1 Objective(s)

- To implement the data link layer framing method bit stuffing.

2 Problem analysis

A technique that allows data frames to contain arbitrary number of bits and allows character codes with arbitrary number of bits per character. **Bit stuffing** is which an zero bit is stuffed after five consecutive ones in the input bit stream. On the other hand, **Bit de-stuffing** is the process of removing the stuffed bit in the output stream.

To provide service to network layer, the data link layer, must use the services provided to it by the physical layer. The bit stream is not guaranteed to be error free. The number of bits received may be less than, equal to, or more than data link layer to detect and, if necessary, correct errors.

The usual approach is for the data link layer to break the bit stream up into discrete frames and compute the checksum for each frame. When a frame arrives at the destination, the checksum is re computed. If the newly computed checksum is different from one contained in the frame, the data link layer knows than an error has occurred and takes steps to deal it.

Each frame begins and ends with a special bit pattern, 01111110.When ever the sender's data link layer encounter five consecutive 1's in the data, it automatically stuffs a 0 bit in to outgoing bit stream. This bit stuffing is analogous to byte stuffing. When ever the receiver sees five consecutive incoming ones, followed by a 0 bit, it automatically de-stuffs the 0 bit.

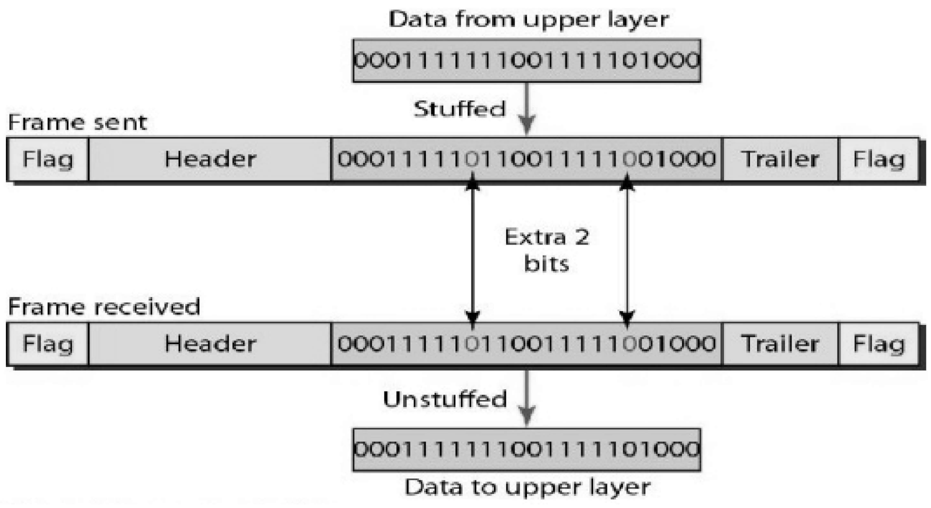

Figure 1: Bit studding and de-stuffing in HDLC frame structure.

3 Algorithm

```
Algorithm 1: Bit stuffing
Input: Given an array, arr[] of size N consisting of 0's and 1's.
/* The idea is to check if the given array consists of 5 consecutive 1's.  Follow the
   steps below to solve the problem:                                                  */
1 Initialize the array brr[] which stores the stuffed array. Also, create a variable count which maintains
   the count of the consecutive 1's.
2 Traverse in a while loop using a variable i in the range [0, N) and perform the following tasks:
     • If arr[i] is 1 then check for the next 4 bits if they are set bits as well. If they are, then insert a 0
       bit after inserting all the 5 set bits into the array brr[].
     • Otherwise, insert the value of arr[i] into the array brr[].
```

```
Algorithm 2: Bit de-stuffing
  Output: Given an array, arr[] of size N consisting of 0's and 1's.
  /* The idea is to check if the given array consists of 5 consecutive 1's.  Follow the
     steps below to solve the problem:                                                  */
     • skip the next bit in the array arr[] in place of inserting a 0 bit in the array brr[].
```

