ASSIGNMENT 2

CSCI 6704 – Advanced Topics in Networks

Dhrumil Amish Shah (B00857606)
dh416386@dal.ca

Question 1 <Digital Encoding Question>

The following bit stream is to be digitally encoded:
1 0 1 1 0 1 0 1 0 1 1 1
Draw the waveforms if the bit stream were to be encoded using
a) Unipolar
b) NRZ
c) Manchester
d) Differential Manchester encoding: Here's how this scheme works. A logic 0 is represented by a transition at the beginning AND at the middle of the clock interval. The transition can be from low to high or high to low, that is, if it was low, it goes to high and if it was high, it goes to low. Logic 1 is represented by a transition ONLY at the middle of the interval. Again, the transition can be either low to high or high to low.
In all the above cases, assume that the signal is HIGH to begin with.

Answer

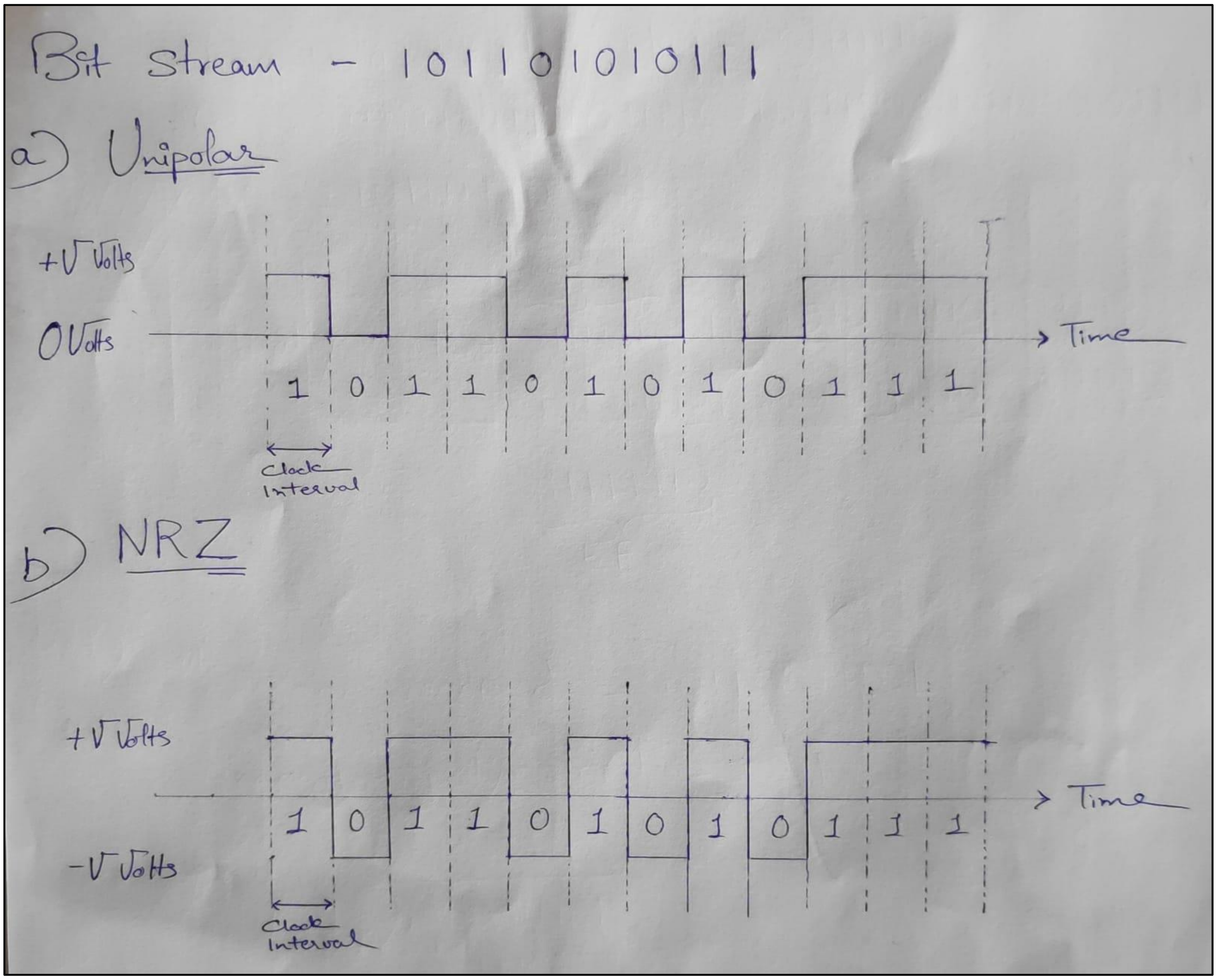


Figure 1 – Unipolar and NRZ encoding of bit stream 1 0 1 1 0 1 0 1 0 1 1 1

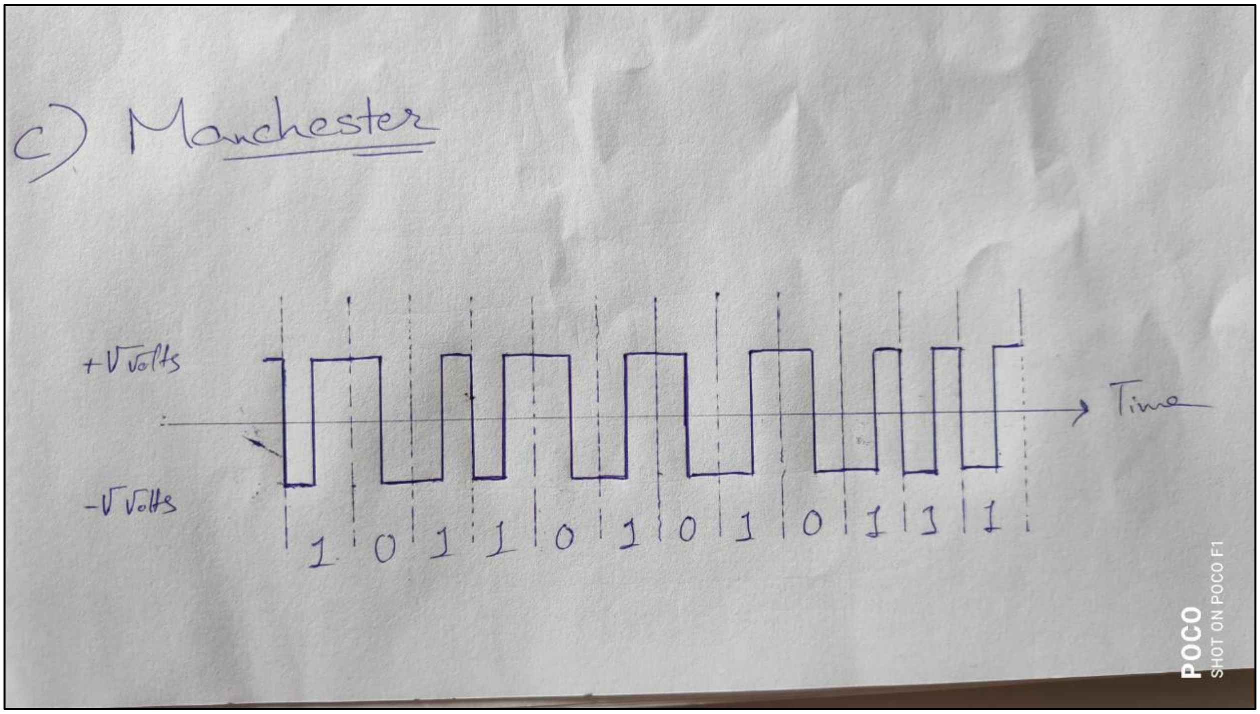


Figure 2 – Manchester encoding of bit stream 1 0 1 1 0 1 0 1 0 1 1 1

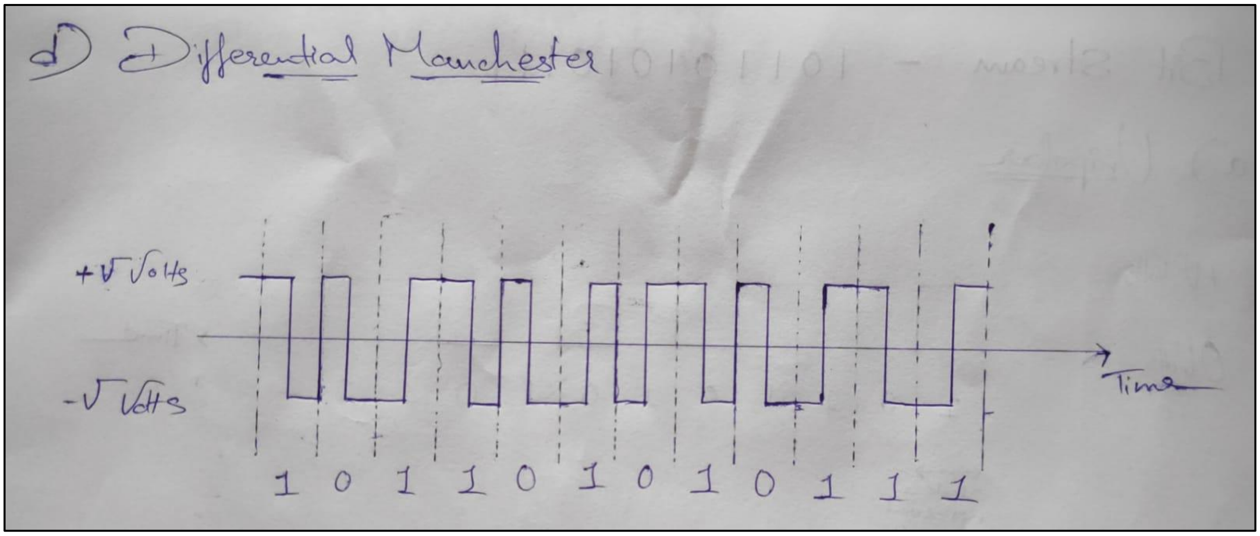


Figure 3 – Differential Manchester encoding of bit stream 1 0 1 1 0 1 0 1 0 1 1 1

Question 2 <Bit Stuffing Question>

