Comparison of Rectangular Parity Check Codes

Emre Kaan Özkan
18014601
Electronic And Communication
Engineering Department
Yıldız Technical University
İstanbul, Türkiye
l1418601@std.yildiz.edu.tr

***Abstract*— Parity bit is widely used in error correction-detection codes. Some of these algorithms are Cyclic Reduncancy Check (CRC), Parity Check, Check Sum. In this paper we examine Single Parity Check Codes and Rectangular Parity Check Codes.**

Keywords—Parity, Hamming, CRC, bit

I. INTRODUCTION TO PARITY

A parity bit, also known as a check bit, is a single bit that can be appended to a binary string. It is set to either 1 or 0 to make the total number of 1-bits either even ("even parity") or odd ("odd parity"). The purpose of the parity bit is to provide a simple way to check for errors later. When data is stored or transmitted electronically, it is not uncommon to "flip" the bits - from 1 to 0 and vice versa. Parity checks can help detect some of these errors. For example, to check a binary array with equal parity, the total number of sequences can be counted. If their number is uneven, an error may have occurred. Parity checking can be one-dimensional or two dimensional.

II. SINGLE PARITY CHECK CODES

Single parity-check code, an extra bit is added to every data unit (e.g., byte, character, block, segment, frame, cell, and packet). The objective of adding the parity bit is to make the total number of 1s in the data unit (including the parity bit) to become even. Some systems may use odd-parity checking, where the total number of 1s should then be odd. Even parity is typically used for synchronous transmission and odd parity for asynchronous transmission. The single parity-check code is used in ASCII where characters are represented by seven bits and the eighth bit is a parity bit. Simple parity-check codes can detect all single-bit errors. It can also detect multiple errors only if the total number of errors in a data unit is odd. A single parity-check code has a remarkable error-detecting capability, as the addition of a single parity bit can bring about half of all possible error patterns detectable. In the Fig.1 and Fig2, transmission scenarios for single parity check codes are given.

Type of bit parity	Successful transmission scenario
Even parity	Alice wants to transmit: 1001 Alice computes parity bit value: 1+0+0+1 (mod 2) = 0 Alice adds parity bit and sends: 10010 Bob receives: 10010 Bob computes parity: 1+0+0+1+0 (mod 2) = 0 Bob reports correct transmission after observing expected even result.
Odd parity	Alice wants to transmit: 1001 Alice computes parity bit value: 1+0+0+1 (mod 2) = 0 Alice adds parity bit and sends: 10011 Bob receives: 10011 Bob computes overall parity: 1+0+0+1+1 (mod 2) = 1 Bob reports correct transmission after observing expected odd result.

Fig. 1 : Transmission Scenario

Type of bit parity error	Failed transmission scenario
Even parity Error in the second bit	Alice wants to transmit: 1001 Alice computes parity bit value: 1^0^0^1 = 0 Alice adds parity bit and sends: 10010 **...TRANSMISSION ERROR...** Bob receives: 11010 Bob computes overall parity: 1^1^0^1^0 = 1 Bob reports incorrect transmission after observing unexpected odd result.
Even parity Error in the parity bit	Alice wants to transmit: 1001 Alice computes even parity value: 1^0^0^1 = 0 Alice sends: 10010 **...TRANSMISSION ERROR...** Bob receives: 10011 Bob computes overall parity: 1^0^0^1^1 = 1 Bob reports incorrect transmission after observing unexpected odd result.

Fig.2 : Failed Transmission Scenario

The single parity-check code takes k information bits and appends a single check bit using modulo-2 arithmetic to form a codeword length of $n(n = k + 1)$, thus yielding a $(k + 1, k)$ block code with $\frac{k}{k+1}$ code rate. The parity bit b_{k+1} is determined as the modulo-2 sum of the information bits[1]:

$$bn = bk + 1 = b1 + b2 + \ldots . bk \quad 1.0$$

Note that in modulo-2 arithmetic, we have 0 + 0 = 0, 0+1 = 1, 1+0 = 1 and 1 + 1 = 0.Many transmission channels introduce bit errors at random, independent of each other, and with low probability of p, where p is assumed to be the probability of an error in a single-bit transmission. Some error patterns are more probable than others.

