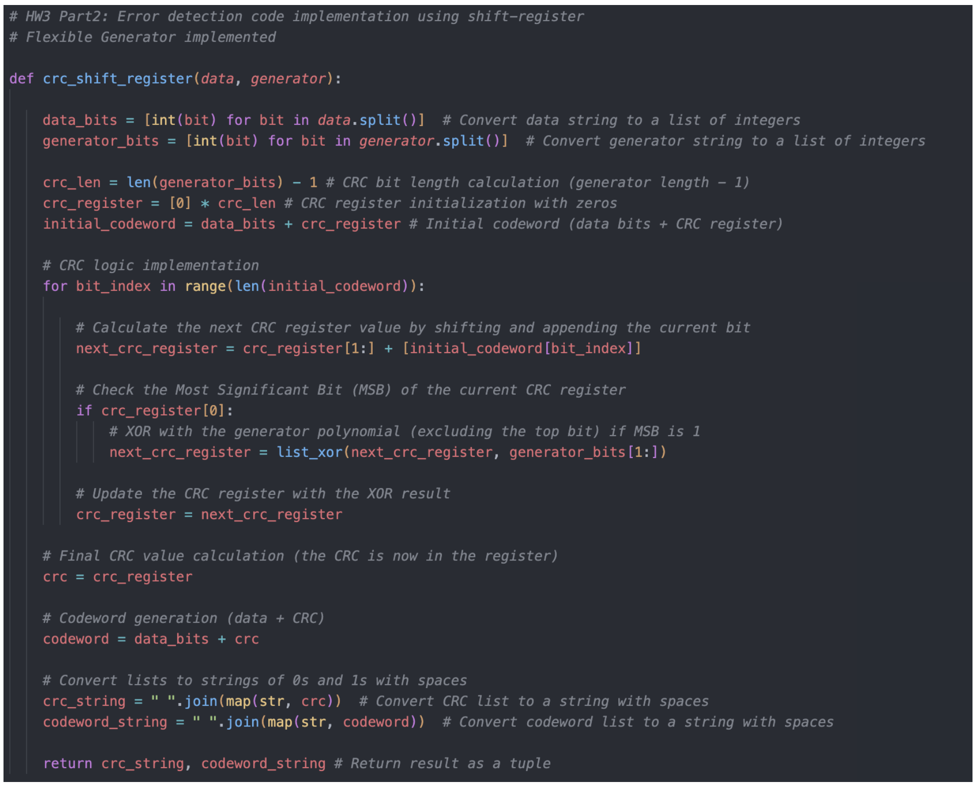
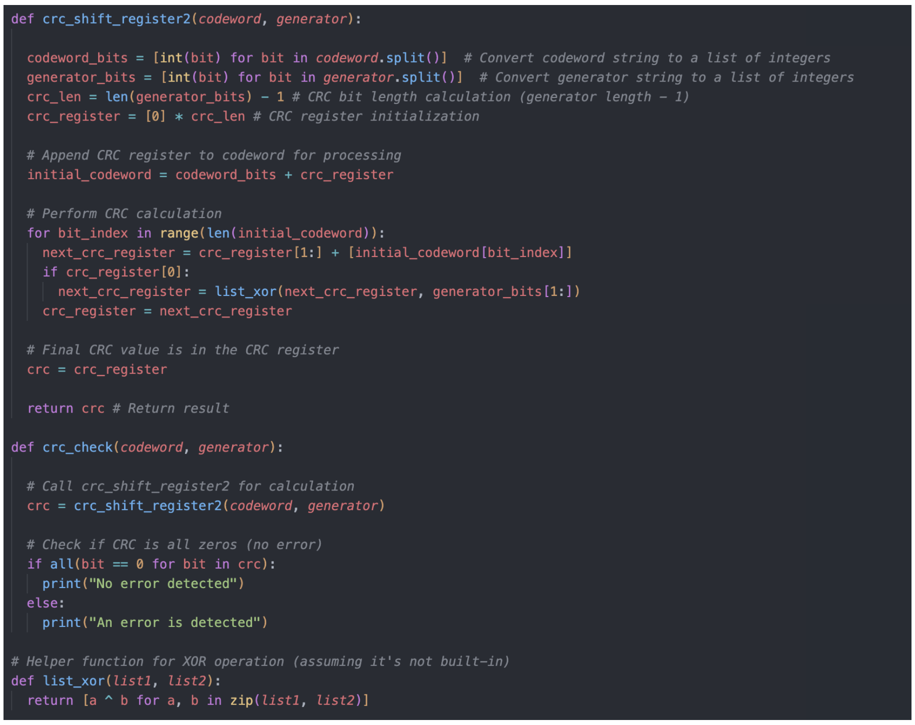
Error Detection Implement Report