The following message is to be sent by a host running a protocol with starting and ending flags and bit stuffing. The starting and ending flags are both 01111110 and they have not yet been added.

- ⇨ 0111111011111011110011111100111111000000111110101111110
- ⇨ What is the message actually sent (after bit stuffing and after adding the starting and ending flags)?

Answer

Starting flag = Ending flag = 01111110

Message = 0111111011111011110011111100111111000000111110101111110

Bit stuffing is done when a bit sequence having 5 or more consecutive 1s is to be transmitted. Extra 0 bit is stuffed after the fifth 1.

So, for message 0111111011111011110011111100111111000000111110101111110, a 0 bit is stuffed after the underlined group of bits.

Thus, message after bit stuffing is as below:

011111**0**1011111**0**011110011111**0**10011111**0**100000011111**0**01011111**0**10

Now, adding starting and ending flags, final message sent by the host is as below

01111110011111**0**1011111**0**011110011111**0**10011111**0**100000011111**0**01011111**0**10**01111110**

Final answer –

01111110011111**0**1011111**0**011110011111**0**10011111**0**100000011111**0**01011111**0**10**01111110**

Question 3 <CRC Question>

If the generator polynomial is x5 + x3 + 1 and the message to be sent is 1110010111, what is the actual bit string transmitted? Show steps