For instance, with the extremely reasonable assumption of p < 0.5, error pattern with a given number of bit errors is more likely than an error pattern with even a larger number of bit errors. In other words, it follows that patterns with no error is more likely than patterns with one error and that in turn is more likely than patterns with two errors, so on and so forth. The single parity-check code fails if the error pattern has an even number of 1s. The probability of error-detection failure (i.e., the probability of undetectable error pattern or equivalently error patterns with even number of 1s) is as follows:

$$P_{Failure} = \binom{n}{2} p^2 (1-p)^{n-2} + \binom{n}{4} p^4 (1-p)^{n-4} + \ldots + \binom{n}{m}$$

Fig. 3 : $P_{Failure}$ Formula

Consider a (15, 14) single parity-check code. Compute the probability of an undetected message error, assuming that all bit errors are independent and the probability of bit error is $p = 10^{-6}$

Using equation ant noting that k=14, we have the probability of error-detection failure $P_{Failure} \cong 10^{-10.}$ This points to the fact that with a very high code rate of $\frac{14}{15}$, with a redundancy of only 6.7%, an impressive error rate reduction of nearly four orders of magnitude can be easily achieved.

III. RECTANGULAR PARITY CHECK CODES

In a “Rectangular Parity Check Codes” the information bits are organized in a matrix consisting of rows and columns. For each row and each column, one parity-check bit is calculated. As a result, the last column consists of check bits for all rows and the bottom row consists of check bits for all columns. In other words, the resulting encoded matrix of bits satisfies the pattern that all rows have even parity and all columns also have even parity. Assuming there are *M* information bits in each row and *N* information bits in each column, the code rate is then as follows:

$$\frac{k}{n} = \frac{MN}{(M+1)(N+1)}$$

Fig. 4: Code Rate Formula

The two-dimensional parity checks can detect and correct all single errors and detect two and three errors that occur anywhere in the matrix. However, only some patterns with four or more errors can be detected.
Fig. 5 [2] shows examples of detectable and undetectable error patterns, for M= 5, N=6, and thus $\frac{k}{n} = \frac{5}{7}$.

A two-dimensional parity check increases the likelihood of detecting burst errors, but on the other hand, requires more check bits. We will later expand on this in the context of interleaving to combat burst errors.

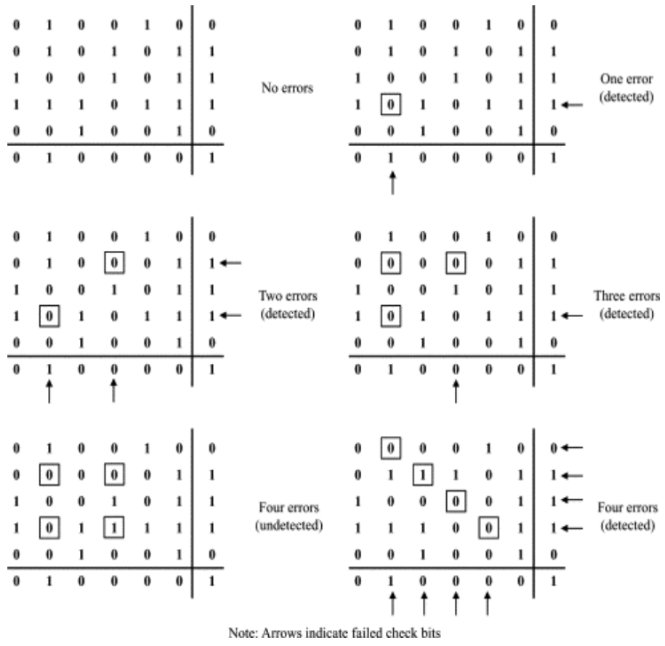

Fig. 5: Rectangular Parity Check Codes

IV. COMPARISON OF RECTANGULAR AND SINGLE PARITY CHECK CODES

In this title, we will examine the Rectangular Parity Check Code against the SPC.

- Error detection-correction
  First, let's explain Hamming distance[4] : Measures the number of bit flips to change one codeword into another. Hamming distance between two messages m1, m2: The number of bit flips needed to change m1 into m2.
  Example: Two bits flips needed to change codeword 00 to codeword 11, so they are Hamming distance of two apart: 00 → 01 → 11
  Suppose the minimum Hamming distance between any pair of codewords is d_{min}. Then we can detect at most $dmin - 1$ bit errors.

