CRC (Cyclic Redundancy Check)

This is also known as a polynomial code. Polynomial codes are based upon treating bit strings as representations of polynomials with coefficients of 0 and 1 only. A k-bit frame is regarded as the coefficient list for a polynomial with k terms, ranging from x^k −1 to x 0. Such a polynomial is said to be of degree k − 1. The high-order (leftmost) bit is the coefficient of x^ −1, the next bit is the coefficient of x^k −2, and so on. For example, 110001 has 6 bits and thus represents a six-term polynomial with coefficients 1, 1, 0, 0, 0, and 1 : 1x^5 + 1x^4 + 0x^3 + 0x^2 + 0x^1 + 1x^0.

```
   10011011     00110011     11110000      01010101
+  11001010   + 11001101   − 10100110   − 10101111
  __________   __________   __________   __________
   01010001     11111110     01010110     11111010
```

When the polynomial code method is employed, the sender and receiver must agree upon a generator polynomial, G(x), in advance. Both the high- and low-order bits of the generator must be 1. To compute the CRC for some frame with m bits corresponding to the polynomial M(x), the frame must be longer than the generator polynomial. The idea is to append a CRC to the end of the frame in such a way that the polynomial represented by the check-summed frame is divisible by G(x). When the receiver gets the check-summed frame, it tries dividing it by G(x). If there is a remainder, there has been a transmission error.

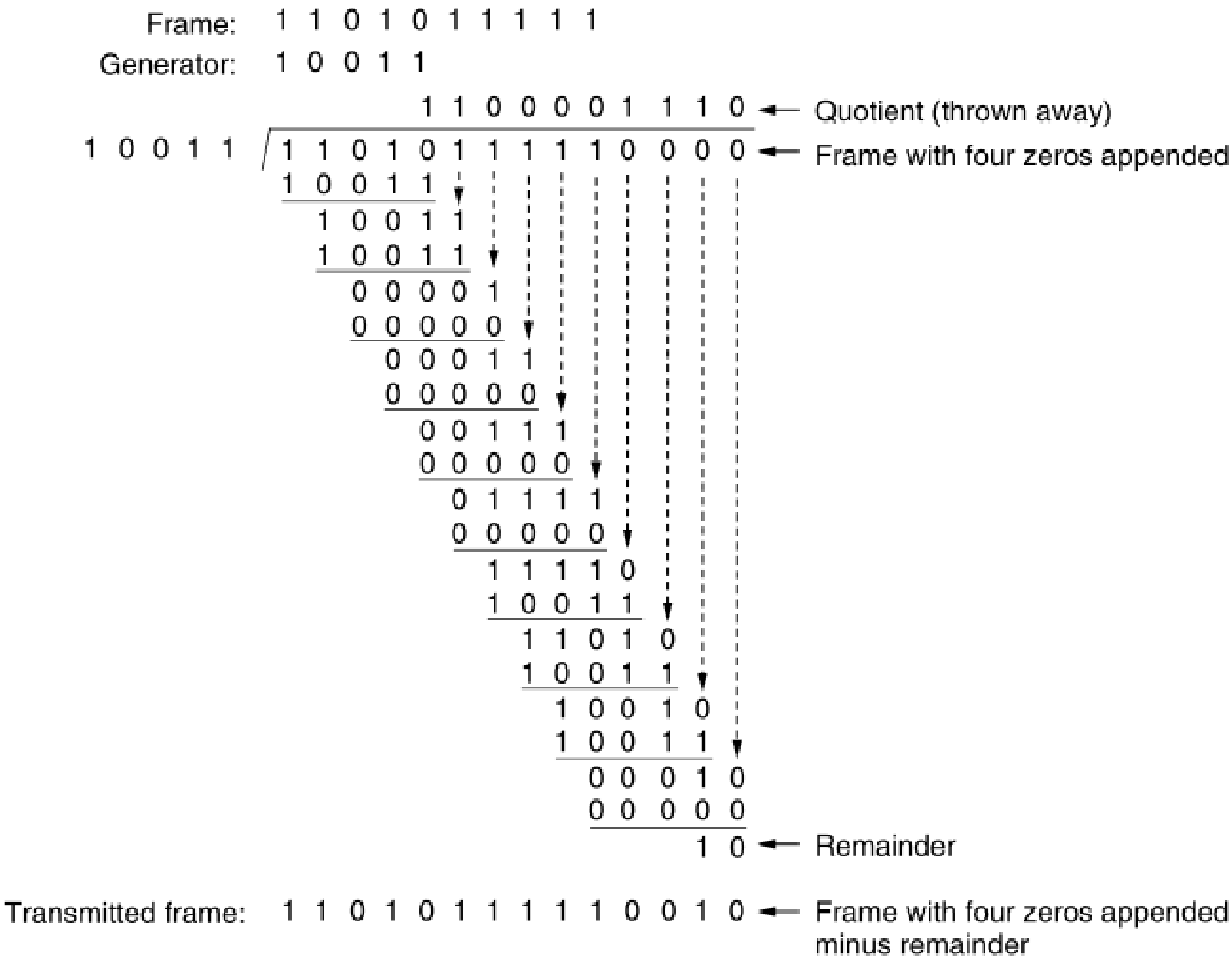
The algorithm for computing the CRC is as follows : -