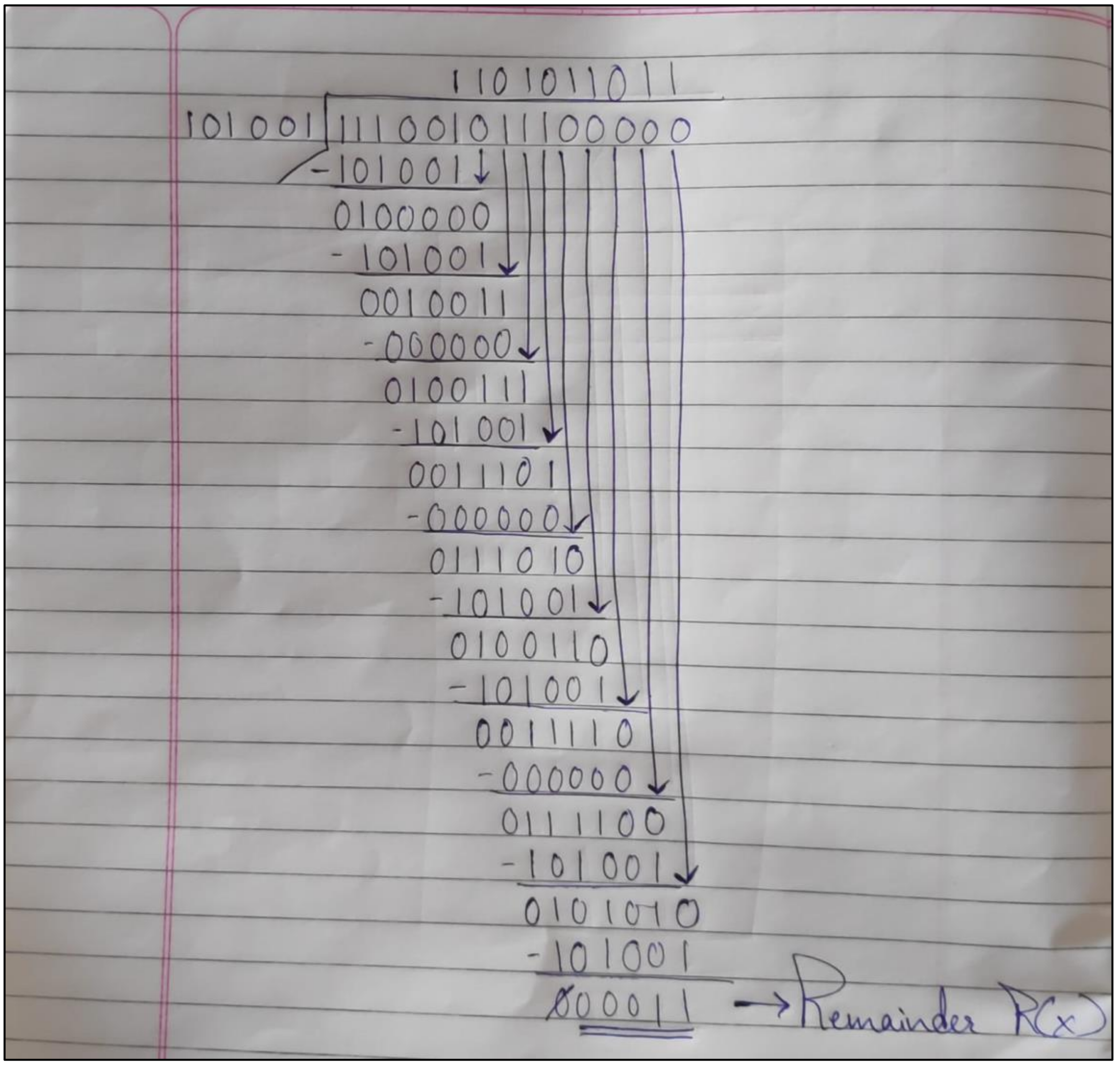
Answer

Let G(x) = x5 + x3 + 1 = 101001
Let M(x) = 1110010111

First, M(x) is converted to M’(x) by appending 5 zeros since G(x) is degree 5 polynomial.

Thus, M’(x) = 111001011100000

Step 1 – Divide M’(x) by G(x)



```
                1101011011
101001 | 111001011100000
        -101001
         0100000
         -101001
          0010011
          -000000
           0100111
           -101001
            0011101
            -000000
             0111010
             -101001
              0100110
              -101001
               0011110
               -000000
                0111100
                -101001
                 0101010
                 -101001
                  ~~0~~00011 → Remainder R(x)
```

Figure 4 – Division of M'(x) by G(x)

Step 2 – Convert M(x) to P(x) such P(x) is exactly divisible by G(x). P(x) is M’(x) – R(x) which is divisible by G(x).

Thus,

```
M’(x) = 111001011100000
R(x)   =                    00011
---------------------------------
P(x)   = 111001011100011
```

Thus, the actual bit stream transmitted is P(x) = 111001011100011

Final answer –

Actual bit stream transmitted is **P(x) = 111001011100011**

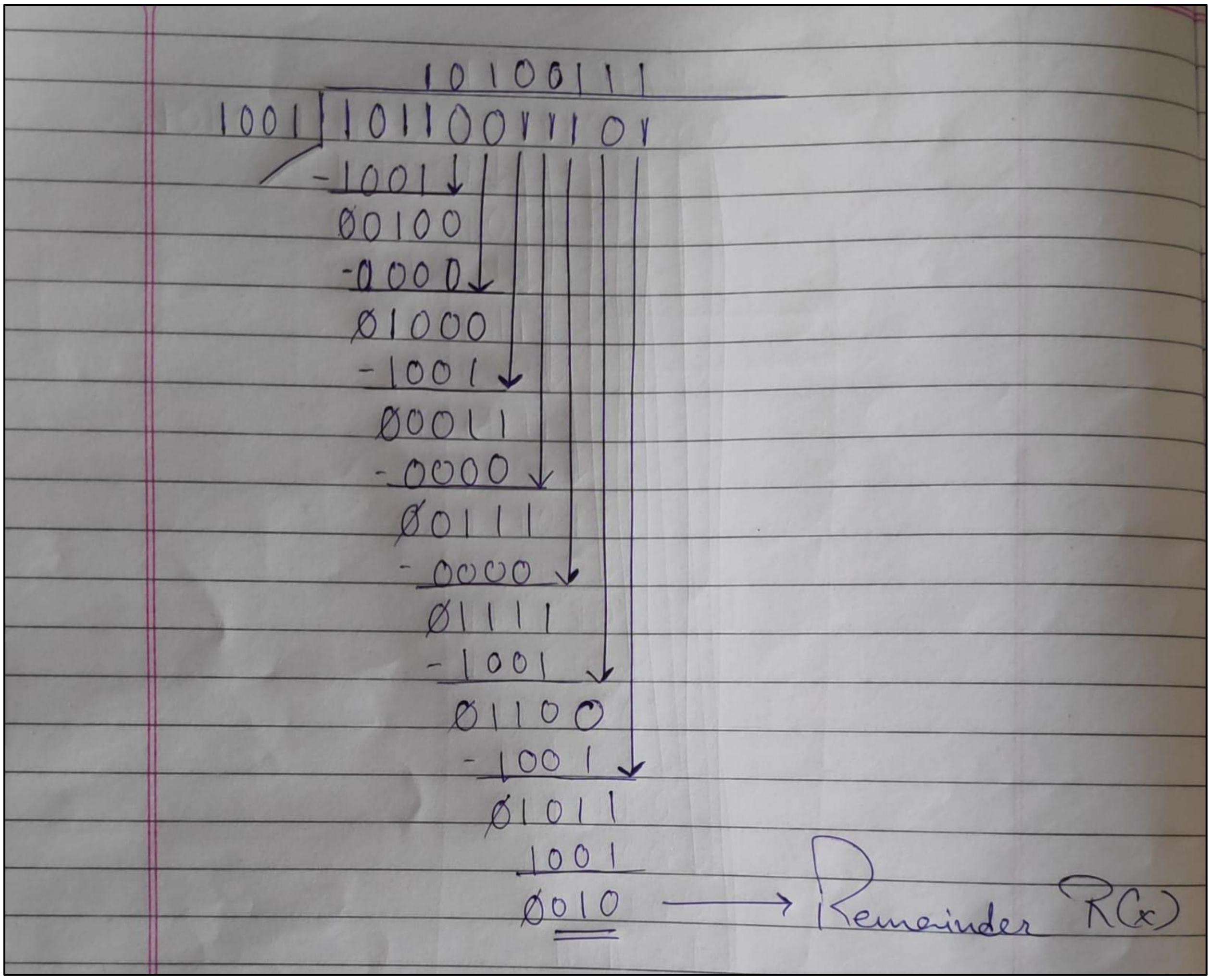
Question 4 <CRC Question>

The data link layer of a host receives the bit string 10110011101. The data link layer uses CRC for error checking and the generator polynomial is 1001. Is there an error detected in the frame? Show steps.

