Name: Malhar Mahajan

NetId: B00934337

Assignment 3: CSCI 4171

1. CRC Warm-up:

a) The message M(x) = 110100111101 and generator polynomial G(x) = 1011.

Compute the transmitted bit string

Answer:

- **Given**: $M = 110100111101$, $G = 1011$(degree $r = 3$).
- **Method**:
  1. Append $r$zeros to M: $M \cdot x^r = 110100111101\ \underline{000}$
  2. Do modulo-2 long division by G(XOR whenever the current bit is 1).
  3. The **remainder** (3 bits) is the CRC; append it to original $M$to get P.
- **Result (I computed it)**:
  - Remainder = **000**
  - Transmitted frame P= **110100111101000**

b) The data string received is 10110011101 with G(x) = 1001.
Determine if an error occurred.

Answer:

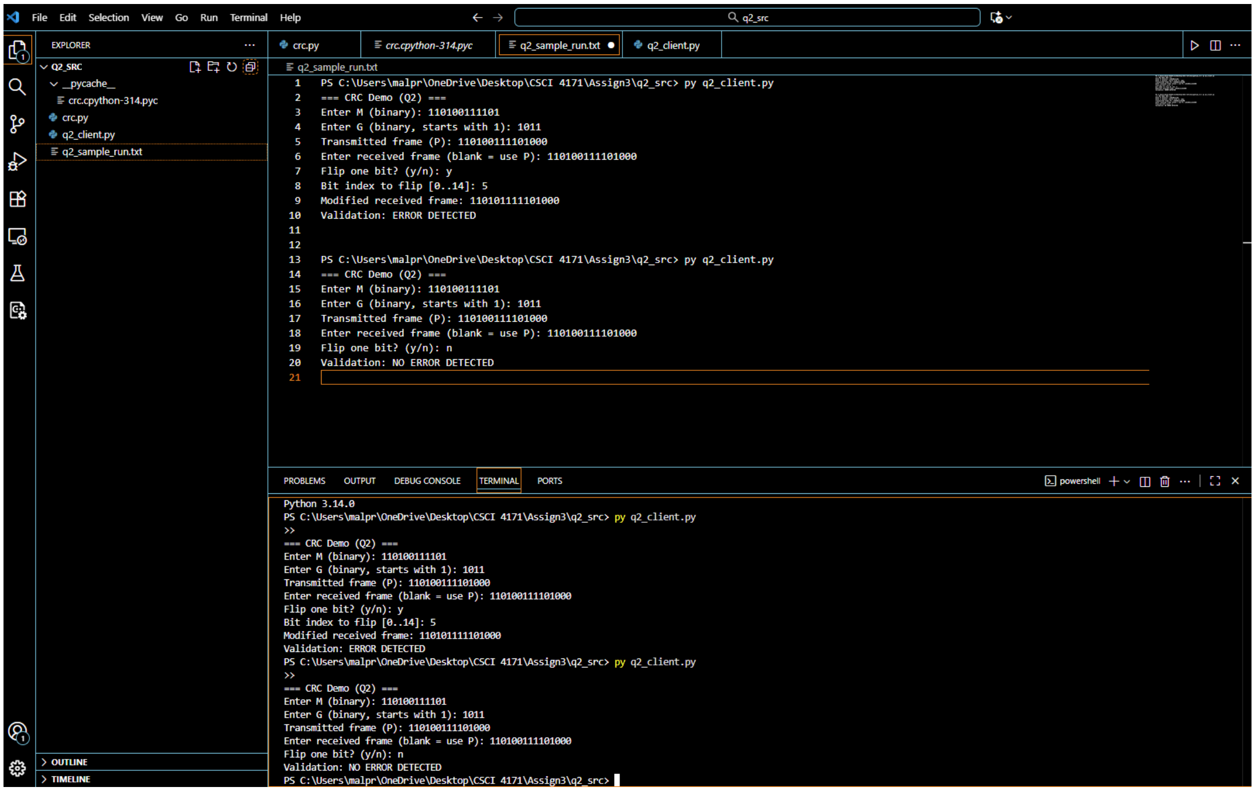
- **Given**: Received frame $R = 10110011101$, $G = 1001$.
- **Method**: Divide $R$by G. If the remainder is **all zeros**, assume "no error"; otherwise "error detected."
- **Result (I computed it)**: remainder = **010** → **Error occurred**.