1. Let r be the degree of G(x). Append r zero bits to the low-order end of the frame so it now contains m + r bits and corresponds to the polynomial x^r M(x).

2. Divide the bit string corresponding to G(x) into the bit string corresponding to x^r M(x), using modulo 2 division.

3. Subtract the remainder (which is always r or fewer bits) from the bit string corresponding to x^r M(x) using modulo 2 subtraction. The result is the checksummed frame to be transmitted. Call its polynomial T(x).

Example calculation of the CRC



Code of Cyclic Redundancy Check

```
#include<iostream>
#include<cstdio>
#define MAX_SIZE 100
using namespace std;
void Scan_Data(bool *Source,int &Source_Size)
{
   // for scanning the Frame and Generator into respective variables
   char c=getchar();
        Source_Size=0;
        while(c!='\n')
```

```
    {
        Source[Source_Size++]=c=='0'?0:1;
        c=getchar();
    }
}

void Display_Result(bool *Source,int Source_Size)
{
    // to display the contents of Frame
    cout<<"\nTransmitted Frame : ";
    for(int i=0;i<Source_Size;i++)
        cout<<Source[i];
    cout<<endl;
}
int main() // 1101011011
{
    bool *Frame,*Generator,*Hold;
    int Frame_Size,Gen_Size,i,j,flag;
    Frame=new bool[MAX_SIZE];
    Hold=new bool[MAX_SIZE];
    Generator=new bool[MAX_SIZE];
    cout<<"\nEnter Frame : ";
    Scan_Data(Frame,Frame_Size);    // Scannimg Frame
    cout<<"\nEnter Generator : ";
    Scan_Data(Generator,Gen_Size);  // Scannimg Generator
    for(int i=0;i<Gen_Size-1;i++)   // insterting Gen_Size-1 0's at end of Frame
        Frame[Frame_Size++]=0;
    for(i=0;Frame[i]!=Generator[0];) // when Hold < Generator
        i++;
    while(i<Gen_Size)   // load Hold with initial data
        Hold[i]=Frame[i++];
    flag=0;
    for(j=Gen_Size;j<Frame_Size;)
    {
        // Display_Result(Hold,Gen_Size);
        int k;
        for(i=flag,k=1;k<Gen_Size;i++)  // XORing Generator with dividend(i.e., Hold)
            Hold[i%Gen_Size]=Hold[(i+1)%Gen_Size]^Generator[k++];
        Hold[i%Gen_Size]=Frame[j++];
        // Display_Result(Hold,Gen_Size);
        while(Hold[flag%Gen_Size]!=Generator[0] && j<Frame_Size) // when Hold <
Generator
```

```
            Hold[flag%Gen_Size]=Frame[j++],flag++;
    }
    for(j=Frame_Size-Gen_Size;j<Frame_Size;flag++) // XORing remainder with
dividend(i.e., Frame)
        Frame[j++]^=Hold[flag%Gen_Size];
    Display_Result(Frame,Frame_Size);   // displaying resulting data
}
```

RESULT : -