<u>Answer</u>

Let M(x) = 10110011101
Let G(x) = 1001

Step 1 – Divide M’(x) by G(x)



```
              10100111
      1001 ) 10110011101
            -1001↓
             Ø0100
             -0000↓
              Ø1000
              -1001↓
               Ø0011
               -0000↓
                Ø0111
                -0000↓
                 Ø1111
                 -1001↓
                  Ø1100
                  -1001↓
                   Ø1011
                    1001
                   Ø010 ⟶ Remainder R(x)
```

Since, the remainder R(x) is not 0, there is an error detected in the frame.

Final answer –

Yes, error is detected in the frame as the remainder R(x) is non-zero.

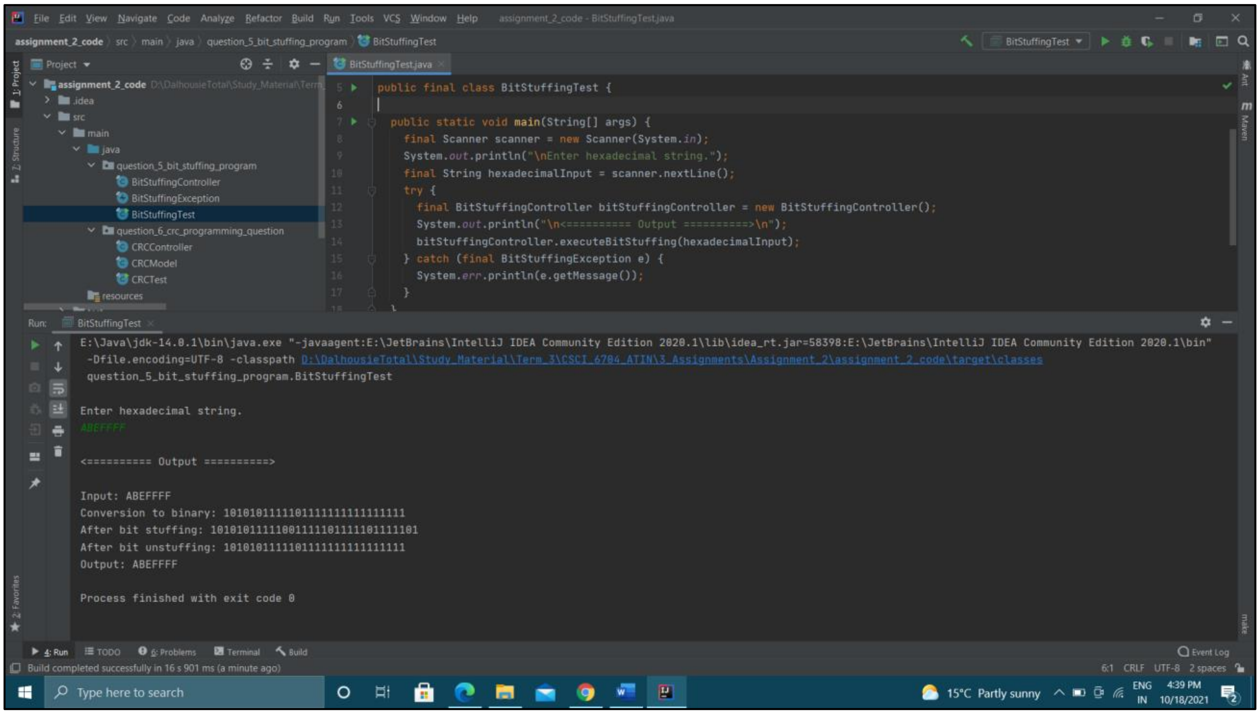
Question 5 <Bit Stuffing Program>

In this exercise, you will be writing a simple program that does the following:
a) Read a String of hex digits
b) Convert the String into a String of binary numbers
c) Perform bit stuffing on the binary String
d) Unstuff the bits from the binary String
e) Produce the original hex String

Answer

Sample Run 1

Figure 5 displays the output of Bit Stuffing Program for input hexadecimal string ABEFFFF.



```java
public final class BitStuffingTest {

  public static void main(String[] args) {
    final Scanner scanner = new Scanner(System.in);
    System.out.println("\nEnter hexadecimal string.");
    final String hexadecimalInput = scanner.nextLine();
    try {
      final BitStuffingController bitStuffingController = new BitStuffingController();
      System.out.println("\n<========== Output ==========>\n");
      bitStuffingController.executeBitStuffing(hexadecimalInput);
    } catch (final BitStuffingException e) {
      System.err.println(e.getMessage());
    }
```

```
E:\Java\jdk-14.0.1\bin\java.exe "-javaagent:E:\JetBrains\IntelliJ IDEA Community Edition 2020.1\lib\idea_rt.jar=58398:E:\JetBrains\IntelliJ IDEA Community Edition 2020.1\bin"
 -Dfile.encoding=UTF-8 -classpath D:\DalhousieTotal\Study_Material\Term_3\CSCI_6704_ATIN\3_Assignments\Assignment_2\assignment_2_code\target\classes
 question_5_bit_stuffing_program.BitStuffingTest

Enter hexadecimal string.
ABEFFFF

<========== Output ==========>

Input: ABEFFFF
Conversion to binary: 1010101111101111111111111111
After bit stuffing: 10101011111001111101111101111101
After bit unstuffing: 1010101111101111111111111111
Output: ABEFFFF

Process finished with exit code 0
```

File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
assignment_2_code - BitStuffingTest.java
assignment_2_code › src › main › java › question_5_bit_stuffing_program › BitStuffingTest
Project
assignment_2_code D:\DalhousieTotal\Study_Material\Term
.idea
src
main
java
question_5_bit_stuffing_program
BitStuffingController
BitStuffingException
BitStuffingTest
question_6_crc_programming_question
CRCController
CRCModel
CRCTest
resources
BitStuffingTest.java
Run: BitStuffingTest
4: Run
TODO
6: Problems
Terminal
Build
Event Log
Build completed successfully in 16 s 901 ms (a minute ago)
6:1 CRLF UTF-8 2 spaces
Type here to search
15°C Partly sunny
ENG IN
4:39 PM 10/18/2021

