip (NoC) in the NoC environment causes multi-bit errors. This results in data loss and requires retransmission. To solve this problem, a Joint Crosstalk Avoidance with Multiple bit Error Correction (JCAMEC) is carried out which uses the extended Hamming code and simple parity check code along with duplications. The results obtained by our coding technique are superior to the Hamming and JMEC coding techniques.

Apart from that bit error detection, similar technique is applied to UART-based Arduino Serial Communication as done in [3]. In this study, the Hamming Code method is used to encode and decode data, as well as detect and correct errors in data that have errors in the testing process. The average delay obtained is 102.7 ms for 5-bit data and 109.5 ms for 4-bit data in the encoding process. And 17.5 ms for 10-bit data and 100.1ms for 11-bit data in the decode process.

Hamming code is widely used in bit error detection, other research is described in [4] and after testing it was found that using Hamming Code is only able to perform single error correction, so the input and output data in the Hamming Code method must be the result of a power of 2n with n must Hamming Code is greater than one with the length of the data input and output Hamming Code must be at least equal to 4 bits, and also the Hamming Code cannot check the position of more than one data error (bad bit).

In addition, in other studies, testing has been carried out for the correction of multi-bit errors using partition hamming code, in [5] it is proven that partition hamming code can detect and correct more errors that occur in a transmitted message compared to sending 7 message bits at once without broken down first. To overcome the many parity bits, it can be concluded that the use of the message bit partition pattern into several smaller message bit blocks, Hamming code can detect and correct multi-bit errors that occur.

Apart from hamming code, there is also an error detection technique that is quite widely used because it is simple and accurate, namely the parity bit. In paper [6], the parity bit is used to detect limited magnitude errors in emerging multilevel cell memory, most of the emerging memory technologies use MLC to achieve higher integration and density due to its high density, non-volatility, and byte-addressable. But the weakness is that the error usually occurs temporarily, while MLC is susceptible to physical phenomena that cause permanent errors. So that the magnitude of error must be strictly limited. The results show that

One Bit Parity (OBP) reduces the complexity and delay of coding circuits and error detection, while Two Bit Parity (TBP) also reduces the number of parity bits for some configurations. Therefore, OBP and TBP can be efficient alternatives for detecting limited magnitude errors in MLC memory using bit-level binary coding.

Remainder of this paper is organized as follows. Section II describes the proposed methodology. Section III discusses the experiments and analysis. Conclusion is provided in section IV.

II. Methodology

This section describes the proposed method by combining two different parity bit generation.

A. *Input data*

This system will perform error detection from the input message which ideally consists of 8 bits. The initial 4 bits will detect whether the value of the bit is a multiple of 4 and the remaining bits will then detect whether the bit is even parity.

B. *System design*

The initial process starts from the input in the form of an 8 binary bit message that represents the message, then the message bits on the partition which are then determined by the parity bit based on a modified scheme that checks for multiples of 4 and even parity. As a result, the message will have 2 bits added at the end of the message so that the number of bits in the output is 10 bits. The system design built to solve the problems can be seen in Figure 1.

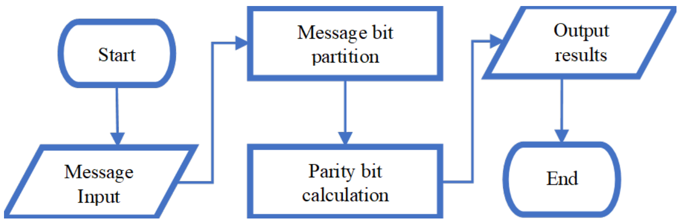

Fig. 1 System design flowchart

C. *Bit Parity modification method*

The parity bit is also called the check bit, which is an additional bit that is placed at the end of a byte. The parity bit is used for accuracy checking purposes, i.e. checking for errors during transmission [7]. Parity bits can also be added during the encoding stage to detect bit error due to compressions [8,9,10].

