Department of
Computer Science and Engineering

Title: Implementing of Error Detection & Correction Mechanism using Hamming Code and Parity Checker.

Data Communication Lab

CSE 208

Green University of Bangladesh

1 Objective(s)

- To attain knowledge on the parity checking and how it works.
- To attain knowledge on the Hamming code and how it works.
- To implement Hamming code error detection and correction algorithms.

2 Problem Analysis

Error detection and correction is the process of detecting the errors that are present in the data transmitted from transmitter to receiver, in a communication system then correct the error. We use some redundancy codes to detect these errors, by adding to the data while it is transmitted from source (transmitter). These codes are called "Error detecting and correcting codes".

Types of error detection and correcting we will study today:

- Parity Checking
- Hamming Code

2.1 Parity Checking

Parity bit means nothing but an additional bit added to the data at the transmitter before transmitting the data. Before adding the parity bit, number of 1's or zeros is calculated in the data. Based on this calculation of data an extra bit is added to the actual information / data. The addition of parity bit to the data will result in the change of data string size.

This means if we have an 8 bit data, then after adding a parity bit to the data binary string it will become a 9 bit binary data string.

Parity check is also called as "Vertical Redundancy Check (VRC)".

There is two types of parity bits in error detection, they are 1. Even parity and 2. Odd parity

Even Parity If the data has even number of 1's, the parity bit is 0. Ex: data is 10000001 -> parity bit 0 and Odd number of 1's, the parity bit is 1. Ex: data is 10010001 -> parity bit 1.

Odd Parity If the data has odd number of 1's, the parity bit is 0. Ex: data is 10011101 -> parity bit 0 and Even number of 1's, the parity bit is 1. Ex: data is 10010101 -> parity bit 1

2.1.1 Algorithm for Even Parity Checking:

1. Start.
2. Take the data from the user.
3. Count the number of 1's.
4. If the number of 1's is even then append a '1' at the MSB position of the data and print the data. Otherwise, append a '0' at the MSB position of the data and print the data.
5. End.

2.1.2 Implementation of Even Parity Checking:

```
#include<iostream>
using namespace std;
//function for measuring array length with data
int a_length(char array[])
{
    int count = 0;
    for(int i = 0; array[i] != NULL; i++)
    {
        count++;
    }
    return count;
```

```
}

int main()
{

    char data[100];
    cout<<"This is a program for even parity checking."<<endl;
    cout<<"Enter the data: "<<endl;
    //taking user data here
    cin>>data;
    //finding the user data length
    int length= a_length(data);

    int count=0;
    for(int i=0; i<length; i++)
    {
        //checking even parity
        if(data[i]=='1')
        {
            count++;
        }
    }
    //increasing the array for adding the parity bit.
    int c=length+1;
    //new array

    if(count%2 == 0)
    {

        for(int i=c,j=c-1; i>0; i--,j--)
        {
            //copying the data to the new array
            data[i] = data[j];
        }
        //initializing the parity
        data[0]='1';
        //displaying the new array
        cout<<"After adding '1' at the front of the data: "<<endl;
        cout<<data<<endl;
    }
    else
    {
        for(int i=c,j=c-1; i>0; i--,j--)
        {
            //copying the data to the new array
            data[i] = data[j];
        }
        //initializing the parity
        data[0]='0';
        //displaying the new array
        cout<<"After adding '0' at the front of the data: "<<endl;
        cout<<data<<endl;
    }
}
```

2.1.3 Output for the code of Even Parity Checking:

```
This is a program for even parity checking.
```

```
Enter the data:
1110001
After adding '1' at the front of the data:
11110001
```

2.2 Hamming Code

Hamming code is a set of error-correction codes that can be used to detect and correct the errors that can occur when the data is moved or stored from the sender to the receiver. It is technique developed by R.W. Hamming for error correction.

Redundant bits are extra binary bits that are generated and added to the information-carrying bits of data transfer to ensure that no bits were lost during the data transfer.

The number of redundant bits can be calculated using the following formula:

$$2^r \geq m + r + 1 \tag{1}$$

where, r = redundant bit, m = data bit.

Suppose the number of data bits is 7, then the number of redundant bits can be calculated using:

$$r = 2^4 \geq 7 + 4 + 1 \tag{2}$$

Thus, the number of redundant bits= 4

2.2.1 A General Algorithm of Hamming code:

The Hamming Code is simply the use of extra parity bits to allow the identification of an error.