2. CRC Simulation

For this part, I implemented a Python program that simulates both the sender and receiver sides of a CRC system.
The sender takes a message $M(x)$and a generator polynomial $G(x)$represented as binary strings. It appends $r = \text{len}(G) - 1$zeros, performs modulo-2 division to find the remainder, and transmits the concatenated frame $P(x) = M(x) \parallel R(x)$.
The receiver divides the received frame by the same generator and checks if the remainder is all zeros — indicating no detected errors.

Results:

- When no bit was flipped, the validation output was “NO ERROR DETECTED.”
- When a single bit was flipped, the receiver detected the corruption and printed “ERROR DETECTED.”



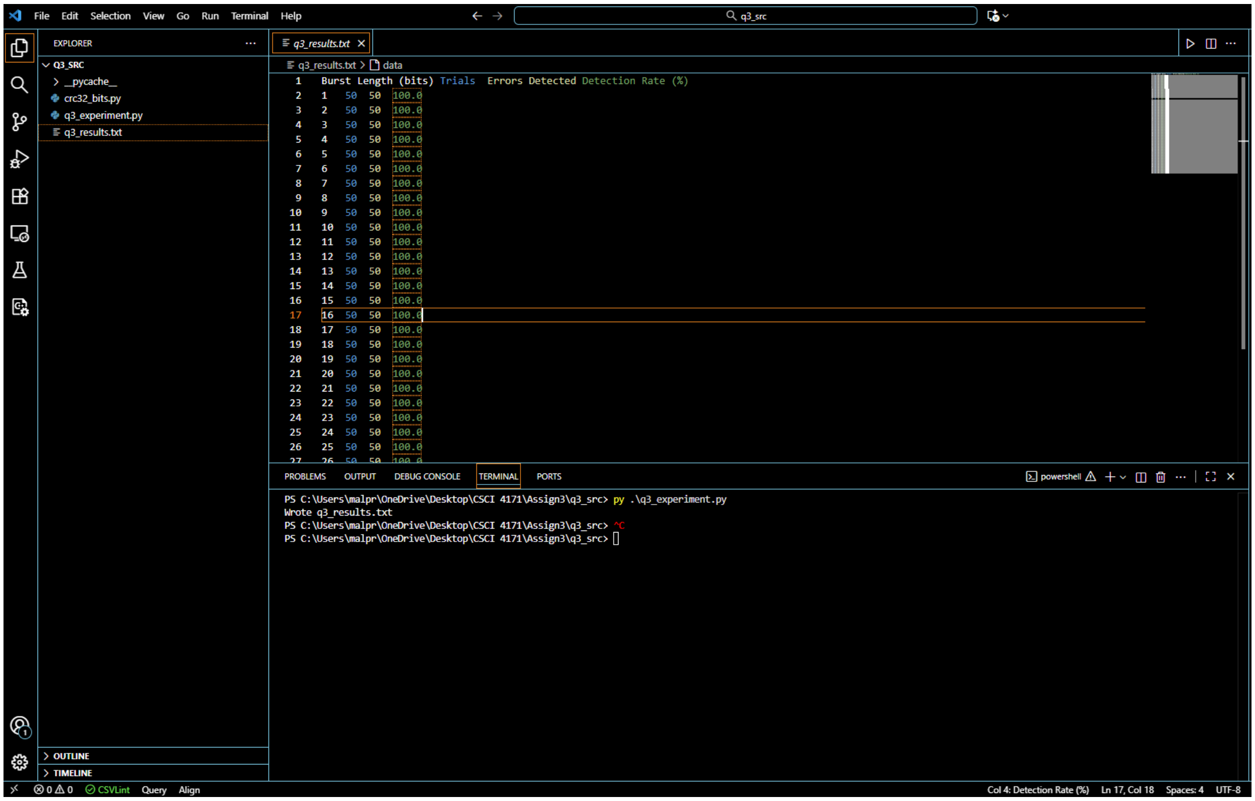
```
PS C:\Users\malpr\OneDrive\Desktop\CSCI 4171\Assign3\q2_src> py q2_client.py
=== CRC Demo (Q2) ===
Enter M (binary): 110100111101
Enter G (binary, starts with 1): 1011
Transmitted frame (P): 110100111101000
Enter received frame (blank = use P): 110100111101000
Flip one bit? (y/n): y
Bit index to flip [0..14]: 5
Modified received frame: 110101111101000
Validation: ERROR DETECTED


PS C:\Users\malpr\OneDrive\Desktop\CSCI 4171\Assign3\q2_src> py q2_client.py
=== CRC Demo (Q2) ===
Enter M (binary): 110100111101
Enter G (binary, starts with 1): 1011
Transmitted frame (P): 110100111101000
Enter received frame (blank = use P): 110100111101000
Flip one bit? (y/n): n
Validation: NO ERROR DETECTED
```

```
Python 3.14.0
PS C:\Users\malpr\OneDrive\Desktop\CSCI 4171\Assign3\q2_src> py q2_client.py
>>
=== CRC Demo (Q2) ===
Enter M (binary): 110100111101
Enter G (binary, starts with 1): 1011
Transmitted frame (P): 110100111101000
Enter received frame (blank = use P): 110100111101000
Flip one bit? (y/n): y
Bit index to flip [0..14]: 5
Modified received frame: 110101111101000
Validation: ERROR DETECTED
PS C:\Users\malpr\OneDrive\Desktop\CSCI 4171\Assign3\q2_src> py q2_client.py
>>
=== CRC Demo (Q2) ===
Enter M (binary): 110100111101
Enter G (binary, starts with 1): 1011
Transmitted frame (P): 110100111101000
Enter received frame (blank = use P): 110100111101000
Flip one bit? (y/n): n
Validation: NO ERROR DETECTED
PS C:\Users\malpr\OneDrive\Desktop\CSCI 4171\Assign3\q2_src>
```