1. Code Analysis

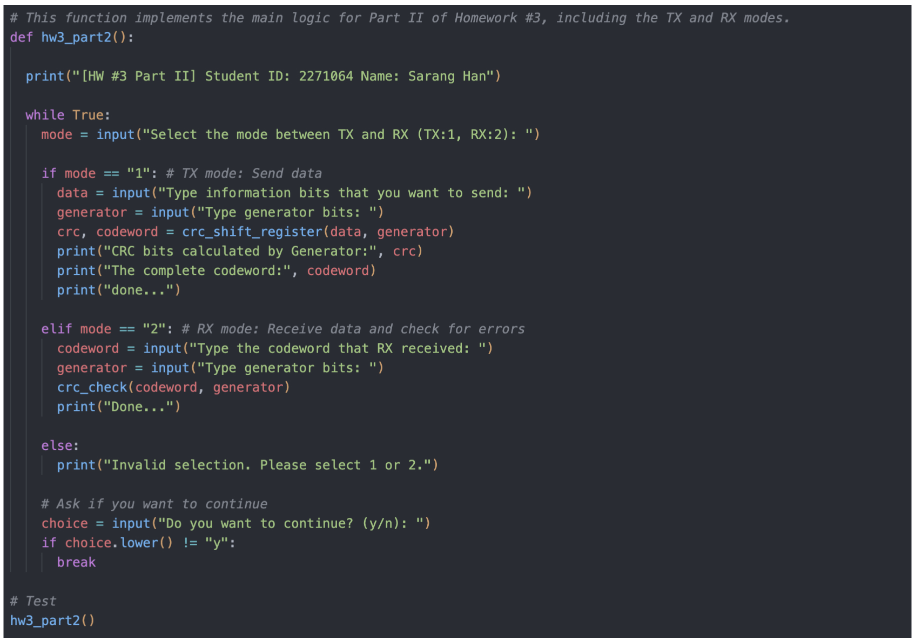
Resource Code



```python
# HW3 Part2: Error detection code implementation using shift-register
# Flexible Generator implemented

def crc_shift_register(data, generator):

    data_bits = [int(bit) for bit in data.split()]  # Convert data string to a list of integers
    generator_bits = [int(bit) for bit in generator.split()]  # Convert generator string to a list of integers

    crc_len = len(generator_bits) - 1 # CRC bit length calculation (generator length - 1)
    crc_register = [0] * crc_len # CRC register initialization with zeros
    initial_codeword = data_bits + crc_register # Initial codeword (data bits + CRC register)

    # CRC logic implementation
    for bit_index in range(len(initial_codeword)):

        # Calculate the next CRC register value by shifting and appending the current bit
        next_crc_register = crc_register[1:] + [initial_codeword[bit_index]]

        # Check the Most Significant Bit (MSB) of the current CRC register
        if crc_register[0]:
            # XOR with the generator polynomial (excluding the top bit) if MSB is 1
            next_crc_register = list_xor(next_crc_register, generator_bits[1:])

        # Update the CRC register with the XOR result
        crc_register = next_crc_register

    # Final CRC value calculation (the CRC is now in the register)
    crc = crc_register

    # Codeword generation (data + CRC)
    codeword = data_bits + crc

    # Convert lists to strings of 0s and 1s with spaces
    crc_string = " ".join(map(str, crc))  # Convert CRC list to a string with spaces
    codeword_string = " ".join(map(str, codeword))  # Convert codeword list to a string with spaces

    return crc_string, codeword_string # Return result as a tuple
```



```python
def crc_shift_register2(codeword, generator):

  codeword_bits = [int(bit) for bit in codeword.split()]  # Convert codeword string to a list of integers
  generator_bits = [int(bit) for bit in generator.split()]  # Convert generator string to a list of integers
  crc_len = len(generator_bits) - 1 # CRC bit length calculation (generator length - 1)
  crc_register = [0] * crc_len # CRC register initialization

  # Append CRC register to codeword for processing
  initial_codeword = codeword_bits + crc_register

  # Perform CRC calculation
  for bit_index in range(len(initial_codeword)):
    next_crc_register = crc_register[1:] + [initial_codeword[bit_index]]
    if crc_register[0]:
      next_crc_register = list_xor(next_crc_register, generator_bits[1:])
    crc_register = next_crc_register

  # Final CRC value is in the CRC register
  crc = crc_register

  return crc # Return result

def crc_check(codeword, generator):

  # Call crc_shift_register2 for calculation
  crc = crc_shift_register2(codeword, generator)

  # Check if CRC is all zeros (no error)
  if all(bit == 0 for bit in crc):
    print("No error detected")
  else:
    print("An error is detected")

# Helper function for XOR operation (assuming it's not built-in)
def list_xor(list1, list2):
  return [a ^ b for a, b in zip(list1, list2)]
```



```python
# This function implements the main logic for Part II of Homework #3, including the TX and RX modes.
def hw3_part2():

  print("[HW #3 Part II] Student ID: 2271064 Name: Sarang Han")

  while True:
    mode = input("Select the mode between TX and RX (TX:1, RX:2): ")

    if mode == "1": # TX mode: Send data
      data = input("Type information bits that you want to send: ")
      generator = input("Type generator bits: ")
      crc, codeword = crc_shift_register(data, generator)
      print("CRC bits calculated by Generator:", crc)
      print("The complete codeword:", codeword)
      print("done...")

    elif mode == "2": # RX mode: Receive data and check for errors
      codeword = input("Type the codeword that RX received: ")
      generator = input("Type generator bits: ")
      crc_check(codeword, generator)
      print("Done...")

    else:
      print("Invalid selection. Please select 1 or 2.")

    # Ask if you want to continue
    choice = input("Do you want to continue? (y/n): ")
    if choice.lower() != "y":
      break

# Test
hw3_part2()
```

