IT 327 - Lab #10

Error detection and Coding: Parity, CRC, and FEC

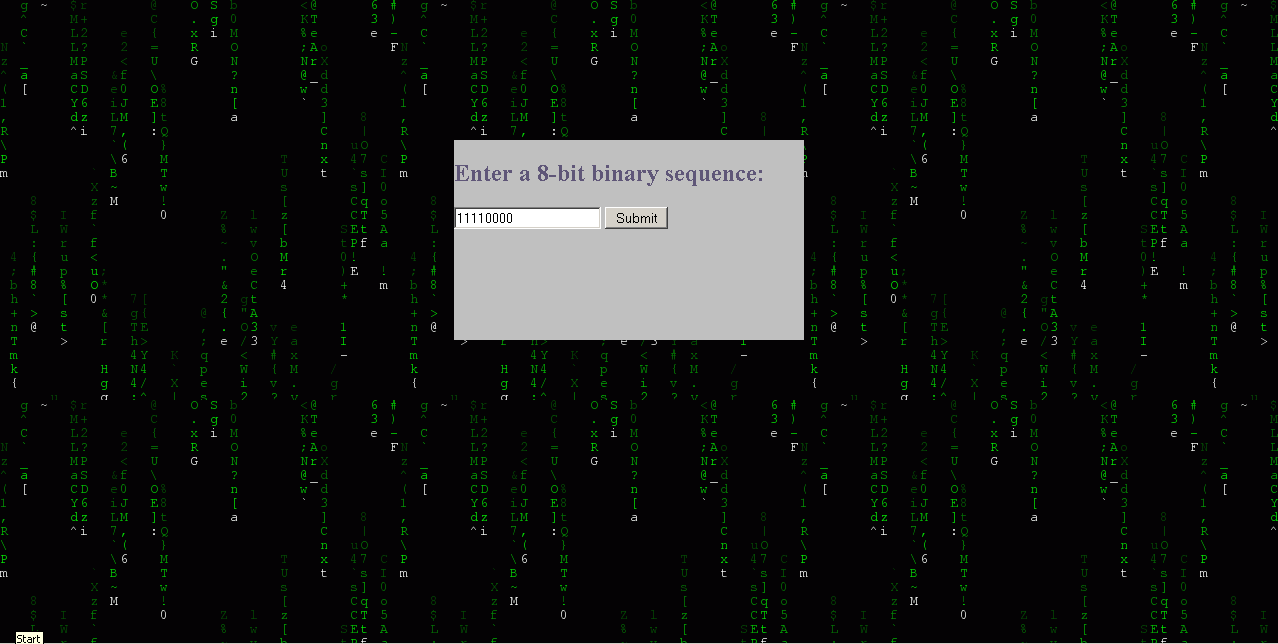
# Objective

The purposes of this lab are to experiment with parity and multi-bit errors in parity, also to experiment with CRC, do some encoding, lastly to write a simple program for forward error correction coding and to experiment with it.