3. CRC in Action: Error Detection Capability

I used the canonical, non-reflected CRC-32 polynomial (0x04C11DB7). For burst lengths $L = 1..64$, I ran 50 trials each by choosing a random start index and forcing that span to all 0s or all 1s, then checked the receiver remainder.

Results: Detection was **100%** across all lengths. This aligns with theory: a degree-32 CRC **guarantees detection** of **all bursts with** $L \leq 32$. For $L > 32$, undetected errors are possible but occur with probability ≈ 2^{-32}per random pattern, so with only 50 trials it’s common to still observe **100%** detection empirically.
Conclusion: CRC-32 provides complete coverage for burst errors up to 32 bits and extremely high practical coverage beyond 32 bits in typical scenarios.



```
Burst Length (bits)	Trials	Errors Detected	Detection Rate (%)
1	50	50	100.0
2	50	50	100.0
3	50	50	100.0
4	50	50	100.0
5	50	50	100.0
6	50	50	100.0
7	50	50	100.0
8	50	50	100.0
9	50	50	100.0
10	50	50	100.0
11	50	50	100.0
12	50	50	100.0
13	50	50	100.0
14	50	50	100.0
15	50	50	100.0
16	50	50	100.0
17	50	50	100.0
18	50	50	100.0
19	50	50	100.0
20	50	50	100.0
21	50	50	100.0
22	50	50	100.0
23	50	50	100.0
24	50	50	100.0
25	50	50	100.0
26	50	50	100.0
```

```
PS C:\Users\malpr\OneDrive\Desktop\CSCI 4171\Assign3\q3_src> py .\q3_experiment.py
Wrote q3_results.txt
PS C:\Users\malpr\OneDrive\Desktop\CSCI 4171\Assign3\q3_src> ^C
PS C:\Users\malpr\OneDrive\Desktop\CSCI 4171\Assign3\q3_src>
```