NETWORK LAB REPORT

ASSIGNMENT- 1

JADAVPUR UNIVERSITY

NAME – SOULIB GHOSH

SECTION – A2

CLASS - BCSE – III

ROLL – 0016105010 47

Deadline - 31st January

Submission – 31st January

Problem Statement

Design and implement an error detection module which has four schemes namely LRC, VRC, Checksum and CRC. Please note that you may need to use these schemes separately for other applications (assignments). You can write the program in any language. The Sender program should accept the name of a test file (contains a sequence of 0, 1) from the command line. Then it will prepare the data frame (decide the size of the frame) from the input. Based on the schemes, code word will be prepared. Sender will send the code word to the Receiver. Receiver will extract the data word from code word and show if there is any error detected. Test the same program to produce a PASS/FAIL result for following cases.

- Error is detected by all four schemes. Use a suitable CRC polynomial.
- Error is detected by checksum but not by CRC.
- Error is detected by VRC but not by CRC.

[Note: Inject error in random positions in the input data frame. Write a separate method for that.]

Introduction:

The error detection and correction mechanism is implemented in the data link layer. There are many noises in the medium that may be wired or wireless. There can be thermal noise, induced noise, and crosstalk between two wires or impulse noise. Due to these noises there is high chances that the send data packets may damage when it reach the receiver. These error detection mechanisms help to verify that the send data packets are correctly reached to the receiver or not. To solve this four popular error detection mechanisms are considered – LRC, VRC, CRC and Checksum. In this work only error detection is focused, error correction is ignored. The main idea behind all the methods is that we add some redundant bits with the data word to form the code word. Those redundant bits will carry some additional information about the data word. The receiver end will receive the code word and process that. The additional information encoded in those redundant bits will help the receiver to determine that weather correct data word has arrived or not. If the received data word is perfect then receiver process the data else the receiver request the sender to resend the current data frame.

Proposed Approach:

The four methods LRC, VRC, CRC and checksum are implemented according to their rule. There are some assumption which are taken:

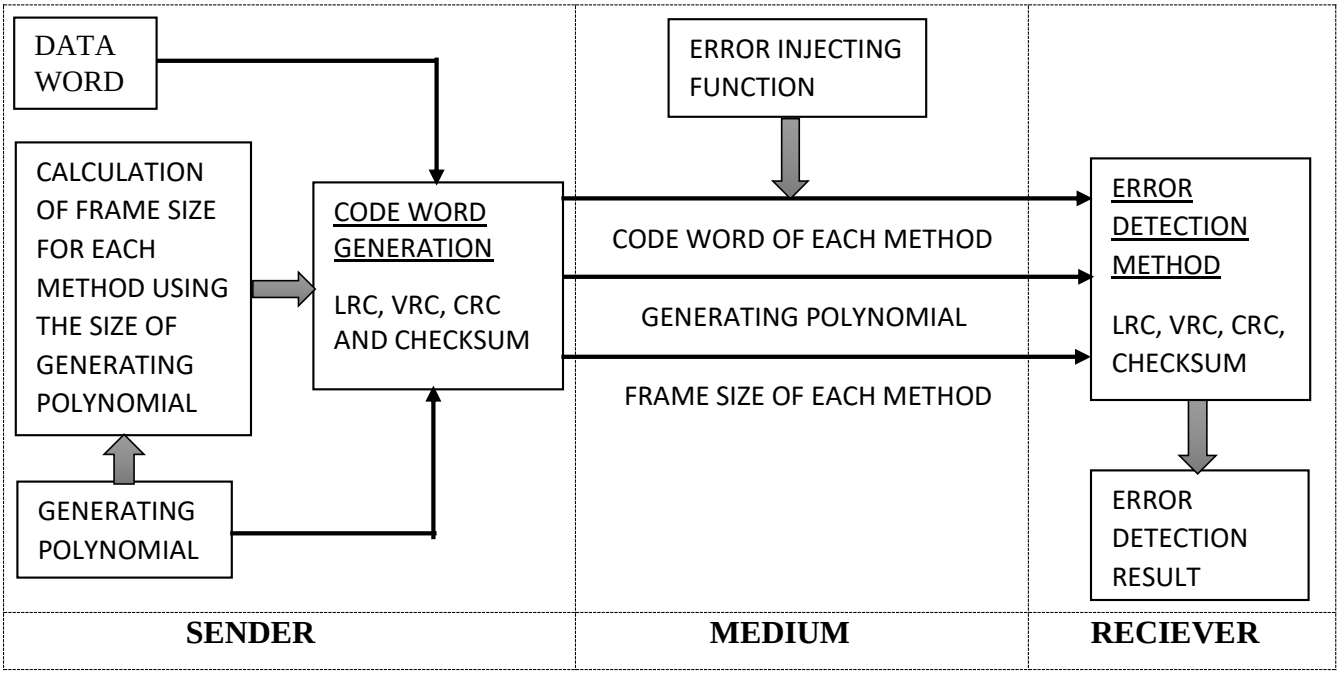
1) The input data word is taken as a sequence of 0 or 1. The bit stream is kept in a csv file.
2) As we need to simulate the entire process in a single machine, there will be no medium possible. To create the medium the code word is stored in an intermediate csv file. Then the csv file is read by the receiver. This intermediate file work as a medium. All the error generating methods are applied on the csv files containing code word.
3) In original case the generating polynomial and the frame size is pre-defined for a sender and receiver pair. In the proposed approach, the generating polynomial and the frame size is sent by the sender to the receiver so that receiver can use these values to detect errors. The main assumption is that error can only be injected in the code word. The generating polynomial and the frame size is correctly transferred all the time.
4) We need to analyze the error detection capabilities of the four methods. It is required that the size of the code word is same for all the four methods. To make the code word length same, in some cases padding is done with 0 (0 is added to the left to keep the value same).