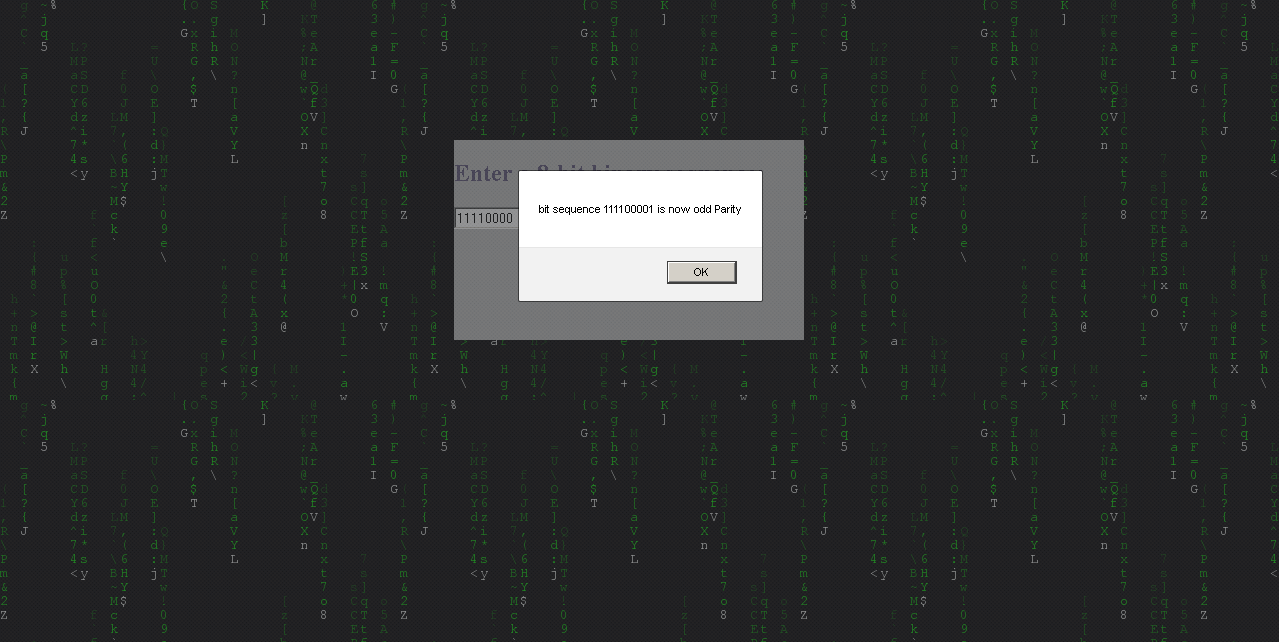
# Procedures

## Step 1

Write a program with any language of your choosing for example: C, C++, JavaScript, or PHP that generates a parity bit (even or odd - you choose) for each byte of data and verify its correct operation. For this lab I have chosen Odd Parity. Below is a screenshot using JavaScript, accepting an input from user.



The output making the bit sequence odd parity is found on the following screenshot found below:



Write another program to detect correct data based on the parity you have defined. Use the program to generate a byte of data along with the parity bit (9-bits total), observe what happens when there are 0, 1, 2, or 3 bits in error in a given byte. Discuss the probability of an undetected error occurring.

For this part of the lab I have chosen bits position 0,1,2 to have bit errors I have discovered that it creates an even Parity which is wrong, since my program is using odd Parity approach. Please see the following screenshots below showing the sequence of user inputting a bit sequence Figure 1, then making the bit sequence odd Parity Figure 2, then checking the bit sequence if it is still odd Parity, if it is still odd Parity then bit sequence is good, otherwise, if even, program throws an error message saying that bit sequence is wrong, see Figure 3:

Figure 1: user input.

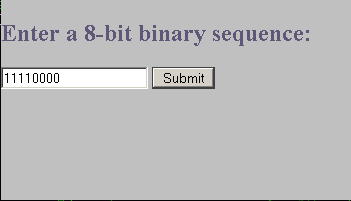


Figure 2: Odd Parity

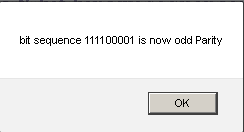
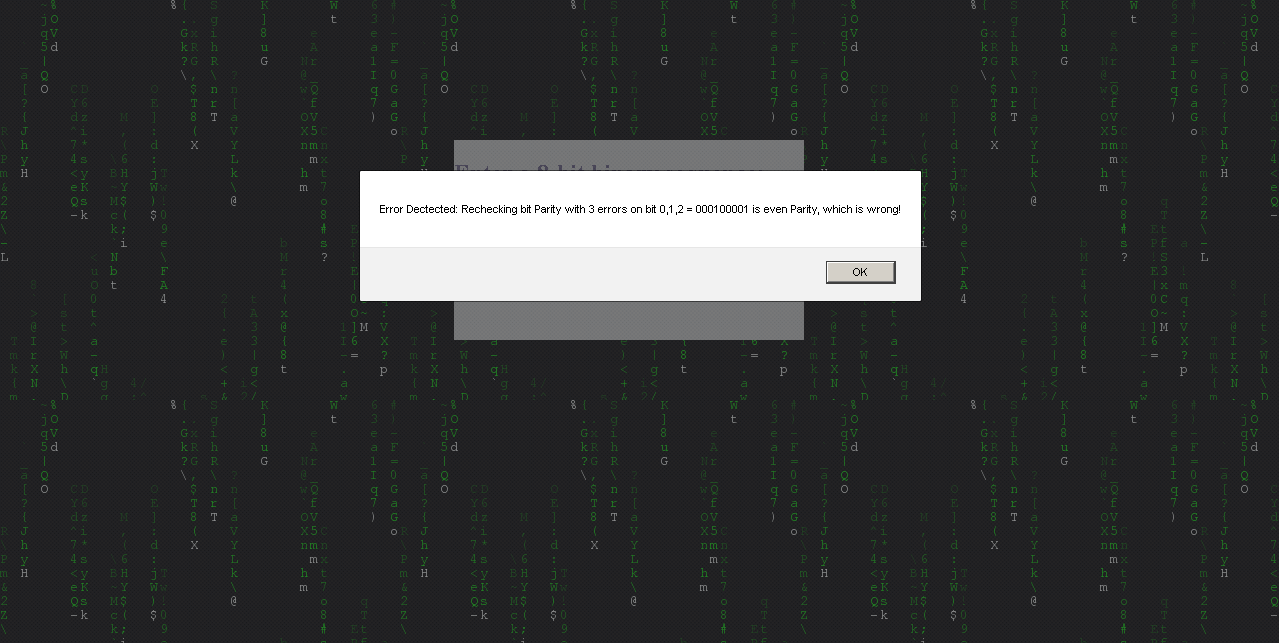