Figure 5 – Output of Bit Stuffing Program for input hexadecimal string ABEFFFF

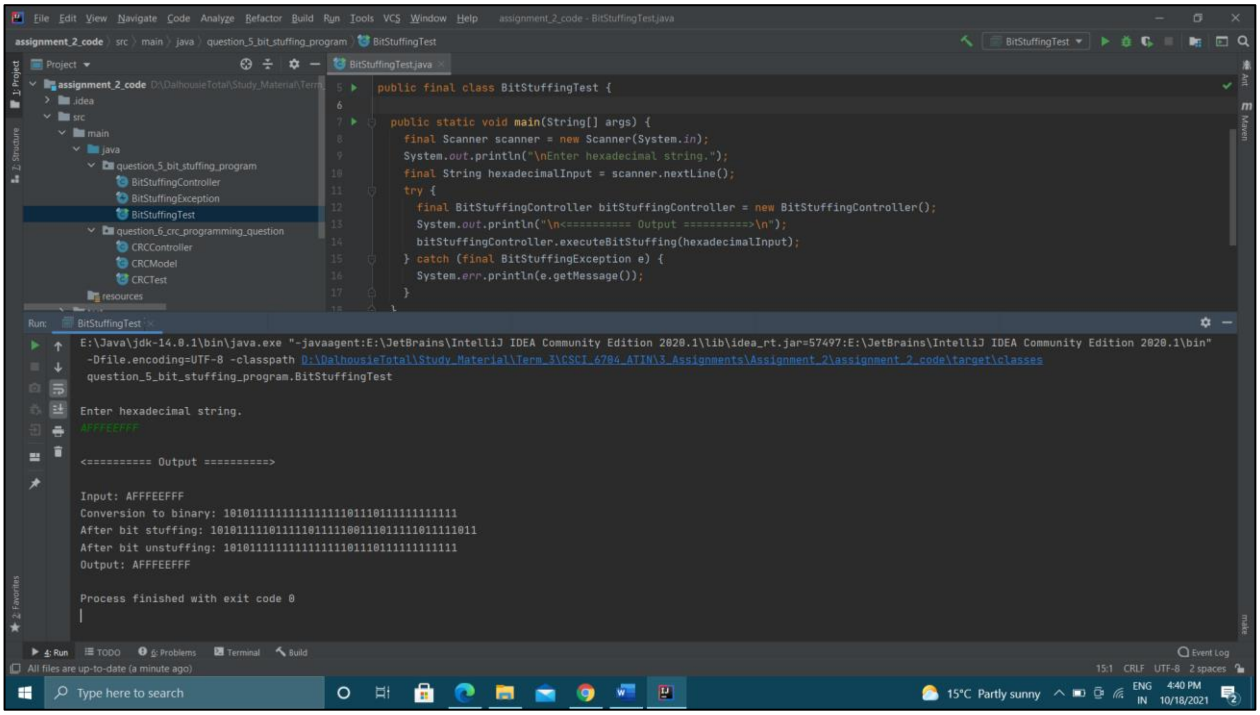
Enter hexadecimal string.
ABEFFFF

<========== Output ==========>

Input: ABEFFFF
Conversion to binary: 1010101111101111111111111111
After bit stuffing: 10101011111001111101111101111101
After bit unstuffing: 1010101111101111111111111111
Output: ABEFFFF

Sample Run 2

Figure 6 displays the output of Bit Stuffing Program for input hexadecimal string AFFFEEFFF.



```
public final class BitStuffingTest {

  public static void main(String[] args) {
    final Scanner scanner = new Scanner(System.in);
    System.out.println("\nEnter hexadecimal string.");
    final String hexadecimalInput = scanner.nextLine();
    try {
      final BitStuffingController bitStuffingController = new BitStuffingController();
      System.out.println("\n<========== Output ==========>\n");
      bitStuffingController.executeBitStuffing(hexadecimalInput);
    } catch (final BitStuffingException e) {
      System.err.println(e.getMessage());
    }
```

```
E:\Java\jdk-14.0.1\bin\java.exe "-javaagent:E:\JetBrains\IntelliJ IDEA Community Edition 2020.1\lib\idea_rt.jar=57497:E:\JetBrains\IntelliJ IDEA Community Edition 2020.1\bin"
 -Dfile.encoding=UTF-8 -classpath D:\DalhousieTotal\Study_Material\Term_3\CSCI_6704_ATIN\3_Assignments\Assignment_2\assignment_2_code\target\classes
 question_5_bit_stuffing_program.BitStuffingTest

Enter hexadecimal string.
AFFFEEFFF

<========== Output ==========>

Input: AFFFEEFFF
Conversion to binary: 101011111111111111101110111111111111
After bit stuffing: 10101111101111101111100111011111011111011
After bit unstuffing: 101011111111111111101110111111111111
Output: AFFFEEFFF

Process finished with exit code 0
```

File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
assignment_2_code - BitStuffingTest.java
assignment_2_code › src › main › java › question_5_bit_stuffing_program › BitStuffingTest
Project
assignment_2_code
.idea
src
main
java
question_5_bit_stuffing_program
BitStuffingController
BitStuffingException
BitStuffingTest
question_6_crc_programming_question
CRCController
CRCModel
CRCTest
resources
BitStuffingTest.java
Run: BitStuffingTest
4: Run
TODO
6: Problems
Terminal
Build
Event Log
All files are up-to-date (a minute ago)
15:1 CRLF UTF-8 2 spaces
Type here to search
15°C Partly sunny
ENG IN
4:40 PM
10/18/2021