crc_shift_register Function

The `crc_shift_register` function calculates the CRC value for a given data string and generator polynomial. It takes two input parameters: `data`, which represents the information bits to be transmitted, and `generator`, which is the generator polynomial used for CRC calculation.

1. Data and Generator Initialization:

   ```
   data_bits = [int(bit) for bit in data.split()]
    generator_bits = [int(bit) for bit in generator.split()]
   ```

   The `data` and `generator` strings are split into individual bits and converted to integers. For example, if `data = "1011"` and `generator = "1101"`, then `data_bits = [1, 0, 1, 1]` and `generator_bits = [1, 1, 0, 1]`.

2. CRC Length Calculation:

   ```
   crc_len = len(generator_bits) - 1
   ```

   The length of the CRC register is determined by the length of the generator polynomial minus 1. This is because the generator polynomial includes the highest degree term, which is always 1.

3. CRC Register Initialization:

   ```
   crc_register = [0] * crc_len
   ```

   The CRC register is initialized with zeros. The length of the register is determined by `crc_len`.

4. Initial Codeword Creation:

   ```
   initial_codeword = data_bits + crc_register
   ```

   The initial codeword is formed by concatenating the `data_bits` with the `crc_register`. This codeword will be processed to calculate the CRC value.

CRC Calculation Loop

1. Loop Through Each Bit:

   ```
   for bit_index in range(len(initial_codeword)):
   ```

   The code iterates through each bit of the initial codeword.

2. Calculate Next CRC Register Value:

   ```
   next_crc_register = crc_register[1:] + [initial_codeword[bit_index]]
   ```

   The next CRC register value is calculated by shifting the current CRC register one position to the right (`crc_register[1:]`) and appending the current bit from the initial codeword (`initial_codeword[bit_index]`).

3. Check Most Significant Bit (MSB):

   ```
   if crc_register[0]:
   ```

   The code checks the MSB of the current CRC register. If it is 1, it indicates that the CRC register needs to be XORed with the generator polynomial.

4. XOR with Generator Polynomial:

   ```
   next_crc_register = list_xor(next_crc_register, generator_bits[1:])
   ```

   If the MSB is 1, the `next_crc_register` is XORed with the generator polynomial (excluding the highest degree term). This step ensures that the CRC value aligns with the generator polynomial.

5. Update CRC Register:

   ```
   crc_register = next_crc_register
   ```

   The CRC register is updated with the new value calculated in the previous step.

Final CRC and Codeword Generation

1. Final CRC Calculation:

   ```
   crc = crc_register
   ```

   After processing all the bits, the final CRC value is stored in the `crc_register`.

2. Codeword Generation:

   ```
   codeword = data_bits + crc
   ```

   The final codeword is generated by concatenating the original `data_bits` with the calculated `crc`.