Figure 3: Adding bit errors on position 0,1,2 and checking if bit sequence is still odd.



After many attempts trying to see if any error goes undetected, the program would always catch the errors, and every time it was able to detect the errors it throws an error message to the user to let him/her know about the error. The probability of an undetected error occurring is not likely.

## Step 2

Write a program that generates an 8-bit CRC checksum for a data stream. Follow the pseudo code in this lab. This same program may be used to check correct reception of the data stream. The simplest data stream for which this may be used is a single byte; simulate at least 5 different 1-byte data streams both with and w/out errors, log the results. Particularly interesting are error data streams which differ by only 1 bit from the correct data. For this portion of the lab I chose to put an error on bit position 0 and output the result for the user to check it on the screen.

**CRC Program Pseudo code Reference:**

Initialize counter and data stream

Loop for each bit in data stream

Define next states for each bit

Next Bit 1 = next bit of Data In

Next Bit 2 = Present Bit 1

Next Bit 3 = Present Bit 2 XOR

present Bit 8

Next Bit 4 = Present Bit 3 XOR

present Bit 8

Next Bit 5 = Present Bit 4 XOR

present Bit 8

Next Bit 6 = Present Bit 5

Next Bit 7 = Present Bit 6

Next Bit 8 = Present Bit 7

Move next states into present states

Present Bit 1 = Next Bit 1

Present Bit 2 = Next Bit 2

Present Bit 3 = Next Bit 3

Present Bit 4 = Next Bit 4

Present Bit 5 = Next Bit 5

Present Bit 6 = Next Bit 6

Present Bit 7 = Next Bit 7

Present Bit 8 = Next Bit 8

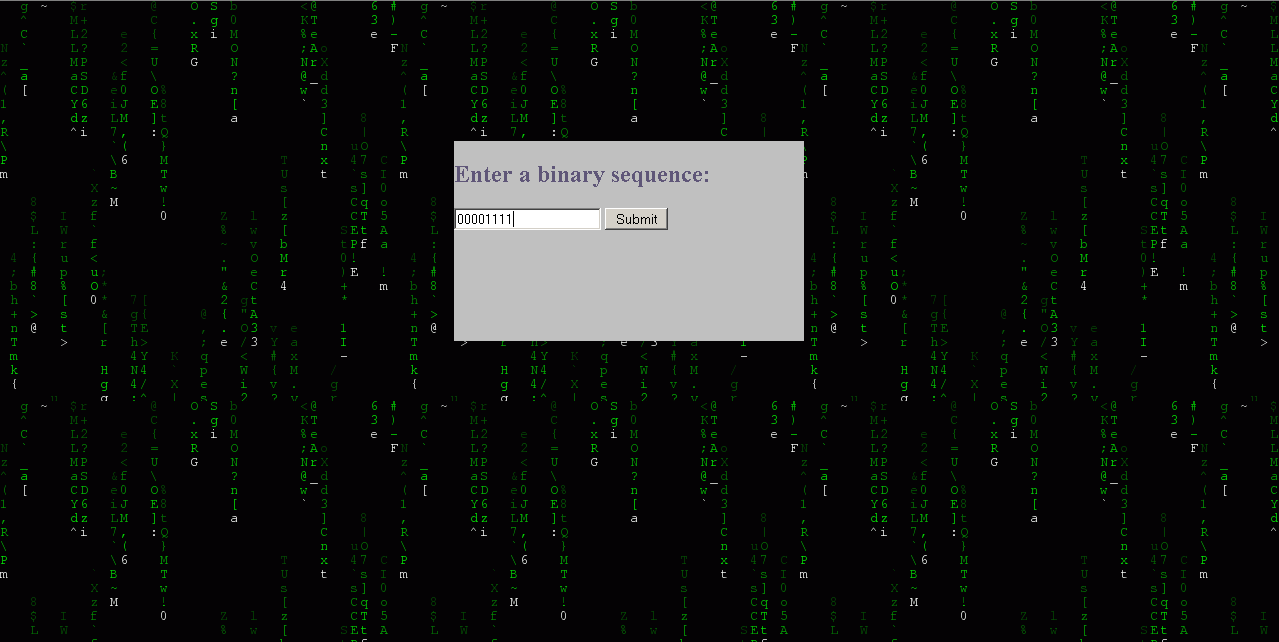
Output CRC encoder byte to screen

End loop

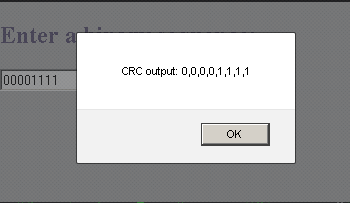
Output final CRC checksum byte

### 1st Byte data stream

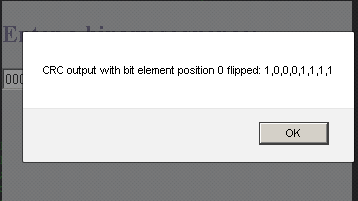
Screenshot showing the program asking for the user’s Input.



Screenshot showing CRC output.

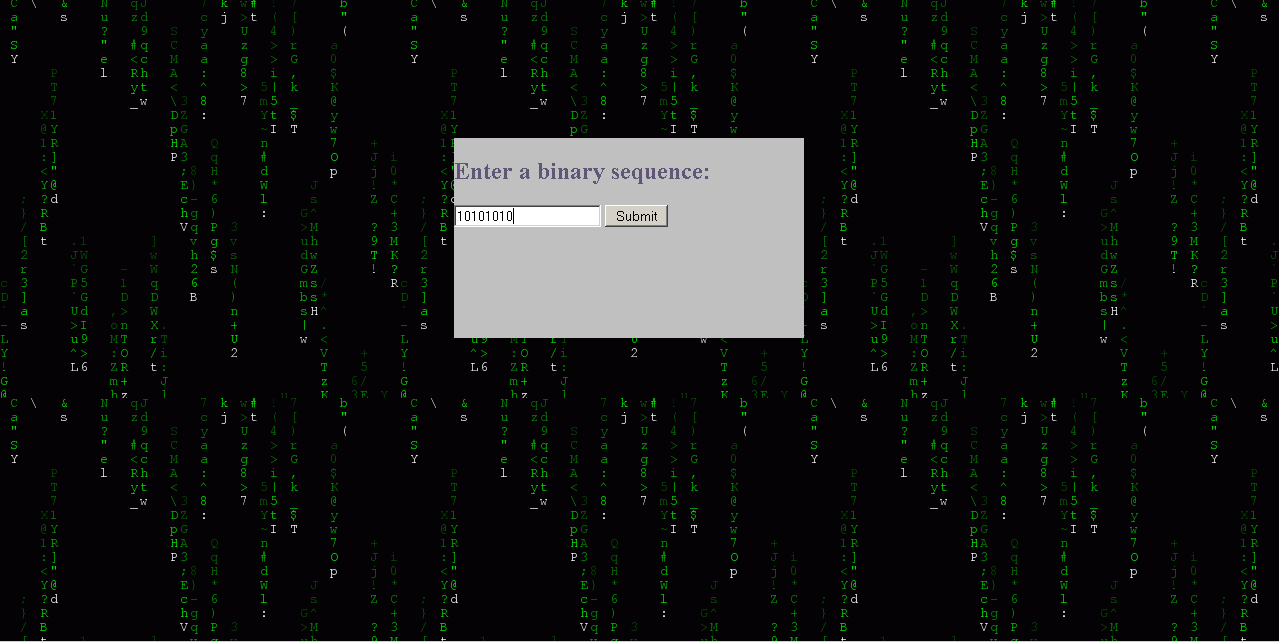


Screenshot showing CRC output with bit position 0 as a bit error.