Figure 6 – Output of Bit Stuffing Program for input hexadecimal string AFFFEEFFF

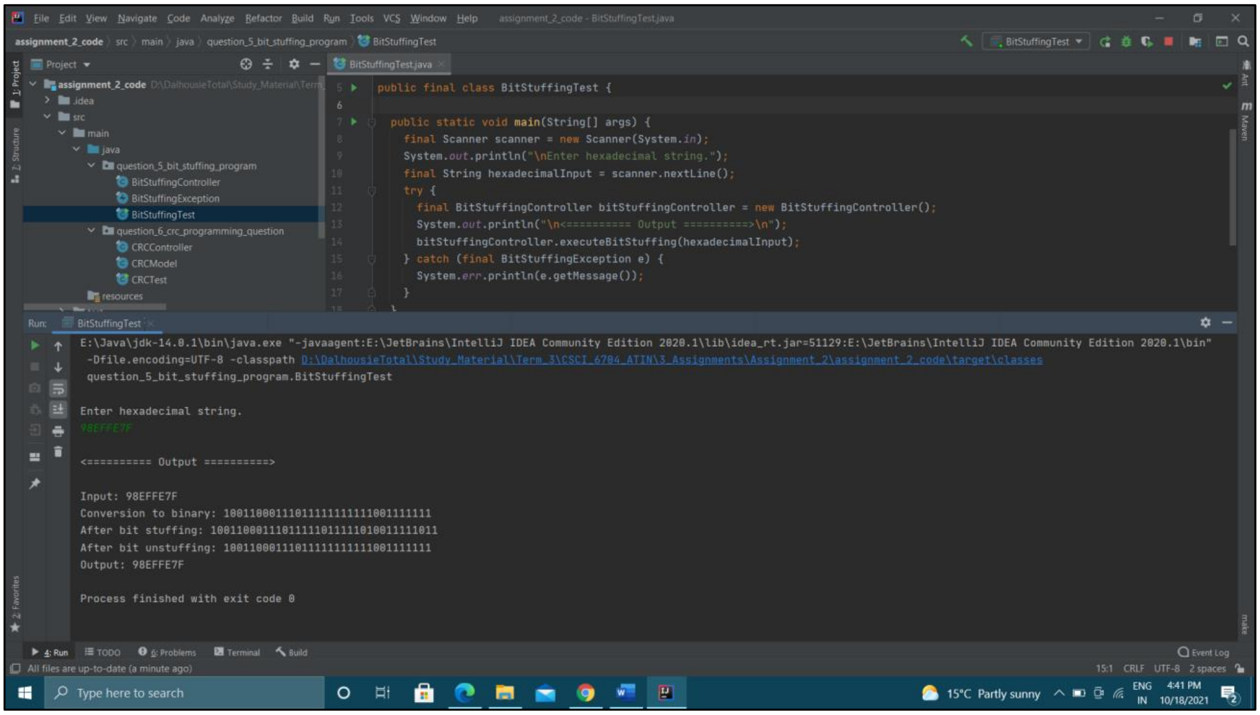
Enter hexadecimal string.
AFFFEEFFF

<========== Output ==========>

Input: AFFFEEFFF
Conversion to binary: 101011111111111111101110111111111111
After bit stuffing: 10101111101111101111100111011111011111011
After bit unstuffing: 101011111111111111101110111111111111
Output: AFFFEEFFF

Sample Run 3

Figure 7 displays the output of Bit Stuffing Program for input hexadecimal string 98EFFE7F.



```java
public final class BitStuffingTest {

  public static void main(String[] args) {
    final Scanner scanner = new Scanner(System.in);
    System.out.println("\nEnter hexadecimal string.");
    final String hexadecimalInput = scanner.nextLine();
    try {
      final BitStuffingController bitStuffingController = new BitStuffingController();
      System.out.println("\n<========== Output ==========>\n");
      bitStuffingController.executeBitStuffing(hexadecimalInput);
    } catch (final BitStuffingException e) {
      System.err.println(e.getMessage());
    }
```

```
E:\Java\jdk-14.0.1\bin\java.exe "-javaagent:E:\JetBrains\IntelliJ IDEA Community Edition 2020.1\lib\idea_rt.jar=51129:E:\JetBrains\IntelliJ IDEA Community Edition 2020.1\bin"
 -Dfile.encoding=UTF-8 -classpath D:\DalhousieTotal\Study_Material\Term_3\CSCI_6704_ATIN\3_Assignments\Assignment_2\assignment_2_code\target\classes
 question_5_bit_stuffing_program.BitStuffingTest

Enter hexadecimal string.
98EFFE7F

<========== Output ==========>

Input: 98EFFE7F
Conversion to binary: 10011000111011111111111001111111
After bit stuffing: 1001100011101111101111101001111011
After bit unstuffing: 10011000111011111111111001111111
Output: 98EFFE7F

Process finished with exit code 0
```

Figure 7 - Output of Bit Stuffing Program for input hexadecimal string 98EFFE7F

Enter hexadecimal string.
98EFFE7F

<========== Output ==========>

Input: 98EFFE7F
Conversion to binary: 10011000111011111111111001111111
After bit stuffing: 1001100011101111101111101001111011
After bit unstuffing: 10011000111011111111111001111111
Output: 98EFFE7F

Source Code

The source code for question 5 is in package “question_5_bit_stuffing_program”. It consists of three JAVA files which are listed below:

1. **BitStuffingController** – This JAVA file contains the core program logic. Four important methods are as below:
    a. **Method 1** – convertHexadecimalInputToBinary(final String hexadecimalInput)
    b. **Method 2** – performBitStuffing(final String binaryInput)
    c. **Method 3** – performBitUnStuffing(final String binaryInputStuffed)
    d. **Method 4** – convertBinaryToHexadecimal(final String binaryInputUnStuffed)
2. **BitStuffingException** – This JAVA file contains the exception logic.
3. **BitStuffingTest** – This JAVA file tests the **BitStuffingController**.

