Experiment 2: Error Detection using CRC-CCITT

Aim: To apply CRC (CCITT Polynomial) for error detection

Objective: After carrying out this experiment, students will be able to:

- Apply CRC CCITT to develop codes for error detection
- Analyze how this CRC is able to detect bit errors irrespective of their length and position in the data

Problem statement: You are required to write a program that uses CRC to detect burst errors in transmitted data. Initially, write the program using the CRC example you studied in class. Your final program should ask the user to input data and choose a generator polynomial from the list given in the figure below. Your program is required to calculate the checksum and the transmitted data. Subsequently, the user enters the received data. Applying the same generator polynomial on the received data should result in a remainder of 0.

Name	Polynomial	Application
CRC-8	$x^8 + x^2 + x + 1$	ATM header
CRC-10	$x^{10} + x^9 + x^5 + x^4 + x^2 + 1$	ATM AAL
CRC-16	$x^{16} + x^{12} + x^5 + 1$	HDLC
CRC-32	$x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$	LANs

Analysis: While analyzing your program, you are required to address the following points:

- How is this method different from 2D parity scheme that you have implemented previously?
- What are the limitations of this method of error detection?

MARKS DISTRIBUTION

Component	Maximum Marks	Marks Obtained
Preparation of Document	7	
Results	7	
Viva	6	
Total	**20**	

Submitted by: K Srikanth

Register No: 17ETCS002124

1. Algorithm/Flowchart

1. **Start**
2. **Declaring Variables**
3. **Input the choice**
4. **Case 1: Redundancy generation (Bitstream to be transmitted)**
    1. Input the bitstream:
    2. N=size of bitstream
    3. If N>9 then choose generator polynomial for ATM header
       Else If N>11 then choose generator polynomial for ATM AAL
       Else If N>17 then choose generator polynomial for HDLC
       Else If N>33 then choose generator polynomial for LANs
       Else print "Invalid input"
    4. R=Selected polynomial
    5. L=length(R)
    6. Append L-1 zeroes to M and store it in M
    7. T=M
    8. I=0
    9. While(T[I+L] is not end of line)
       If T[I] is '1' then T [I: I+L] XOR (T [I: I+L], R)
    10. Output Remainder = T [I: I+L]
    11. Output Message = XOR (M, T)
5. **Case 2: Redundancy Check (Bitstream received)**

    1. Input the received bitstream.
    2. Select the generating polynomial
    3. R=Selected polynomial
    4. L=length(R)
    5. I=0
    6. While (T[I+L] is not end of line)
        a. If T[I] is '1' then T [I: I+L] XOR (T [I: I+L], R)
    7. Remainder = T [I: I+L]
    8. If **Remainder** is 00000... then output "Error Free"
    9. Else print **ERROR**= Remainder

6. **Stop**

2. Program

C Program

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
// 17ETCS002124 K Srikanth
int main(int argc, char** argv) {
    char m[50],r[50],t[100],c[100];
    int n,l;
    int i=0,j,count,choice;
    printf("******** MENU *********\n");
    printf("\n1.Redundancy Generation\n2.Redundancy Check\nEnter your choice:");
    scanf("%d",&choice);
    printf("\n");
    switch(choice){
        case 1:
            printf("Input the bitstream: ");
            scanf("%s",&m);
            n=strlen(m);
            if(n>33){strcpy(r,"100000100110000010001110110110111");}
            else if(n>17){strcpy(r,"10001000000100001");}
            else if(n>11){strcpy(r,"11000110101");}
            else if(n>9){strcpy(r,"100000111");}
            else{
                printf("please enter a bit stream whose length is greater than generator polynomial\n");
                exit(0);
            }
```

Figure 1 C Program Code for the given problem statement

```c
            strcpy(t,m);
            l=strlen(r)-1;
            while(i<l){
                t[n+i]='0';
                i++;
            }
            strcpy(c,t);
            t[n+i]='\0';
            i=0;
            n=strlen(t);
            while(t[i+l]!='\0'){
                if(t[i]=='1'){
                    for(j=0;j<=l;j++){
                        if(t[i+j]==r[j]){
                            t[i+j]='0';
                        }else{
                            t[i+j]='1';
                    }}}
                    i++;
            }
            printf("Remainder: %s\n",t);
            n=strlen(t);
            for(j=0;j<n;j++){
                        if(t[j]==c[j]){
                            t[j]='0';
                        }else{
                            t[j]='1';
                }
                }
            printf("Message: %s\n",t);
            break;
```

Figure 2 C Program Code for the given problem statement. Continued