The way the parity bit works starts from the sender who will add 1 additional bit (Parity Bit) to the data, to describe the characteristics of the data. The value of the parity bit (1 or 0) is not allowed arbitrarily. In the process of transmitting, the data is sent together (data and parity bits). At the receiving terminal, our data is read and decoded in the same way as when determining the parity bit value on the sender's side. Then the result of the decodization is compared with the parity bit delivered by the sender. If the result of reading (decodization) of the sent data is the same as the parity bit, then the data can be considered correct. And if the difference in value between the decodization results and the parity bit is obtained, the data can be classified as error data.

In this system, a modification of the parity bit technique will be built in which the parity bit scheme used to detect errors in received messages is designed into two parts, namely to detect whether the message information is a multiple of 4 and to detect whether the message is a parity event. After detection, the additional bits for detection of multiples of 4 will become the 9th bit and followed by additional bits for the detection of even parity on the 10th bit. So that the number of bits resulting from error detection is 10 bits. The modified parity bit design can be seen in Figure 2. The procedure is given by:

$$b_9 = p_1 = f(b_5, b_6, b_7, b_8), \tag{1}$$

where p_i denotes the i-th parity bit and b_n is the n-th bit. The parity function f is defined as:

$$f(b_5, b_6, b_7, b_8) = \overline{\left((b_5 \oplus b_6) \oplus b_7\right) \oplus b_8}. \tag{2}$$

The operations include the exclusive-or ($\oplus$) and the inversion ($\bar{\cdot}$). The 10-th bit is calculated by:

$$b_{10} = p_2 = g(b_1, b_2, b_3, b_4), \tag{3}$$

$$g(b_5, b_6, b_7, b_8) = \overline{(b_1 + b_2) + (b_3 + b_4)}\,, \tag{4}$$

where $(+)$ denotes the binary or-operation. Regular even parity bits use standard operation for all bits.

$$h(b_n, b_{n+1}, b_{n+2}, b_{n+3}) \\ = b_n \oplus b_{n+1} \oplus b_{n+2} \oplus b_{n+3}. \tag{5}$$

Bit 1 to 4 will pass 3 OR gates and 1 NOT gate, where there must be 4 bits to detect error. Whereas for bit 5 to the last bit there is an XOR gate and 1 NOT gate at the very end. The XOR process is carried out in pairs on each existing bit so that the bit length that can be detected is unlimited. Final message and parity bit structure is shown in Table I.

III. Experiments and Analysis

The 8-bit message is processed for error detection of multiples of 4 in the first 4 bits and even parity for the next 4 bits. In the initial 4 bits of the message, determining the value of the parity bit (1 or 0) is done by XORing all the bits in the data pairs, the final result of the XOR process for all these bits will be used as a reference for determining the value of the parity bit. to be added. The added parity bit is the inversion of the resulting bit after XOR, which was 0 to 1 and from 1 to 0.

For the next 4 bits, determining the value of the parity bit is done by OR on the first 2 bits and doing OR for the last 2 bits, the results of the OR process for the last 2 bits are NOR with the inversion results of the initial 2 bits that have been OR-ed. The final result after doing NOR is the addition of bits for the parity bit.

For example data in the form of 01001010, then there are 2 parts that will be detected, namely 4 initial bits 0100 and 4 bits ending 1010, the process is carried out, namely:

1. (Bit 1) OR (Bit 2) = 0 OR 1 = 1
2. NOT (Result 1) = NOT 1 = 0
3. (Bit 3) OR (Bit 4) = 0 OR 0 = 0
4. (Result 2) NOR (Result 3) = 0 NOR 0 = 1

Therefore, the parity bit for the first 4 bits that detects a multiple of 4 error is 1

Furthermore, the parity event detection process is carried out, namely:

1. (Bit 5) XOR (Bit 6) = 1 XOR 0 = 1
2. (Result 1) XOR (Bit 7) = 1 XOR 1 = 0
3. (Result 2) XOR (Bit 8) = 0 XOR 0 = 0
4. NOT (Result 3) = NOT 0 = 1

So the parity bit for the last 4 bits which detects the error in the parity event is 1