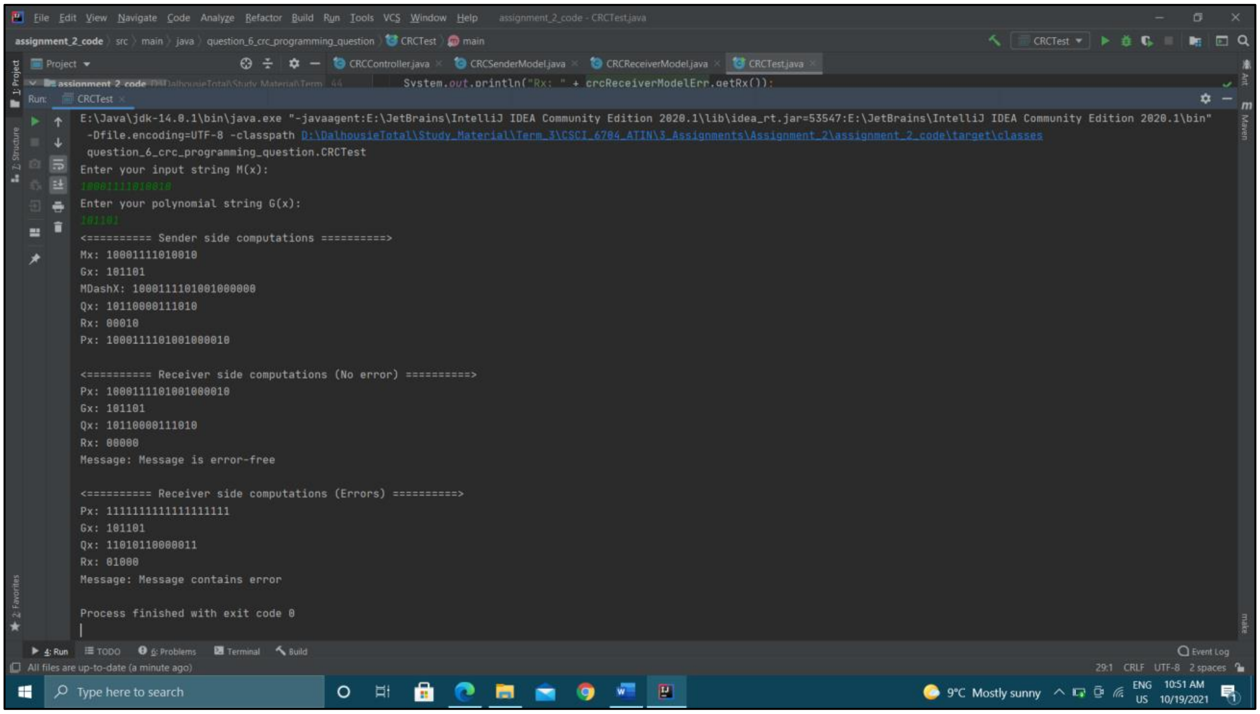
Question 6 <CRC Programming Question>

Implement the sending and receiving CRC protocols by writing program routines (functions/methods) for each of the following:
a. Given a bit string, compute the CRC remainder and generate the bit string that is transmitted.
b. Given a bit string with the CRC remainder appended, divide by G(x) and determine if the message is error-free.
c. Use the above methods in a test program that accepts from user input the values of G(x) and the input string, introduce random errors in the transmitted bit string and demonstrate how the receiver can detect the error.

Answer

Sample Run 1

Figure 8 displays the output of CRC Programming Question for input string M(x) = 10001111010010 and G(x) = 101101.

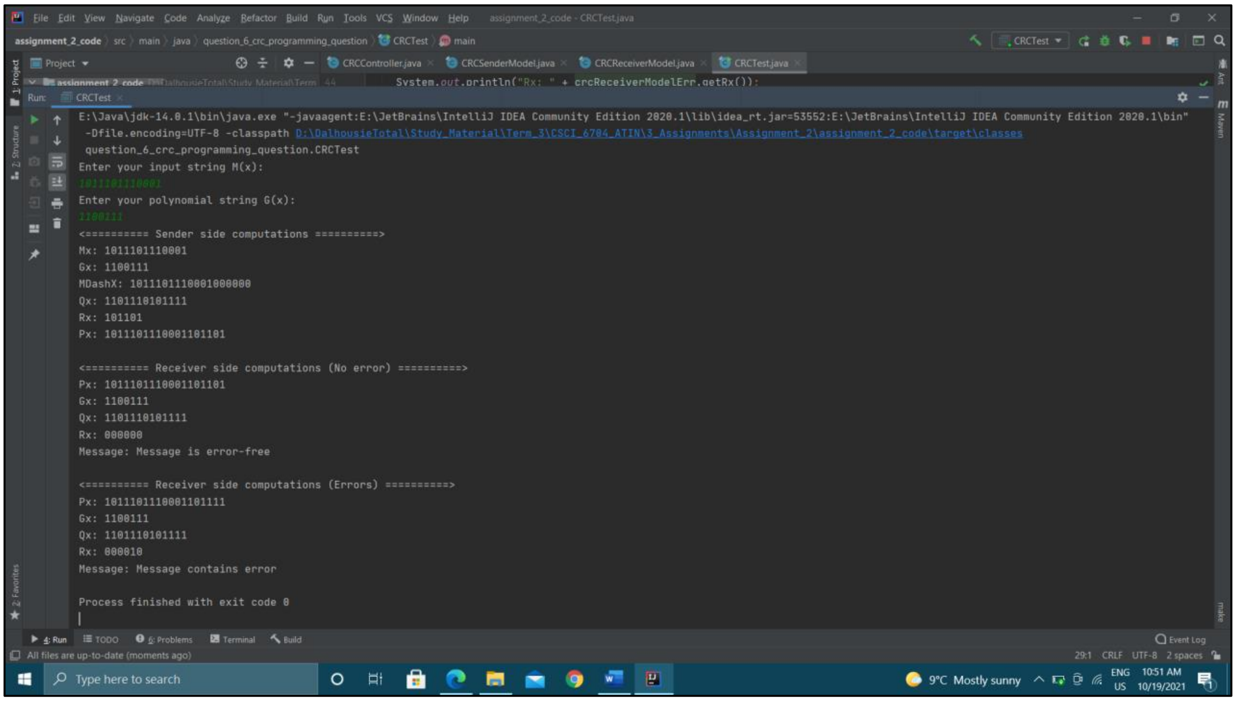


```
E:\Java\jdk-14.0.1\bin\java.exe "-javaagent:E:\JetBrains\IntelliJ IDEA Community Edition 2020.1\lib\idea_rt.jar=53547:E:\JetBrains\IntelliJ IDEA Community Edition 2020.1\bin"
 -Dfile.encoding=UTF-8 -classpath D:\DalhousieTotal\Study_Material\Term_3\CSCI_6704_ATIN\3_Assignments\Assignment_2\assignment_2_code\target\classes
 question_6_crc_programming_question.CRCTest
Enter your input string M(x):
10001111010010
Enter your polynomial string G(x):
101101
<========== Sender side computations ==========>
Mx: 10001111010010
Gx: 101101
MDashX: 1000111101001000000
Qx: 10110000111010
Rx: 00010
Px: 1000111101001000010

<========== Receiver side computations (No error) ==========>
Px: 1000111101001000010
Gx: 101101
Qx: 10110000111010
Rx: 00000
Message: Message is error-free

<========== Receiver side computations (Errors) ==========>
Px: 1111111111111111111
Gx: 101101
Qx: 11010110000011
Rx: 01000
Message: Message contains error

Process finished with exit code 0
```

Figure 8 – Output of CRC Programming Question for input string M(x) = 10001111010010 and G(x) = 101101

Sample Run 2

Figure 9 displays the output of CRC Programming Question for input string M(x) = 1011101110001 and G(x) = 1100111.