1. Write the bit positions starting from 1 in binary form (1, 10, 11, 100, etc).
2. All the bit positions that are a power of 2 are marked as parity bits (1, 2, 4, 8, etc).
3. All the other bit positions are marked as data bits.
4. Each data bit is included in a unique set of parity bits, as determined its bit position in binary form.
   (a) Parity bit 1 covers all the bits positions whose binary representation includes a 1 in the least significant position (1, 3, 5, 7, 9, 11, etc).
   (b) Parity bit 2 covers all the bits positions whose binary representation includes a 1 in the second position from the least significant bit (2, 3, 6, 7, 10, 11, etc).
   (c) Parity bit 4 covers all the bits positions whose binary representation includes a 1 in the third position from the least significant bit (4–7, 12–15, 20–23, etc).
   (d) Parity bit 8 covers all the bits positions whose binary representation includes a 1 in the fourth position from the least significant bit bits (8–15, 24–31, 40–47, etc).
   (e) In general, each parity bit covers all bits where the bit-wise AND of the parity position and the bit position is non-zero.
5. Since we check for even parity set a parity bit to 1 if the total number of ones in the positions it checks is odd.
6. Set a parity bit to 0 if the total number of ones in the positions it checks is even.

2.2.2 Approach and Example of Hamming code:

Hamming code uses redundant bits (extra bits) which are calculated according to the below formula:-

$$2^r \geq m + r + 1 \tag{3}$$

Where r is the number of redundant bits required and m is the number of data bits.
R is calculated by putting r = 1, 2, 3 . . . until the above equation becomes true.
R1 bit is appended at position 2^0
R2 bit is appended at position 2^1

R3 bit is appended at position 2^2 and so on.

These redundant bits are then added to the original data for the calculation of error at receiver's end. At receiver's end with the help of even parity (generally) the erroneous bit position is identified and since data is in binary we take complement of the erroneous bit position to correct received data.

Respective index parity is calculated for r1, r2, r3, r4 and so on.