4 Implementation in C

4.1 Bit Stuffing

```c
#include <stdio.h>
#include <string.h>


// Function for bit stuffing
void bitStuffing(int N, int arr[])
{
    int brr[30]; // Stores the stuffed array

    int i, j, k;  // Variables to traverse arrays
    i = 0;
    j = 0;
    int count = 1; // Stores the count of consecutive ones

    // Loop to traverse in the range [0, N)
    while (i < N) {

        // If the current bit is a set bit
        if (arr[i] == 1) {

            brr[j] = arr[i]; // Insert into array brr[]

            // Loop to check for  next 5 bits
            for (k = i + 1; arr[k] == 1 && k < N  && count < 5; k++)
            {
                    j++;
                    brr[j] = arr[k];
                     count++;

                    // If 5 consecutive set bits are found insert a 0 bit
                     if (count == 5) {
                        j++;
                        brr[j] = 0;
                     }
                    i = k;
            }
        }

        // Otherwise insert arr[i] into the array brr[]
        else {
            brr[j] = arr[i];
        }
        i++;
        j++;
    }

    // Print Answer
    for (i = 0; i < j; i++)
        printf("%d", brr[i]);
}

// Driver Code
int main()
{
    int N = 6;
    int arr[] = { 1, 1, 1, 1, 1, 1 };

    bitStuffing(N, arr);

    return 0;
}
```

4.2 Bit De-stuffing

```c
#include <stdio.h>
#include <string.h>

// Function for bit de-stuffing
void bitDestuffing(int N, int arr[])
{
    int brr[30]; // Stores the de-stuffed array

    int i, j, k; // Variables to traverse the arrays
    i = 0;
    j = 0;

    int count = 1; // Stores the count of consecutive ones

    // Loop to traverse in the range [0, N)
    while (i < N) {

        // If the current bit is a set bit
        if (arr[i] == 1) {

            brr[j] = arr[i];  // Insert into array brr[]

            // Loop to check for  the next 5 bits
            for (k = i + 1; arr[k] == 1 && k < N && count < 5; k++) {
                j++;
                brr[j] = arr[k];
                count++;

                // If 5 consecutive set bits are found skip the next bit
                //in arr[]
                if (count == 5) {
                    k++;
                }
                i = k;
            }
        }

        // Otherwise insert arr[i] into the array brr
        else {
            brr[j] = arr[i];
        }
        i++;
        j++;
    }

    // Print Answer
    for (i = 0; i < j; i++)
        printf("%d", brr[i]);
}

// Driver Code
int main()
{
    int N = 7;
    int arr[] = { 1, 1, 1, 1, 1, 0, 1 };

    bitDestuffing(N, arr);

    return 0;
}
```

5 Input/Output (Compilation,Debugging & Testing)

Stuffing:

Input: 111111

Output: 1111101

De-stuffing:

Input: 1111101

Output: 111111

6 Discussion & Conclusion

Based on the focused objective(s) to understand about bit stuffing and de-stuffing, the additional lab exercise made me more confident towards the fulfilment of the objectives(s).

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

- Implement the bit stuffing and de-stuffing together where the system provides a choice to change the transmitted bit stream before de-stuffing.

8 Lab Exercise (Submit as a report)

- Implement a bit Stuffing & De- Stuffing algorithm in C/C++/Java and verify its functionality with a sample input. You are provided with a sample input data sequence, and your task is to stuff it using your algorithm. Additionally, there will be a fixed header and trailer, **"01111110"** at the beginning and end of the data sequence, respectively.

Stuffing-

- ***Sample Input: 01111110 1111111111 01111110***
- ***Sample Output: 01111110 111110111110 01111110***

De Stuffing-

- ***Sample Input: 01111110 111110111110 01111110***
- ***Sample Output: 01111110 1111111111 01111110***

9 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.