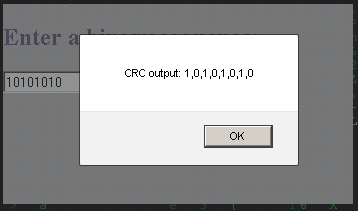


### 2nd Byte data stream

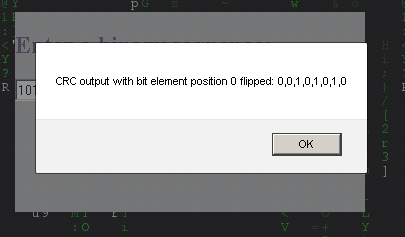
Screenshot showing the program asking for the user’s Input.



Screenshot showing CRC output.

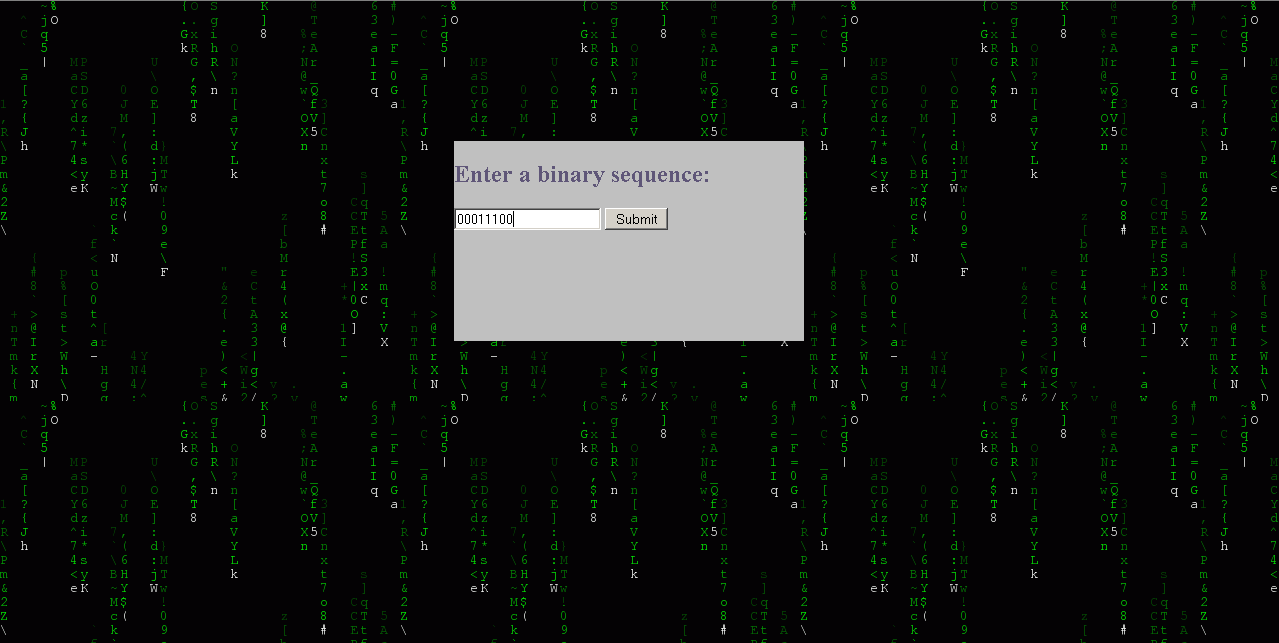


Screenshot showing CRC output with bit position 0 as a bit error.