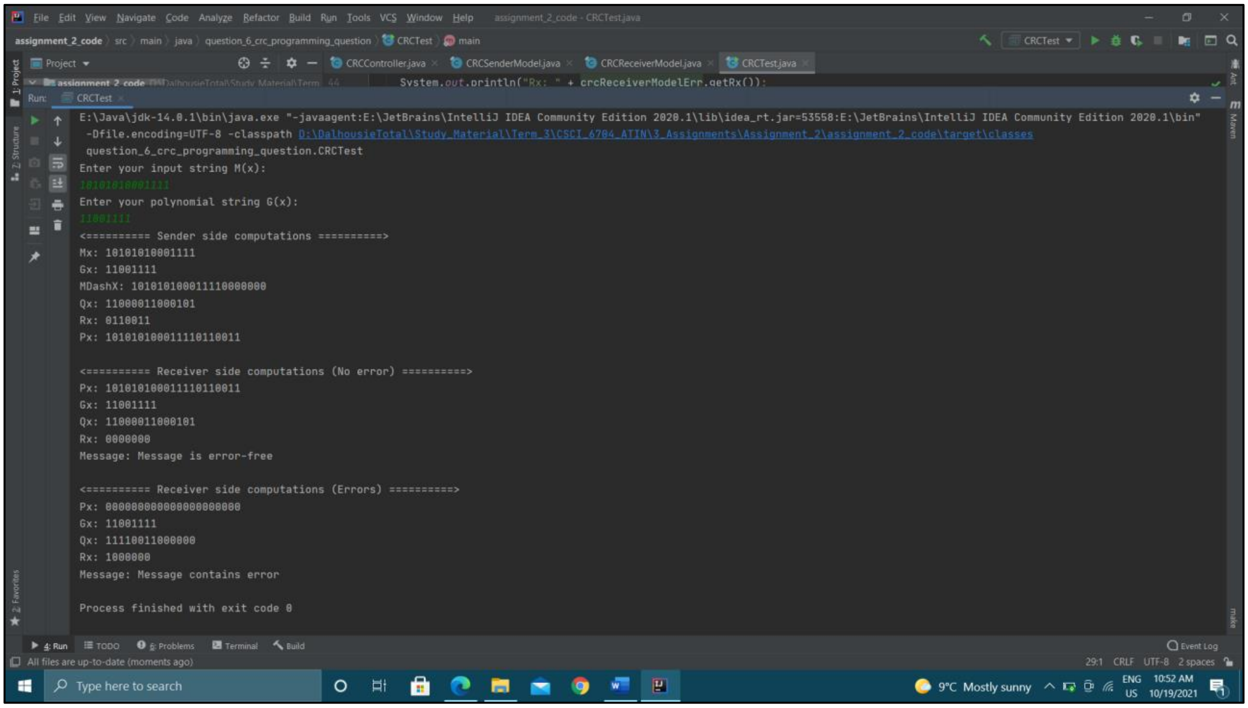
```
E:\Java\jdk-14.0.1\bin\java.exe "-javaagent:E:\JetBrains\IntelliJ IDEA Community Edition 2020.1\lib\idea_rt.jar=53552:E:\JetBrains\IntelliJ IDEA Community Edition 2020.1\bin"
 -Dfile.encoding=UTF-8 -classpath D:\DalhousieTotal\Study_Material\Term_3\CSCI_6704_ATIN\3_Assignments\Assignment_2\assignment_2_code\target\classes
 question_6_crc_programming_question.CRCTest
Enter your input string M(x):
1011101110001
Enter your polynomial string G(x):
1100111
<========== Sender side computations ==========>
Mx: 1011101110001
Gx: 1100111
MDashX: 1011101110001000000
Qx: 1101110101111
Rx: 101101
Px: 1011101110001101101

<========== Receiver side computations (No error) ==========>
Px: 1011101110001101101
Gx: 1100111
Qx: 1101110101111
Rx: 000000
Message: Message is error-free

<========== Receiver side computations (Errors) ==========>
Px: 1011101110001101111
Gx: 1100111
Qx: 1101110101111
Rx: 000010
Message: Message contains error

Process finished with exit code 0
```

File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
assignment_2_code - CRCTest.java
assignment_2_code › src › main › java › question_6_crc_programming_question › CRCTest › main
CRCTest
Project
CRCController.java
CRCSenderModel.java
CRCReceiverModel.java
CRCTest.java
System.out.println("Rx: " + crcReceiverModelErr.getRx());
Run: CRCTest
4: Run
TODO
6: Problems
Terminal
Build
Event Log
All files are up-to-date (moments ago)
29:1 CRLF UTF-8 2 spaces
Type here to search
9°C Mostly sunny
ENG US
10:51 AM 10/19/2021

Figure 9 – Output of CRC Programming Question for input string M(x) = 1011101110001 and G(x) = 1100111

Sample Run 3

Figure 10 displays the output of CRC Programming Question for input string M(x) = 10101010001111 and G(x) = 11001111.



```
E:\Java\jdk-14.0.1\bin\java.exe "-javaagent:E:\JetBrains\IntelliJ IDEA Community Edition 2020.1\lib\idea_rt.jar=53558:E:\JetBrains\IntelliJ IDEA Community Edition 2020.1\bin"
 -Dfile.encoding=UTF-8 -classpath D:\DalhousieTotal\Study_Material\Term_3\CSCI_6704_ATIN\3_Assignments\Assignment_2\assignment_2_code\target\classes
 question_6_crc_programming_question.CRCTest
Enter your input string M(x):
10101010001111
Enter your polynomial string G(x):
11001111
<========== Sender side computations ==========>
Mx: 10101010001111
Gx: 11001111
MDashX: 101010100011110000000
Qx: 11000011000101
Rx: 0110011
Px: 101010100011110110011

<========== Receiver side computations (No error) ==========>
Px: 101010100011110110011
Gx: 11001111
Qx: 11000011000101
Rx: 0000000
Message: Message is error-free

<========== Receiver side computations (Errors) ==========>
Px: 000000000000000000000
Gx: 11001111
Qx: 11110011000000
Rx: 1000000
Message: Message contains error

Process finished with exit code 0
```

Figure 10 - Output of CRC Programming Question for input string M(x) = 10101010001111 and G(x) = 11001111

Source Code

The source code for question 6 is in package "question_6_crc_programming_question". It consists of four JAVA files which are listed below:

1. **CRCController** – This JAVA file contains the core program logic. Three important methods are as below:
    a. Method 1 – performSenderSideComputation(final String Mx, final String Gx)
    b. Method 2 – performReceiverSideComputation(final String Px, final String Gx)
    c. Method 3 – introduceRandomError(final String Px)
2. **CRCReceiverModel** – This JAVA file contains receiver model class.
3. **CRCSenderModel** – This JAVA file contains sender model class.
4. **CRCTest** – This JAVA file tests the **CRCController** class.