  Single parity check codes only catches odd numbers of bit errors and can't catch if 2 bits interchanged. SPC can't error correction. SPC $d_{min} = 2$ and it detects only 1 error.

  Rectangular parity check codes can error correction for one bit. Both simple parity and 2D parity do not catch all the errors also it can be detected up to 3 bite. RPC Codes $d_{min} = 4$ It can't use in 4-bit errors.

- Code Rate
  Suppose codewords of length n, messages length k (k < n), code rate R = k/n is a fraction between 0 and 1. So, we have a tradeoff:
  – High-rate codes (R approaching one) correct fewer errors but add less overhead.[3]
  – Low-rate codes (R close to zero) correct more errors but add more overhead.
  According to this information, Single Parity Check Codes code rate R = k/k+1 (k = messages length)

Rectangular parity check codes code rate varies with matrix size. For example 2*2 parity check codes code rate R = 4/9. SPC code rate more than RPC codes. Single Parity Check codes can fewer error detect but adding less overhead.

V.COMPARISON OF RECTANGULAR PARITY CHECK CODES

In this title, we will examine the rectangular parity check code comparison in itself. Firstly we examine code rate and error detection. We suppose it is 5*6 parity matrix. This matrix have 5/7 code rate. This matrix code rate more than 2*2 matrix. The code ratio of 2 * 2 matrix is 4/9. This case can be interpreted as follows:

- As the parity check matrix grows, the chance of error correction decreases.
- As the parity check matrix grows, number of adding bits increase. This case reduce efficiency.

VI.SIMULATION

In this section, we simulated the situations we reviewed above using MATLAB. In the first simulation, we simulated single parity check code. If we explain the first simulation:

First, the message we received as a string has been made encodable. And it was encoded. This coding process consisted of adding the parity bit to the end of the message. An error was added to the channel after encoding. And error detection was made in decoding section. The message was prepared again and displayed on the screen.

As seen in the Fig. 6, the message sent as "E" was received as "U" with the error. There is an error in column 3 of the matrix. And it was detected that the error exists.

The code rate in this simulation is 7/8. d_{min} = 2, the maximum number of errors we can detect is 1.

```
%% SINGLE PARITY CHECK - EVEN
    clear;
    %MESSAGE PREPARING
    disp("Program Starting")
    message = 'E'; %Input Message
    disp("Message :")
    disp(message)
    binary_message = dec2bin(message); %Message to binary
    %Resize
    msg = reshape(binary_message,[],1);
    msg = str2num(msg);
    msg = reshape(msg,[],7);

    len_row = length(msg(:,1));
    len_column = length(msg);

    %ENCODING
    for row=1:(len_row)
        row_sum(row) = mod(sum(msg(row,1:end)),2); %adding parity bit
    end
    disp("MESSAGE With Parity Bit")
    matrix = [msg row_sum']; %concat parity bit
    disp(matrix)
```

```
%CHANNEL ERROR
matrix(3) = ~matrix(3); %adding error
disp("Error With Matrix");
disp(matrix)

%DECODING - ERROR DETECTION
for row=1:(len_row)
    if row_sum(row) ~= mod(sum(matrix(row,1:end-1)),2) %parity check
        disp("Error Detected ! ")
        %disp(row)
    else
        disp("No Error: ")
        %disp(row)
    end
end
%MESSAGE PREPARING
matrix = matrix(:,1:end-1);
received_matrix = char(reshape(char('0' + matrix),[],7));
received_message = char(bin2dec(received_matrix));
received_message = reshape(received_message,[],length(message));
disp("Received Message With Error")
disp(received_message)
```

Fig. 5: Single Parity Check Codes Simulation

```
Program Starting
Message :
E
MESSAGE With Parity Bit
     1     0     0     0     1     0     1     1

Error With Matrix
     1     0     1     0     1     0     1     1

Error Detected !
Received Message With Error
U
```