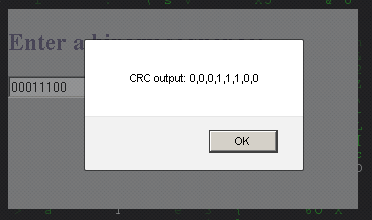


### 3rd Byte data stream

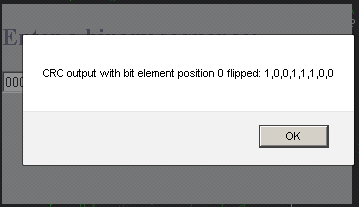
Screenshot showing the program asking for the user’s Input.



Screenshot showing CRC output.

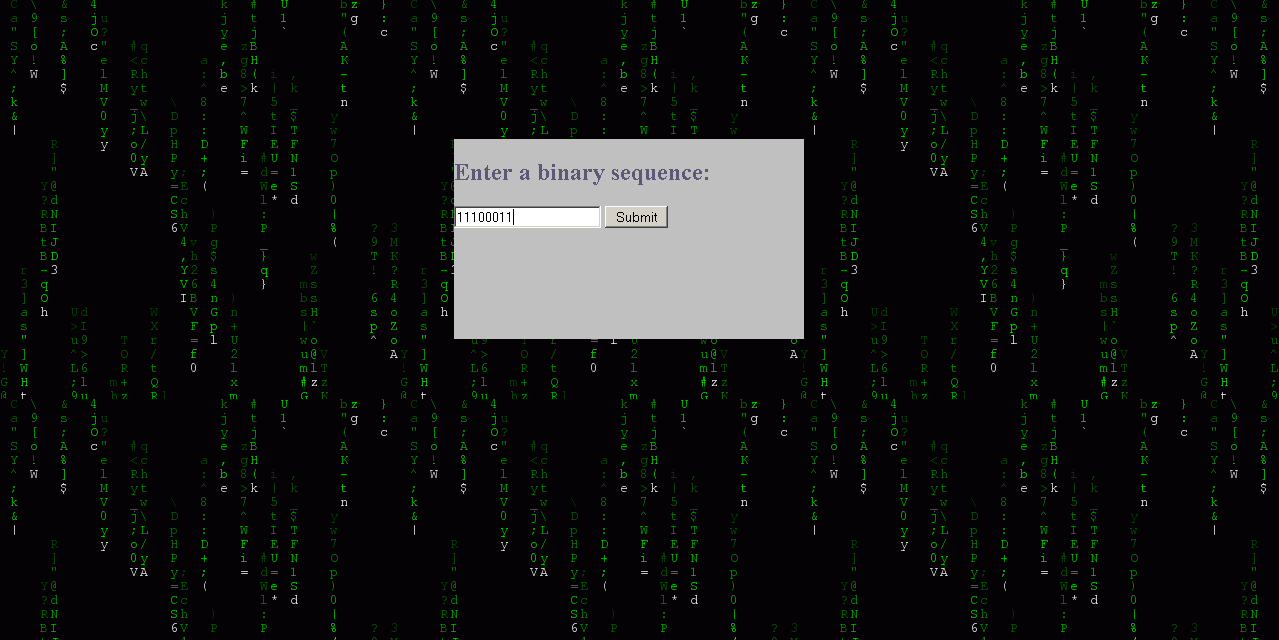


Screenshot showing CRC output with bit position 0 as a bit error.

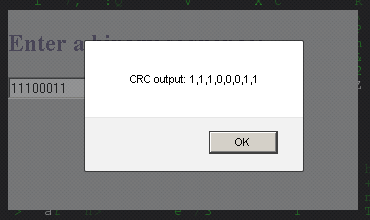


### 4th Byte data stream

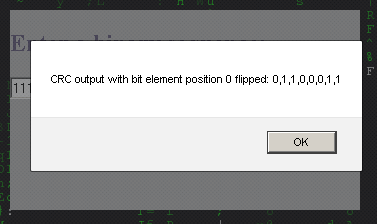
Screenshot showing the program asking for the user’s Input.



Screenshot showing CRC output.