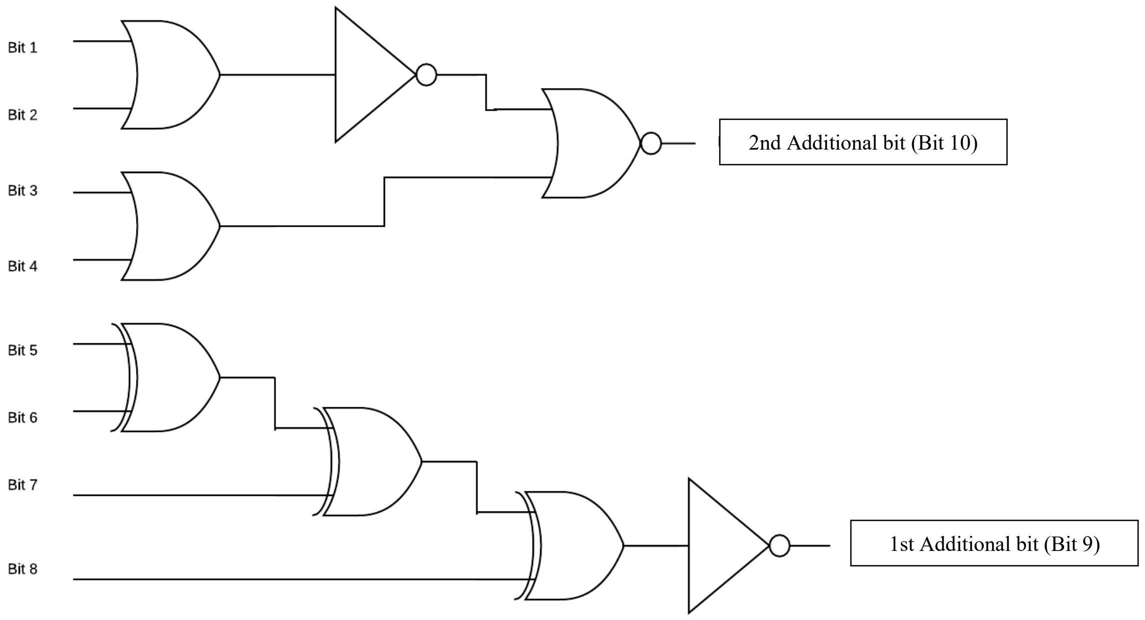

Fig. 2 Triple and even parity arrangement scheme

TABLE I. MESSAGE AND PARITY BITS STRUCTURE

Message bits								Parity bits	
Bit 1	*Bit 2*	*Bit 3*	*Bit 4*	*Bit 5*	*Bit 6*	*Bit 7*	*Bit 8*	*Bit 9*	*Bit 10*
0	1	0	0	1	0	1	0	1	1

The first bit through to the 8th bit on the partition and a scheme designed to detect errors is carried out. The results of the parity bit in the detection of the first 4 bits of error will be used as an additional bit to be the 9th bit of the initial message and the results of the parity bit in the detection of the next 4-bit error will be made the additional bit to the 10th bit right after the parity bit of 4.

Based on the process described the message 01001010 after the parity bit is carried out to detect errors and an additional bit is added, will be 0100101011 where the last 2 bits are the parity bits resulting from the error detection that has been done.

A. Pattern Case Detection

All bit patterns with a bit length of 8 can be detected properly by the built system so that there are no case patterns that cannot be detected if the number of bits is 8, but if the message length is not 8 new bits will be found some cases that cannot be detected properly because on checking the number of bit values multiples of 4, there are at least 4 bits in the message so that detection can run properly. The results of the implementation of error detection that were built can be seen in Table II.

In another case, if there is a message that is more than 8 bits it can also be detected but still with the same rule, the initial 4 bits are detected whether the value is a multiple of 4 and the remaining bits are detected whether even parity.

TABLE II. AN EXAMPLE OF IMPLEMENTING CASE DETECTION.