```
58          case 2:
59              printf("Input the bitstream: ");
60              scanf("%s",&m);
61              printf("Generating Polynomial:\n");
62              printf("1.100000111\n");
63              printf("2.11000110101\n");
64              printf("3.10001000000100001\n");
65              printf("4.10000010011000001000111011011O111\n");
66              scanf("%d",&choice);
67              n=strlen(m);
68              switch(choice){
69                  case 1:strcpy(r,"100000111");break;
70                  case 2:strcpy(r,"11000110101");break;
71                  case 3:strcpy(r,"10001000000100001");break;
72                  case 4:strcpy(r,"10000010011000001000111011011O111");break;
73                  default:printf("invalid choice\n");
74              }
75              strcpy(t,m);
76              n=strlen(t);
77              i=0;
78              l=strlen(r)-1;
79              while(t[i+l]!='\0'){
80                  if(t[i]=='1'){
81                      for(j=0;j<=l;j++){
82                          if(t[i+j]==r[j]){
83                              t[i+j]='0';
84                          }else{
85                              t[i+j]='1';
86                  }}}
87                      i++;
88              }
89              for (i = 0; i < n; i++) {
90                  if(t[i]=='1')count++;
91              }
92              if(count>0)printf("Error: %s\n",t);
93              else printf("Error free-> remainder:%s\n",t);
94              break;
95          default:
96              printf("Invalid choice");
97      }
98      return (EXIT_SUCCESS);
99  }
100
```

Figure 3 C Program Code for the given problem statement... Continued

3. Results

```
******** MENU *********

1.Redundancy Generation
2.Redundancy Check
Enter your choice:1

Input the bitstream: 101010101010
Remainder: 000000000000010011111111
Message: 101010101010100111111
Program ended with exit code: 0
```

Figure 4 C Program output for the given problem statement.

```
******** MENU *********

1.Redundancy Generation
2.Redundancy Check
Enter your choice:2

Input the bitstream: 1010101010101010
Generating Polynomial:
1.100000111
2.11000110101
3.10001000000100001
4.10000100110000010001110110110111
1
Error: 0000000011110101
Program ended with exit code: 0
```

Figure 5 4 C Program output for the given problem statement.

4. Analysis and Discussions

Redundancy Generator (Sender side):

- The binary data (message) is first augmented by adding L-1 zeros in the end of the data
- Use modulo-2 binary division to divide binary data by the key and store remainder of division.
- In each step, a copy of the divisor R (or generating polynomial) is XORed with the L bits of the dividend (or message).
- The result of the XOR operation (remainder) is (L-1) bits, which is used for the next step after 1 extra bit is pulled down to make it L bits long.
- When there are no bits left to pull down, we have a result. The (L-1)-bit remainder which is appended at the sender side.

Redundancy check (Receiver side):

- Perform modulo-2 division again and if remainder is 0, then there are no errors.

How is this method different from 2D parity scheme that you have implemented previously?

- In 2D parity in case of burst errors, there is possibility that the error data can be accepted by the receiver as correct data. Thus, 2D parity cannot fully detect burst errors, whereas CRC can successfully detect single bit errors as well as burst errors.
- In CRC, if error occurs, then the remainder at receiver will not be 0 at any case.
- The data is accepted at the receiver only if the remainder obtained after division is 0.

What are the limitations of this method of error detection?

- CRC uses redundant bits for error detection, which makes extra bits getting added hence process is complex for computation.
- Time consuming. As we can see execution time is much higher than that of 2d parity bit error detection's

5. Conclusions

The user is given a choice to enter a bitstream thereby a generating polynomial is chosen by the program itself and producing message to be transmitted. Receiver end tries to decode the message and checks whether the remainder is zero. CRC is easy to detect single bit as well as multiple burst errors. They can easily be implemented on several hardware and software programs.

6. Comments

a. Limitations of the experiment

At the receiver's end automatic generating polynomial selection is not implemented hence pushing the users to select polynomial by themselves.

b. Limitations of the results obtained

The transmitted message has to be input separately instead of directly giving a choice to detect the error after encoding message.

c. Learning

CRC, Burst errors and advantages of CRC over 2d parity