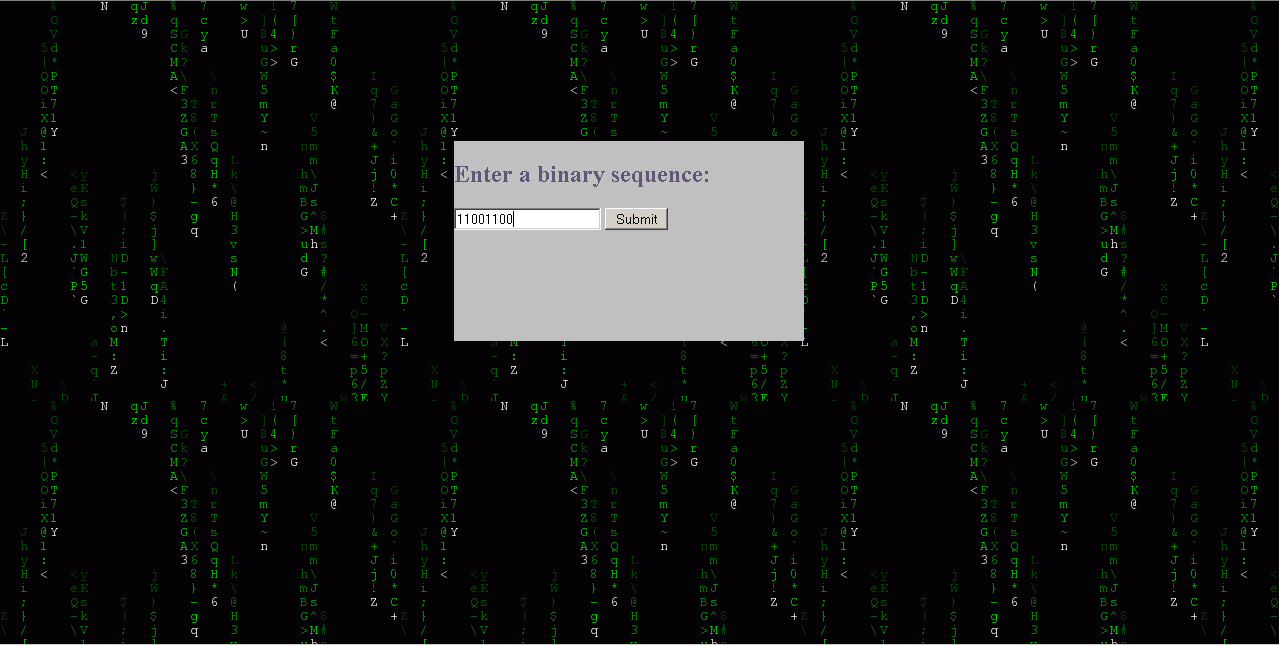


Screenshot showing CRC output with bit position 0 as a bit error.

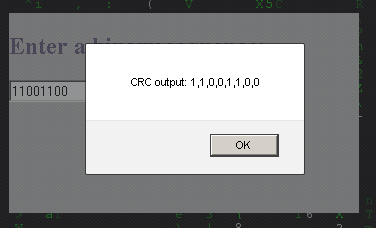


### 5th Byte data stream

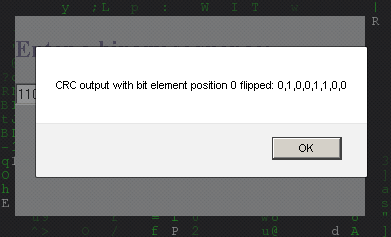
Screenshot showing the program asking for the user’s Input.



Screenshot showing CRC output.



Screenshot showing CRC output with bit position 0 as a bit error.