Conversion to String Format

1. CRC and Codeword Conversion:

   ```
   crc_string = " ".join(map(str, crc))
    codeword_string = " ".join(map(str, codeword))
   ```

   The CRC and codeword lists are converted to strings, with spaces separating each bit. For example, if `crc = [0, 1, 0, 1]`, then `crc_string = "0 1 0 1"`.

Return Result

1. Return CRC and Codeword:

   ```
   return crc_string, codeword_string
   ```

   The function returns the calculated CRC and codeword as a tuple.

crc_shift_register2 Function

The `crc_shift_register2` function is similar to `crc_shift_register`, but it takes a codeword as input instead of separate data and generator parameters. It calculates the CRC value for the given codeword using the same logic as the previous function.

list_xor Function

The `list_xor` function performs an element-wise XOR operation on two lists. It takes two input lists and returns a new list containing the XOR result. This function is used to XOR the CRC register with the generator polynomial during the CRC calculation.

crc_check Function

The `crc_check` function is responsible for checking errors in a received codeword using the CRC value. It takes two input parameters: `codeword`, which is the received codeword, and `generator`, which is the generator polynomial.

1. CRC Calculation:

   ```
   crc = crc_shift_register2(codeword, generator)
   ```

   The function calls `crc_shift_register2` to calculate the CRC value for the received codeword.

2. Error Checking:

   ```
   if all(bit == 0 for bit in crc):
       print("No error is detected! (according to generator)")
    else:
       print("An error is detected! (according to generator)")
   ```

   The code checks if all the bits in the calculated CRC are zero. If all bits are zero, it indicates that no error was detected during transmission. Otherwise, an error is detected.

hw3_part2 Function

The `hw3_part2` function implements the main logic for Part II of the homework assignment. It provides a user interface for selecting the mode (TX or RX) and performing the corresponding operations.

1. User Input for Mode:

   ```
   mode = input("Select the mode between TX and RX (TX:1, RX:2): ")
   ```

   The user is prompted to select the mode: TX for transmitting data or RX for receiving and checking data.

2. TX Mode:

   - Data and Generator Input:

     ```
     data = input("Type information bits that you want to send: ")
     ```

```
generator = input("Type generator bits: ")
```

In TX mode, the user is prompted to enter the information bits they want to send and the generator polynomial.

- CRC and Codeword Calculation:

```
crc, codeword = crc_shift_register(data, generator)
```

The `crc_shift_register` function is called to calculate the CRC value and generate the codeword.

- Display Results:

```
print("CRC bits calculated by Generator:", crc)
 print("The complete codeword:", codeword)
```

The calculated CRC and the complete codeword are displayed to the user.

3. RX Mode:

- Codeword and Generator Input:

```
codeword = input("Type the codeword that RX received: ")
 generator = input("Type generator bits: ")
```

In RX mode, the user is prompted to enter the received codeword and the generator polynomial.

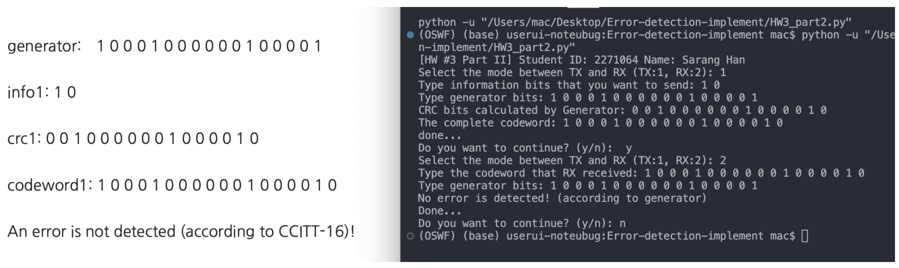
- Error Checking:

```
crc_check(codeword, generator)
```

The `crc_check` function is called to check for errors in the received codeword.