After assuming these, in the sender end the code word generating procedure for LRC, VRC, CRC and checksum is implemented. There is also a mechanism in the sender side to calculate the frame size for each method.

The sender successfully send the frame size and the generating polynomial to the receiver end. The code word is stored in an intermediate file. All the error inject mechanism are done on those intermediate files.

The receiver end read those intermediate files and perform error detection for all the four above mentioned method. Then the receiver tells us that which methods have successfully detected the error if there is an error.

Diagram of the Proposed Approach:



Input - Output:

In this proposed method the input is a bit stream containing 0 and 1 stored in a csv file. The output is a message stating that which method has detected error successfully (if there is an error). The intermediate code words are stored in csv files which are considered as medium.

CODE:

1) Sender Side:

In the sender side we have the data word which is to send in the receiver side after adding appropriate redundant bits. The sender side contains code word generation of four error detection methods – checksum, LRC, VRC and CRC. The methods are called one after another to generate the code word for each methods.

```
data = csvread('data.csv');
cs_size = 9;
data = pad_data(data,cs_size);
%checksum
cs_size = 9;
final = data_to_mat(data,cs_size);
CSdata = pad_data(data,cs_size);
[CheckSumToBeTransmitted] = csf(cs_size,CSdata);
mat_CS = [final; CheckSumToBeTransmitted];
code_word_CS = [CSdata CheckSumToBeTransmitted];
%VRC
vrc_size = 9;
```

```
[mat_VRC, code_word_VRC] = VRC_generator(data, vrc_size);
%LRC
lrc_size = 1;
[mat_LRC,code_word_LRC] = LRC_gen_main(data, lrc_size);
%CRC
G = [1 0 1 1];
crc_size = 3;
[mat_CRC,code_word_CRC] = CRC_gen_main(data, crc_size, G);
% codeword
csvwrite('LRC.csv',code_word_LRC);
csvwrite('VRC.csv',code_word_VRC);
csvwrite('CRC.csv',code_word_CRC);
csvwrite('CS.csv',code_word_CS);
fprintf("The send code word is generated. Code word is stored in the CSV.\n");
```

2) Receiver Side:

The receiver will get the code word from the medium. The main purpose of the receiver side is to determine whether there is error in the send data word or not using the redundant bits. The receiver side contains error uncovering functions of the four error detection methods – Checksum, LRC, VRC and CRC. The sender side delivers the information that if there is an error or not. If there is an error then it also provides us which method has successfully detected the error.

```
code_word_LRC = csvread('LRC.csv');
code_word_VRC = csvread('VRC.csv');
code_word_CS = csvread('CS.csv');
code_word_CRC = csvread('CRC.csv');
[~,error_VRC] = VRC_detector(code_word_VRC, vrc_size);
[error_CS] = cs_detector(cs_size,code_word_CS);
[error_LRC] = LRC_detector_main(code_word_LRC,lrc_size);
[error_CRC] = CRC_detector_main(code_word_CRC, crc_size, G);
error = 0;
if error_VRC ~= 0
    fprintf("Error in VRC\n");
    error = 1;
end
if error_LRC ~= 0
    fprintf("Error in LRC\n");
    error = 1;
end
if error_CRC ~= 0
    fprintf("Error in CRC\n");
    error = 1;
end
if error_CS ~= 0
    fprintf("Error in CS\n");
    error = 1;
end
if error == 0
    fprintf("No Error Detected\n");
end
```

3) Code Word generation of LRC:

This function takes two arguments – data word and frame size. The data word is broken into frames of the specified frame size. After that LRC method is applied to generate the code word. This method finally gives the code word as output.

```
function [fin,cw] = LRC_gen_main(data, frame_size)
[final] = data_to_mat(data,frame_size);
[r,~] = size(final);
fin = [];
cw = [];
for i = 1:r
    c = final(i,:);
    x = LRC_generator(c);
    fin = [fin; x];
end
[r,~] = size(fin);
for i = 1:r
    cw = [cw fin(i,:)];
end
end
```

```
function [data] = LRC_generator(data)
if mod(sum(data),2) ~= 0
    data = [data 1];
else
    data = [data 0];
end
end
```

4) Code Word Generation of VRC:

This function takes two arguments – data word and frame size. The data word is broken into frames of the specified frame size. After that VRC method is applied to generate the code word. This method finally gives the code word as output.

```
function [final, code_word] = VRC_generator(data, frame_size)
final = data_to_mat(data,frame_size);
[r,c] = size(final);
for i = 1:c
    ss = sum(final(:,i));
    if mod(ss,2) == 1
        final(r+1,i) = 1;
    else
        final(r+1,i) = 0;
    end
end
code_word = [];
for i = 1:(r+1)
    code_word = [code_word final(i,:)];
end
end
```

```
function [final] = data_to_mat(data,frame_size)
l = length(data);
res = mod(l,frame_size);
if res ~= 0
    x = zeros([1 (frame_size-res)]);
    data = [x data];
end
l = length(data);
num_frames = l/frame_size;
final = [];
for i = 1:num_frames
    x = frame_size*(i-1) + 1;
```

```
    y = frame_size*i;
    final = [final; data(x:y)];
end
end
```

5) Code Word Generation of CRC:

This method take data word, CRC frame size and generating polynomial as input and gives the final code word generated by that polynomial as output.

```
function [fin,cw] = CRC_gen_main(data, frame_size, G)
[final] = data_to_mat(data,frame_size);
[r,~] = size(final);
fin = [];
cw = [];
for i = 1:r
    c = final(i,:);
    msg_tx = CRC_generator(G,c);
    fin = [fin; msg_tx];
end
[r,~] = size(fin);
for i = 1:r
    cw = [cw fin(i,:)];
end
end
```

```
function [msg_tx] = CRC_generator(G,data)
len_G=length(G);
frame = length(data);
checks = 1;
msg_bits=frame/checks;
y=msg_bits;
z=1;
for p=1:checks
    msg_in=data(z:y);
    len_msg=length(msg_in);
    for k=1:len_G-1
        msg_in(len_msg+k)=0;
    end
    msg=msg_in;
    check=zeros;
    for k=1:len_msg
        if G(1)==msg(1)
            for m=1:len_G
                msg(m)=xor(msg(m),G(m));
            end
        end
        msg=circshift(msg,[0 -1]);
    end
    msg_in(len_msg+1:end)=msg(1:len_G-1);
    msg_tx=msg_in;
end
end
```

6) Code Word Generation of Checksum:

This function takes all the code word size and the data as input and gives the code word as output. The code word contains only the checksum. After that the supplied data is to be merged with the generated sum to get the final code word.

```
function [CheckSumToBeTransmitted] = checkSumGenerator(wordSize,Din)
[m, n]=size(Din);
N=n/wordSize;
copy=zeros(N,wordSize);
for i=0:N-1
    for j=1:wordSize
     copy(i+1,j)=Din(wordSize*i+j);
    end
end
checksum=0;
for k=1:N
    checksum=checksum+bin2dec(num2str(copy(k,:)));
end
d=checksum;
re=zeros(1,N*wordSize);
for i=1:N*wordSize
    re(i)=mod(d,2);
    d=d-re(i);
    d=d/2;
end
 re=fliplr(re);
 wrap=zeros(N,wordSize);
for i=0:N-1
    for j=1:wordSize
     wrap(i+1,j)=re(wordSize*i+j);
    end
end
checksumm=0;
for k=1:N
    checksumm=checksumm+bin2dec(num2str(wrap(k,:)));
end
d=checksumm;
re=zeros(1,wordSize);
for i=1:wordSize
    re(i)=mod(d,2);
    d=d-re(i);
    d=d/2;
end
 re=fliplr(re);
 CheckSumToBeTransmitted=~re;
end
```

7) Error Detection for LRC:

This function takes the code word and the frame size as input and returns a Boolean value to tell that whether there is an error or not.

```
function [error] = LRC_detector_main(data, frame_size)
[final] = data_to_mat(data,frame_size+1);
[r,~] = size(final);
error = 0;
for i = 1:r
    c = final(i,:);
    err = LRC_detector(data);
    if err ~= 0
        error = 1;
        break;
    end
end
end
```

```
function [error] = LRC_detector(data)
ss = mod(sum(data(1:(length(data)-1))),2);
error = 0;
if (ss == 1 && data(length(data)) == 0) || (ss == 0 && data(length(data)) == 1)
    error = 1;
end
end
```

8) Error Detection for VRC:

This function takes the code word and the frame size as input and returns a Boolean value to tell that whether there is an error or not.

```
function [matt,error] = VRC_detector(code_word, frame_size)
l = length(code_word);
r = l/frame_size;
matt = [];
for i = 1:r
    x = frame_size*(i-1) + 1;
    y = frame_size*i;
    matt = [matt; code_word(x:y)];
end
[r,c] = size(matt);
error = 0;
for i = 1:c
    ss = sum(matt(:,i)) - matt(r,i);
    if mod(ss,2) == 1 && matt(r,i) == 0
        error = 1;
    end
    if mod(ss,2) == 0 && matt(r,i) == 1
        error = 1;
    end
end
end
```

9) Error Detection for Checksum:

The function takes the code word and the frame size as input and returns a Boolean value to detect the error.

```
function [s] = cs_detector(wordSize,code_word)
Din = code_word;
[m, n]=size(Din);
N=n/wordSize;
copy=zeros(N,wordSize);
for i=0:N-1
    for j=1:wordSize
     copy(i+1,j)=Din(wordSize*i+j);
    end
end
checksum=0;
for k=1:N
    checksum=checksum+bin2dec(num2str(copy(k,:)));
end
d=checksum;
re=zeros(1,N*wordSize);
for i=1:N*wordSize
    re(i)=mod(d,2);
    d=d-re(i);
    d=d/2;
```

```
end
 re=fliplr(re);
 wrap=zeros(N,wordSize);
for i=0:N-1
    for j=1:wordSize
     wrap(i+1,j)=re(wordSize*i+j);
    end
end
checksumm=0;
for k=1:N
    checksumm=checksumm+bin2dec(num2str(wrap(k,:)));
end
d=checksumm;
re=zeros(1,wordSize);
for i=1:wordSize
    re(i)=mod(d,2);
    d=d-re(i);
    d=d/2;
end
 re=fliplr(re);
 CheckSumToBeTransmitted=~re;
 ss = sum(CheckSumToBeTransmitted);
 if ss == 0
     s = 0;
 else
     s = 1;
 end
end
```

10) Error Detection for CRC:

The function takes code word, frame size and the generator polynomial as input and outputs TRUE if there is error or FALSE if there is no error.

```
function [error] = CRC_detector_main(data, frame_size, G)
[final] = data_to_mat(data,frame_size+length(G)-1);
[r,~] = size(final);
error = 0;
for i = 1:r
    c = final(i,:);
    err = CRC_detector(G,c);
    if err ~= 0
        error = 1;
        break;
    end
end
end
```

```
function [final] = CRC_detector(G,input_codeword)
checks = 1;
count_noerror=0;count_error=0;
len_msg = length(input_codeword) - length(G) + 1;
len_G = length(G);
for p=1:checks
    msg=input_codeword;
    for k=1:len_msg
        if G(1)==msg(1)
            for m=1:len_G
                msg(m)=xor(msg(m),G(m));
            end
```

```
        end
        msg=circshift(msg,[0 -1]);
    end
    rem=msg;
    if rem==0
        count_noerror=count_noerror+1;
    else
        count_error=count_error+1;
    end
end
final = count_error;
end
```

11) Error Generating Function:

This function is used to generate error. It takes the positions of the code word in an array and the code word as input. The respective bits specified in the array is reversed to generate the error. The content of the array can be dynamic or user defined.

```
function [code_word] = error_geenerat(arr, code_word)
l = length(arr);
for i = 1:l
    code_word(arr(i)) = ~code_word(arr(i));
end
end
```

Strength of the Method:

1) The sender side contains all the four code word generation module separately. Any subset of the four methods can be used very easily without changing much. If we want to ignore any method we just need to comment out the line in which that method is called.
2) Similarly for the receiver side, for error detection method any subset of the four methods can be used. We just required to comment out the line to ignore that method.
3) In CRC, the generating polynomial is taken as an array. Any polynomial can be given as generating as polynomial. To change the degree of the polynomial we just need to update the array.
4) In CRC the entire process is carried out using array operations. So, the size of number or polynomial is not an issue. Any higher degree polynomial can be used easily.
5) There is a separate function for error detection. The function can be changed accordingly to test various inputs.

Limitations of the Method:

LRC:

1) If two or even number of bits in exactly the same positions in another data frame are also damaged, the LRC checker will not detect an error.

VRC:

1) If two or any even number of bits in one data frame are damaged, the VRC checker will fail to detect the error.

Checksum:

1) If any two data frames are swapped with respect to positions then checksum fails to detect error.
2) If two data frames are damaged such that one is increased and other is decreased is by same amount. As a result total sum remain constant. In this case checksum fails to detect errors.
3) Let the code word length of checksum is m. If the error occurred such that the sum is increased or decreased by a multiple of (2^m – 1) then error detection by checksum is not possible.

CRC:

1) If the error occurred in such a way that when the code word is divided by the generator, the remainder remain zero then it is not possible for CRC to detect those types of error.

Test cases:

To check the performance of the proposed approach, we can use a random algorithm to generate error in random position of the code word. But to satisfy the condition mentioned in the question randomized algorithm will not serve the purpose. We need to generate error specifically. The error generation process for the following schemes are as follows.

1) Error is detected by all four schemes. Use a suitable CRC polynomial.

For this only 1 bit from each frame is reversed. If one bit is reversed then error will be definitely detected by VRC. As the bit of same position is changed error will also get detected by LRC. Changing one bit will always give a reminder for any good generating polynomial so error is also detected by CRC. One bit change will also detect error by checksum.

2) Error is detected by checksum but not by CRC.

Suppose the frame size of the code word is m. CRC generating polynomial is G. Let us take G’ of size m generated by zero padding on the left size. If we XOR the code word and G’ and get m’. Now if an error occurred such that the code word becomes m’ from m then the error will not be detected by CRC but detected by checksum. In this way the remainder occurred after dividing m’ by G will always be 0.

3) Error is detected by VRC but not by CRC.

Suppose the frame size of the code word is m. CRC generating polynomial is G. Let us take G’ of size m generated by zero padding on the left size. If we XOR the code word and G’ and get m’. Now if an error occurred such that the code word becomes m’ from m then the error will not be detected by CRC but detected by checksum. In this way the remainder occurred after dividing m’ by G will always be 0.

Comments:

The lab assignment is moderate. The main thing which I learn from this assignment is how to simulate the error detection mechanism. The assignment would be more interesting if we can run the sender and receiver in separate machine and communicate between two nodes. Then we could get a proper occurrence of noise in the medium.