No.	Input	Multiples of 4	Even	Multiple of 4 Extended Bit	Extended Even bit
1	01001010	1100	1010	1	1
2	10001111	1010	1111	1	1
3	11101011	1110	1011	0	0
4	10001110	1000	1110	0	0
5	11000111	1100	0111	1	0
6	01000001	0100	0001	1	0
7	11111100	1111	1100	0	1
8	01110011	0111	0011	0	1

An example is the case, for example the message is 1110101001 so the process will be carried out as follows:

1. (Bit 1) OR (Bit 2) = 1 OR 1 = 1
2. NOT (Result 1) = NOT 1 = 0

3. (Bit 3) OR (Bit 4) = 1 OR 0 = 1
4. (Result 2) NOR (Result 3) = 0 NOR 1 = 0

So the parity bit for the first 4 bits that detects a multiple of 4 error is 0.

Furthermore, the parity event detection process is carried out, namely:

1. (Bit 5) XOR (Bit 6) = 1 XOR 0 = 1
2. (Result 1) XOR (Bit 7) = 1 XOR 1 = 0
3. (Result 2) XOR (Bit 8) = 0 XOR 0 = 0
4. (Result 3) XOR (Bit 9) = 0 XOR 0 = 1
5. (Result 4) XOR (Bit 10) = 1 XOR 1 = 0
6. NOT (Result 5) = NOT 0 = 1

So the parity bit for the next remaining bit that detects even parity is 0.

The first bit to the 10th bit on the partition and a scheme designed to detect errors is carried out. The results of the parity bit in the detection of the first 4 bits of error will be used as an additional bit to be the 11th bit of the initial message and the results of the parity bit in error detection in the remaining bits will be used as the additional bit to be the 12th bit right after the parity bit of 4.

Based on the described process, the message 1110101001 after the parity bit is carried out to detect errors and an additional bit is added, it will become 111010100100 where the last 2 bits are the parity bits resulting from the error detection that has been done.

B. *Undetected pattern case*

There are several cases where the method developed cannot detect errors from the message input. When the message to be detected is less than 5 bits long, the system will not be able to detect it accurately, because it lacks bits when it detects the initial 4 bits.

For example, for example message 101, it will be processed: (Bit 1) OR (Bit 2) = 1 OR 0, then bit 3, which is 1, there is no bit pair to OR, so in this case the message is not detected properly and may cause errors in the resulting parity bit.

C. *Rate calculation*

The difference in bit length in the initial message input and after adding the parity bit can affect the level of efficiency in the error detection process. To find out the code rate, it can be calculated from the data bit length (k) divided by the total bits after adding the parity bit (n) so that a comparison of the code rate of the different initial message input can be done as shown in Table III. The codeword length (n) is calculated by:

$$n = k + r. \tag{6}$$

The code rate (R) that represents the coding scheme efficiency is given by

$$R = k/n. \tag{7}$$

Higher values of the code rates indicate higher coding efficiency. Whereas the lower values indicate low efficiency hence the coding scheme is not desirable.

TABLE III. CODE RATE CALCULATION AND COMPARISON

No.	Data Bit (k)	Total parity (r)	Codeword (n = k + r)	Code rate (R = k / n)
1	8	2	10	80%
2	9	2	11	81%
3	10	2	12	83.33%
4	11	2	13	84.6%
5	12	2	14	85.7%

From the table above, it can be seen that the longer the initial message input, the higher the code rate. So it is proven that if the longer the message is detected, the more efficient it will be because the parity bit is added only 2 bits for any message bit length.

D. *Comparison with the regular even parity bit*

The even parity bit has the process of determining the value of the parity bit (1 or 0), namely by XORing all the bits in the data in pairs, the final result of the XOR process enters the NOT gate so that the final result is inversed from the XOR process.

An example of an ordinary parity bit, for example, message 10110100, so the process is as follows:

1. (Bit 1) XOR (Bit 2) = 1 XOR 0 = 1
2. (Result 1) XOR (Bit 3) = 1 XOR 1 = 0
3. (Result 2) XOR (Bit 4) = 0 XOR 1 = 1
4. (Result 3) XOR (Bit 5) = 1 XOR 0 = 1
5. (Result 4) XOR (Bit 6) = 1 XOR 1 = 0
6. Result 5) XOR (Bit 7) = 0 XOR 0 = 0
7. Result 6) XOR (Bit 8) = 0 XOR 0 = 0
8. NOT (Result 8) = NOT 0 = 1

Therefore, the parity bit that detects the error in the message is 0.