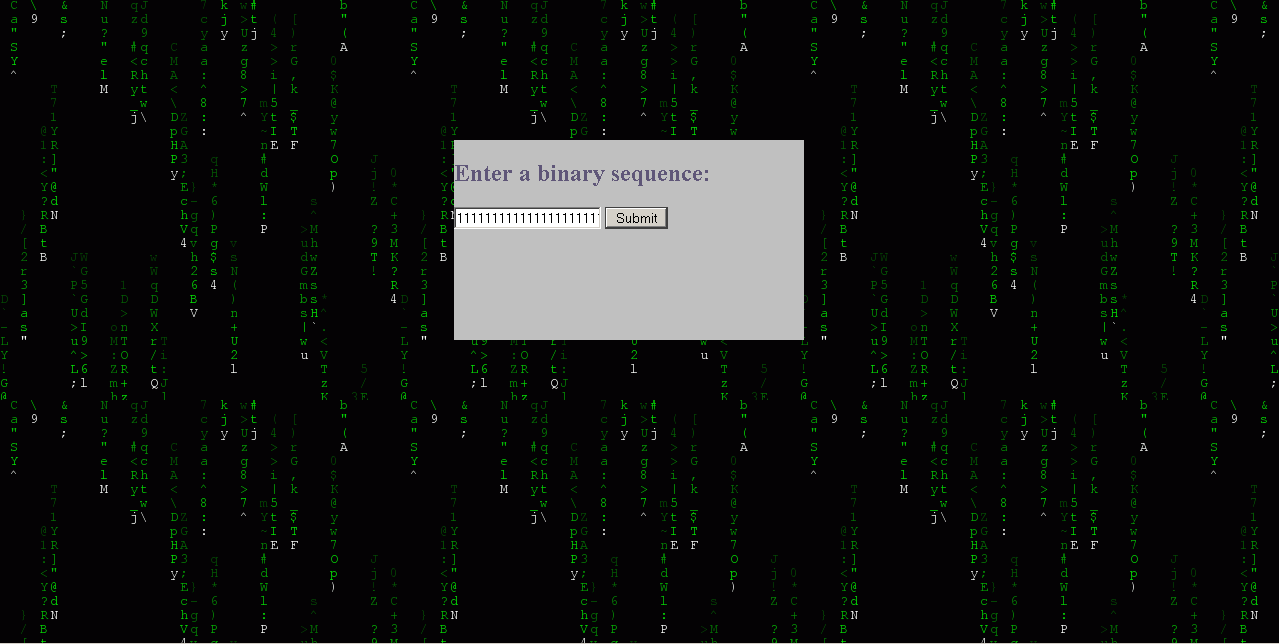


### 32 byte data stream

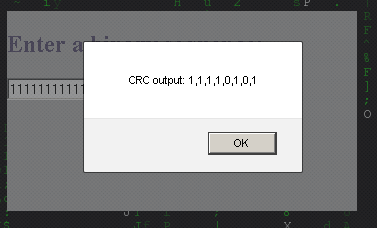
Simulate at least one 32-byte data stream with at least one error and display the results.

This is the input: 1111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111110000000000

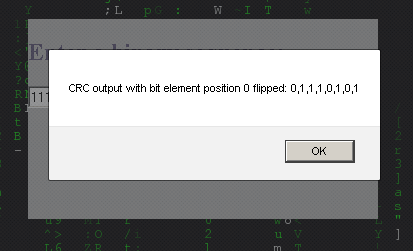
Screenshot showing the program asking for the user’s Input.



Screenshot showing CRC output with bit position 0 as a bit error.



Screenshot showing CRC output with bit position 0 as a bit error.



After many CRC input attempts trying to see if any error goes undetected, the program would always catch the errors by showing a different bit sequence than previously inputted. The probability of an undetected error occurring is not likely.

# Equipment Used

* Windows 7
* Firefox – version 7.0.1
* Notepad++

# Report

This lab helped me to understand how checksum works as to detect any errors that may be found in a bit sequence. As I programmed the code for each part of the lab I was able to see how to look at each bit sequence and make comparisons as to check if the bit sequence were the same or not.

I also saw and understood how CRC works by looking at the pseudo code attached in this report I was able to follow it and code the logic behind a CRC check sum.

I have learned that on each step of this lab it is not likely for the program to pass by an error without noticing it, it is a very strong logic that works every single time.

# Conclusion

Error coding and correction are very important techniques that enable reliable delivery of digital data over unreliable communication channels. A lot of communication channels are subject to channel noise, and therefore errors may be introduced during transmission from source to receiver. Error detection techniques allow detecting errors, while error correction supports reconstruction of the original data.

The most meaningful lesson learned in this lab was to understand how a CRC check sum works and how Parity works in the detection of errors. I enjoyed coding this lab and learned a great deal about error detection algorithms.

Error detection techniques are found in many locations such as downloads of internet programs, many websites let you download their programs and in order for the user to check if the download was good or not is by using tools such as check sums, if it is the same number of bits it was a good download otherwise you may need to start all over again to avoid corrupted software.

# References

Lab 10 instructions: <http://it327.groups.et.byu.net/>

Lab 10 example code: <http://it327.groups.et.byu.net/>

TA: Michael Hughes

**Name: Lehi Alcantara**

**Completion Date: 11/08/2011**