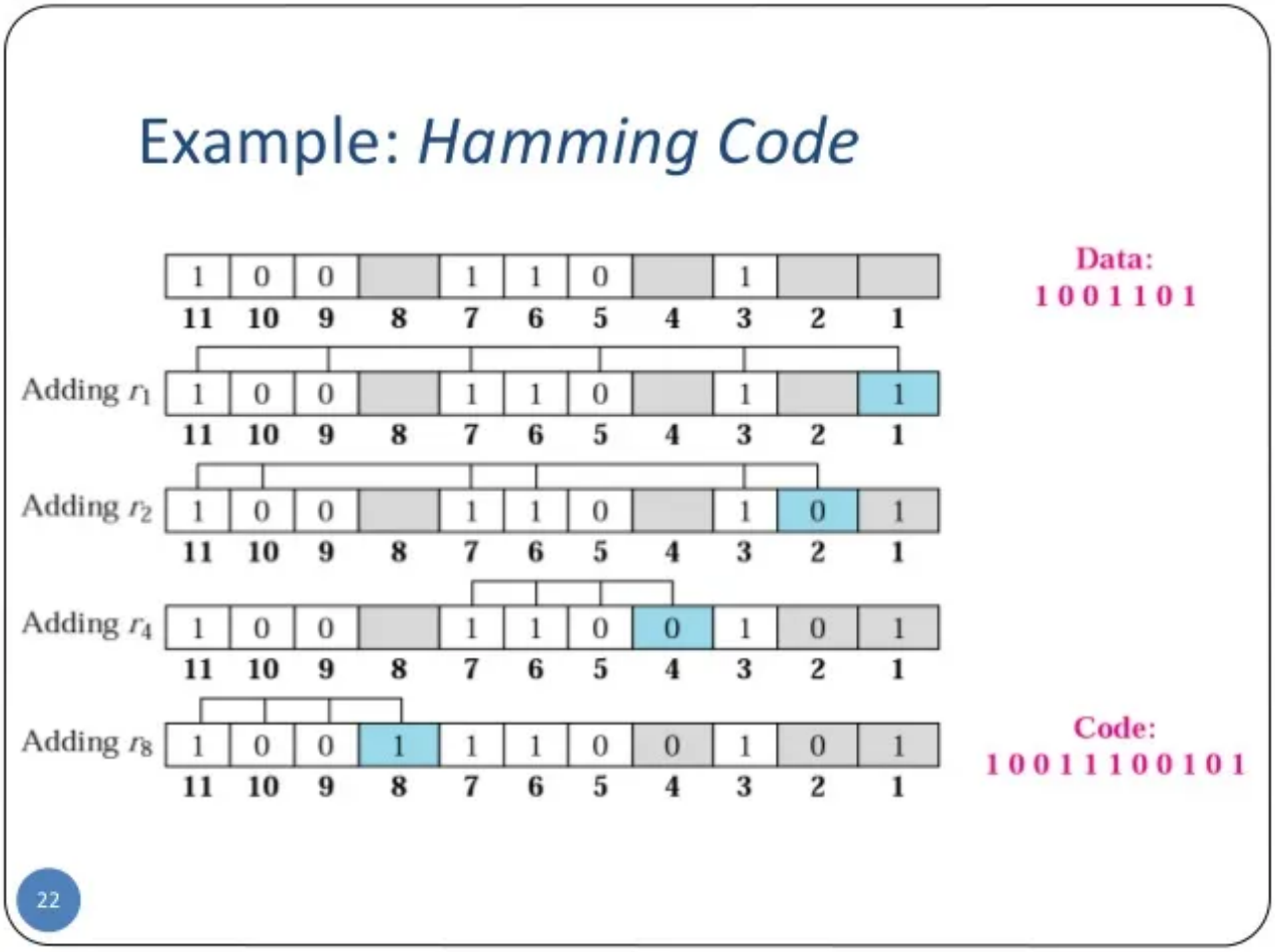

Figure 1: Example of Hamming Code.

2.2.3 Implementation of Hamming Code for Even Parity Checking:

```
#include <math.h>
#include <stdio.h>

// Store input bits
int input[32];

// Store hamming code
int code[32];

int ham_calc(int, int);
void solve(int input[], int);

// Function to calculate bit for
// ith position
int ham_calc(int position, int c_l)
{
```

```
    int count = 0, i, j;
    i = position - 1;

    // Traverse to store Hamming Code
    while (i < c_l) {
        for (j = i; j < i + position; j++) {

            // If current boit is 1
            if (code[j] == 1)
                count++;
        }

        // Update i
        i = i + 2 * position;
    }
    if (count % 2 == 0)
        return 0;
    else
        return 1;
}

// Function to calculate hamming code
void solve(int input[], int n)
{
    int i, p_n = 0, c_l, j, k;
    i = 0;

    // Find msg bits having set bit
    // at x'th position of number
    while (n > (int)pow(2, i) - (i + 1)) {
        p_n++;
        i++;
    }

    c_l = p_n + n;
    j = k = 0;

    // Traverse the msgBits
    for (i = 0; i < c_l; i++) {

        // Update the code
        if (i == ((int)pow(2, k) - 1)) {
            code[i] = 0;
            k++;
        }

        // Update the code[i] to the
        // input character at index j
        else {
            code[i] = input[j];
            j++;
        }
    }

    // Traverse and update the
    // hamming code
    for (i = 0; i < p_n; i++) {
```

```
        // Find current position
        int position = (int)pow(2, i);

        // Find value at current position
        int value = ham_calc(position, c_l);

        // Update the code
        code[position - 1] = value;
    }

    // Print the Hamming Code
    printf("\nThe generated Code Word is: ");
    for (i = 0; i < c_l; i++) {
        printf("%d", code[i]);
    }
}

// Driver Code
void main()
{
    // Given input message Bit 0111
    input[0] = 0;
    input[1] = 1;
    input[2] = 1;
    input[3] = 1;

    int N = 4;

    // Function Call
    solve(input, N);
}
```

2.2.4 Output for the code of Even Parity Checking:

```
The generated Code Word is: 0001111
```

3 Discussion & Conclusion

Based on the focused objective(s) to understand about the Hamming code, the additional lab exercise made me more confident towards the fulfilment of the objectives(s).

4 Lab Task (Please implement yourself and show the output to the instructor)

Given a generated hamming code as user input, write a program to detect if there were any errors in transmission and correct if there are any errors.

5 Lab Exercise (Submit as a report)

.Given a message bitstream(the message can be of any length) as user input, write a program to first calculate the number of parity bits needed and with that, generate the hamming code.

6 Policy

Copying from internet, classmate, seniors, or from any other source is strongly prohibited. 100% marks will be *deducted* if any such copying is detected.