Fig. 6: Output of SPC Simulation

If we review our second simulation: Here, two-dimensional parity check code is used. Firstly, "Kaan" message consisting of 4 ASCII characters was created. Encode after the message is prepared. During this coding process, parity row and parity column were added. And while sending the message, an artificial error was created on the channel. An error was detected and corrected in the decoder. As seen in Output, it was decoded and printed on the screen as "Kaan" again

```
%% TWO DIMENSIONAL PARITY CHECK CODE
    clear;
    %MESSAGE and PREPARING
    disp("Program Starting")
    message = 'Kaan'; %Input Message
    disp(message)
    %Resize
    binary_message = dec2bin(message);
    msg = reshape(binary_message,[],1);
    msg = str2num(msg);
    msg = reshape(msg,[],7);
    len_row = length(msg(:,1));
    len_column = length(msg);
    error_row = 0;
    error_column = 0;
    error_row_sum = 0;
    error_col_sum = 0;

  %ENCODING
   for row=1:(len_row)
       row_sum(row) = mod(sum(msg(row,1:end)),2); %adding parity row
   end
   matrix = [msg row_sum']; %concat parity row
   for col=1:len_column+1
       col_sum(col)= mod(sum(matrix(1:end,col)),2); %adding parity column
   end
   matrix = [matrix;col_sum]; %concat cheksum column
   disp("Code : ")
   disp(matrix)

   %CHANNEL ERROR
   matrix(3,1) = ~matrix(3,1); % add one bit error

   disp("Code with Error Bit : ")
   disp(matrix)
```

Fig. 7: RPC Code Simulation

```
    %%%% FIND-CORRECT ERROR AND DECODING%%%
%find error row
    for row=1:(len_row)
        x = mod(sum(matrix(row,1:end-1)),2);
        if x~= matrix(row,end)
            error_row = row
            error_row_sum = x;
        end
    end

    %find error column
    for col=1:(len_column)
        y = mod(sum(matrix(1:end-1,col)),2);
        if y ~= matrix(end,col)
            error_column = col
            error_col_sum = y ;
        end
    end
%When no error or the only error is the sum of all parity(right-bottom corner)
if error_row == 0 && error_column == 0
        disp("No error or Right Buttom Error")
        matrix(len_row+1,len_column+1) = mod(sum(matrix(len_row+1,1:len_column)),2)
%When the error is parity row
elseif error_row == 0 && error_column ~= 0
    m = 0;
    for row=1:(len_row-1)
        m = matrix(row,error_column)+m
    end
    matrix(end,error_column) = m

%When the error is parity column
elseif error_row ~= 0 && error_column == 0
    n=0;
    for col=1:(len_column-1)
        n = matrix(error_row,col)+n;
    end
    matrix(error_row,end) = n;

else
    matrix(error_row,error_column) = matrix(error_row,error_column)
    /- (error_row_sum -matrix(error_row,end));
end
matrix = mod(matrix,2);

%MESSAGE PREPARING
matrix = matrix(1:end-1,1:end-1);
received_matrix = char(reshape(char('0' + matrix),[],7));
received_message = char(bin2dec(received_matrix));
received_message = reshape(received_message,[],length(message));
disp("Corrected Received Message ")
disp(received_message)
```

Fig. 8: RPC Code Simulation

```
Message :
Kaan
Code :
     1     0     0     1     0     1     1     0
     1     1     0     0     0     0     1     1
     1     1     0     0     0     0     1     1
     1     1     0     1     1     1     0     1
     0     1     0     0     1     0     1     1

Code with Error Bit :
     1     0     0     1     0     1     1     0
     1     1     0     0     0     0     1     1
     0     1     0     0     0     0     1     1
     1     1     0     1     1     1     0     1
     0     1     0     0     1     0     1     1


error_row =

     3


error_column =

     1

Corrected Received Message
Kaan
```

Fig. 9: Output of RPC Code Simulation

It is a 4 * 7 parity check matrix that we observed in Figure. The code rate is 7/10. d_{min} = 4 and the maximum number of errors that can be detected is 3.

REFERENCES

[1] Cihun-Siyong, Alex Gong, Li-Ren Huang, "Two-Dimensional Parity Check with Variable Length Error Detection Code for the Non-Volatile Memory of
Smart Data" Chang Gung University, 24 July 2018

[2] Ali Grami, "Introduction to Digital Communications", 2016

[3] Kyle Jamieson, "Detecting and Correcting Bit Errors", Wireless Networks L8,2015

[4] McGraw-Hill, "Data Communications and Networking Ch-10 Error Detection and Correction",2015