In the ordinary even parity bit process, the error detection process from bit 1 to the last bit will only be checked for one type of checking, namely even parity checking. This is different from the system built where after modification of the even parity bit method, multiples of 4 can be detected in the first 4 bits.

In addition, the result of adding the parity bit to the message is the last bit and the final result is 101101001 where the addition of the parity bit is only 1 in each message. This shows the difference with the built method because in the modification of the parity bit the addition of the bit is 2 bits.

TABLE IV. PARITY BIT FUNCTION COMPARISON

n	Proposed method	Even parity	Odd parity
9	$f(b_5, b_6, b_7, b_8)$	$h(b_n, b_{n+1}, b_{n+2}, b_{n+3})$	$h\overline{(b_n, b_{n+1}, b_{n+2}, b_{n+3})}$
10	$g(b_5, b_6, b_7, b_8)$		

Table IV summarizes the parity bits operations using the proposed method, standard even, and odd parity scheme. It can be seen that the proposed method uses two different equations for the first and second four bits. Whereas the even and odd parity use the same equations for all bit string.

IV. Conclusions

Based on the analysis of the overall system testing performed, the proposed parity bit modification results in better detection than the usual even parity bit. After modification, the parity bit can detect the number of bit values whether a multiple of 4 or not in the initial 4 bits. This method works more efficiently if the message has a large enough number of bits by looking at the resulting code rate which increases as the total message bits get longer.

References

[1] Mahendra, Rizqa Gardha, Marti Widya Sari, and Meilany Nonsi Tentua. "Simulasi Deteksi Bit Error Menggunakan Metode Hamming Code Berbasis web." Jurnal Dinamika Informatika 5.2 (2016).

[2] Teja, T. Siva, et al. "Joint crosstalk avoidance with multiple bit error correction coding technique for NoC interconnect." 2018 International Conference on Advances in Computing, Communications and Informatics (ICACCI). IEEE, 2018.

[3] Andana, Anggi Fajar. "Implementasi Deteksi Dan Koreksi Error Pada Komunikasi Serial Arduino Berbasis Uart Dengan Metode Hamming Code" Diss. Universitas Brawijaya, 2018.

[4] Fauzi, Achmad, and R. Rahim. "Bit Error Detection and Correction with Hamming Code Algorithm." Int. J. Sci. Res. Sci. Eng. Technol 3.1 (2017): 76-81.

[5] Muhajir, Fajar, Syahril Efendi, and Sutarman Sutarman. "Deteksi dan Koreksi Multi Bit Error dengan Partition Hamming Code." Jurnal Teknovasi: Jurnal Teknik dan Inovasi 3.2 (2018): 1-9.

[6] Liu, Shanshan, Pedro Reviriego, and Fabrizio Lombardi. "Detection of Limited Magnitude Errors in Emerging Multilevel Cell Memories by One-Bit Parity (OBP) or Two-Bit Parity (TBP)." IEEE Transactions on Emerging Topics in Computing (2019).

[7] SlideShare, " [Online]. Available: https://www.slideshare.net/byanrich/modul-teknik-digital-dan-logika.

[8] Liu, Bo, Mohamed Mohandes, Hilal Nuha, Mohamed Deriche, Faramarz Fekri, and James H. McClellan. "A Multitone Model-Based Seismic Data Compression." IEEE Transactions on Systems, Man, and Cybernetics: Systems (2021).

[9] Nuha, Hilal Hudan, and Novian Anggis Suwastika. "Fractional fourier transform for decreasing seismic data lossy compression distortion." In 2015 3rd International Conference on Information and Communication Technology (ICoICT), pp. 590-593. IEEE, 2015.

[10] Nuha, Hilal H., Bo Liu, M. Mohandes, and M. Deriche. "Seismic data compression using signal alignment and PCA." In 2017 9th IEEE-GCC Conference and Exhibition (GCCCE), pp. 1-6. IEEE, 2017.