**6. WAP to Implement Framing Protocol: Bit Stuffing**

**Steps to Implement Bit Stuffing**

**1. Understand the Concept**

* **Flag Sequence**: A fixed bit pattern (e.g., 01111110) marks the start and end of a frame.
* **Bit Stuffing Rule**:
  + If five consecutive 1s appear in the data, insert a 0 immediately after them.
* **Sender**:
  + Adds the flag at the start and end of the frame.
  + Stuffs a 0 after five consecutive 1s in the data.
* **Receiver**:
  + Detects the flag.
  + Removes the stuffed 0 after every five consecutive 1s.

**2. Key Steps**

1. Read the input data as a binary string.
2. Add flag sequences.
3. Perform bit stuffing (insert a 0 after five 1s).
4. Simulate the receiver removing stuffed bits.

**3. Code Implementation**

Below is an example program in C:

c

Copy code

#include <stdio.h>

#include <string.h>

#define FLAG "01111110"

// Function to perform bit stuffing at the sender's side

void sender(const char \*data, char \*stuffed\_data) {

int count = 0, j = 0;

// Add the flag sequence at the start of the frame

strcpy(stuffed\_data, FLAG);

j += strlen(FLAG);

for (int i = 0; data[i] != '\0'; i++) {

if (data[i] == '1') {

count++;

} else {

count = 0;

}

// Add the current bit to the stuffed data

stuffed\_data[j++] = data[i];

// Stuff a '0' after five consecutive '1's

if (count == 5) {

stuffed\_data[j++] = '0';

count = 0;

}

}

// Add the flag sequence at the end of the frame

strcpy(stuffed\_data + j, FLAG);

j += strlen(FLAG);

stuffed\_data[j] = '\0'; // Null-terminate the stuffed data

}

// Function to perform bit unstuffing at the receiver's side

void receiver(const char \*stuffed\_data, char \*original\_data) {

int count = 0, j = 0;

// Skip the initial flag sequence

int start = strlen(FLAG);

for (int i = start; stuffed\_data[i] != '\0'; i++) {

// Stop at the final flag sequence

if (strncmp(stuffed\_data + i, FLAG, strlen(FLAG)) == 0) {

break;

}

if (stuffed\_data[i] == '1') {

count++;

} else {

count = 0;

}

// Add the current bit to the original data

original\_data[j++] = stuffed\_data[i];

// Skip the stuffed '0' after five consecutive '1's

if (count == 5 && stuffed\_data[i + 1] == '0') {

i++;

count = 0;

}

}

original\_data[j] = '\0'; // Null-terminate the original data

}

int main() {

const char \*data = "0111111011111100001111111"; // Input binary string

char stuffed\_data[100], original\_data[100];

// Step 1: Perform bit stuffing

sender(data, stuffed\_data);

printf("Stuffed Data: %s\n", stuffed\_data);

// Step 2: Perform bit unstuffing

receiver(stuffed\_data, original\_data);

printf("Original Data: %s\n", original\_data);

return 0;

}

**4. Explanation of the Code**

1. **Sender Function**:
   * Adds the flag sequence at the beginning and end of the frame.
   * Scans the input data for five consecutive 1s and inserts a 0 after them.
   * Constructs the stuffed data frame.
2. **Receiver Function**:
   * Skips the initial flag sequence.
   * Reads the frame and reconstructs the original data by removing the stuffed 0 after five consecutive 1s.
   * Stops processing upon detecting the final flag sequence.
3. **Main Function**:
   * Defines an input binary string.
   * Calls the sender to stuff the data.
   * Calls the receiver to unstuff and reconstruct the original data.
   * Displays both stuffed and original data.

**5. Compilation and Execution**

* Compile the program:

bash

Copy code

gcc -o bit\_stuffing bit\_stuffing.c

* Run the program:

bash

Copy code

./bit\_stuffing

**6. Expected Output**

plaintext

Copy code

Stuffed Data: 01111110011111001111101000011111001111110

Original Data: 0111111011111100001111111

**7. Notes**

* The FLAG sequence is predefined as 01111110 but can be customized.
* This implementation assumes the input data is a valid binary string.
* The program simulates a simple sender-receiver communication for bit stuffing.