2. Experimental Results

CCITT-16 Example



generator: 1 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 1

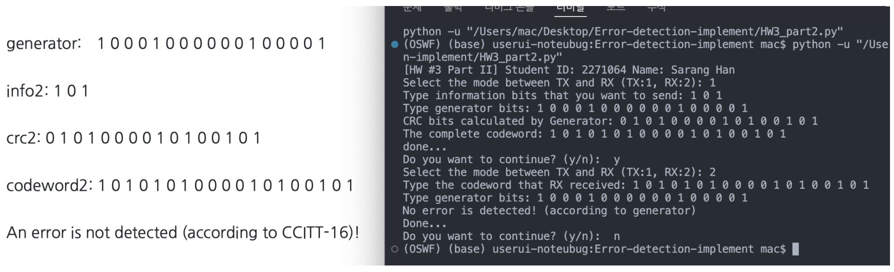
info1: 1 0

crc1: 0 0 1 0 0 0 0 0 0 1 0 0 0 0 1 0

codeword1: 1 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 1 0

An error is not detected (according to CCITT-16)!

```
python -u "/Users/mac/Desktop/Error-detection-implement/HW3_part2.py"
● (OSWF) (base) userui-noteubug:Error-detection-implement mac$ python -u "/Use
n-implement/HW3_part2.py"
[HW #3 Part II] Student ID: 2271064 Name: Sarang Han
Select the mode between TX and RX (TX:1, RX:2): 1
Type information bits that you want to send: 1 0
Type generator bits: 1 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 1
CRC bits calculated by Generator: 0 0 1 0 0 0 0 0 0 1 0 0 0 0 1 0
The complete codeword: 1 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 1 0
done...
Do you want to continue? (y/n):  y
Select the mode between TX and RX (TX:1, RX:2): 2
Type the codeword that RX received: 1 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 1 0
Type generator bits: 1 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 1
No error is detected! (according to generator)
Done...
Do you want to continue? (y/n): n
○ (OSWF) (base) userui-noteubug:Error-detection-implement mac$ 
```



generator: 1 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 1

info2: 1 0 1

crc2: 0 1 0 1 0 0 0 0 1 0 1 0 0 1 0 1

codeword2: 1 0 1 0 1 0 1 0 0 0 0 1 0 1 0 0 1 0 1

An error is not detected (according to CCITT-16)!

```
python -u "/Users/mac/Desktop/Error-detection-implement/HW3_part2.py"
● (OSWF) (base) userui-noteubug:Error-detection-implement mac$ python -u "/Use
n-implement/HW3_part2.py"
[HW #3 Part II] Student ID: 2271064 Name: Sarang Han
Select the mode between TX and RX (TX:1, RX:2): 1
Type information bits that you want to send: 1 0 1
Type generator bits: 1 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 1
CRC bits calculated by Generator: 0 1 0 1 0 0 0 0 1 0 1 0 0 1 0 1
The complete codeword: 1 0 1 0 1 0 1 0 0 0 0 1 0 1 0 0 1 0 1
done...
Do you want to continue? (y/n):  y
Select the mode between TX and RX (TX:1, RX:2): 2
Type the codeword that RX received: 1 0 1 0 1 0 1 0 0 0 0 1 0 1 0 0 1 0 1
Type generator bits: 1 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 1
No error is detected! (according to generator)
Done...
Do you want to continue? (y/n):  n
○ (OSWF) (base) userui-noteubug:Error-detection-implement mac$ 
```

Different Generator Example

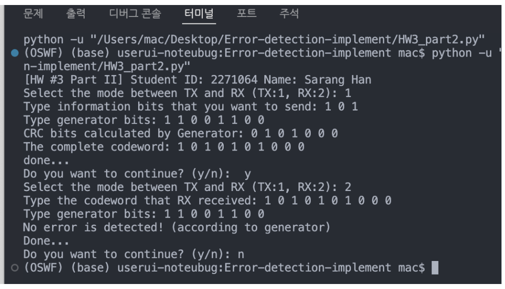
generator: 1 1 0 0 1 1 0 0

info6: 1 0 1

crc6: 0 1 0 1 0 0 0

codeword6: 1 0 1 0 1 0 1 0 0 0

An error is not detected!



```
python -u "/Users/mac/Desktop/Error-detection-implement/HW3_part2.py"
● (OSWF) (base) userui-noteubug:Error-detection-implement mac$ python -u "
n-implement/HW3_part2.py"
[HW #3 Part II] Student ID: 2271064 Name: Sarang Han
Select the mode between TX and RX (TX:1, RX:2): 1
Type information bits that you want to send: 1 0 1
Type generator bits: 1 1 0 0 1 1 0 0
CRC bits calculated by Generator: 0 1 0 1 0 0 0
The complete codeword: 1 0 1 0 1 0 1 0 0 0
done...
Do you want to continue? (y/n):  y
Select the mode between TX and RX (TX:1, RX:2): 2
Type the codeword that RX received: 1 0 1 0 1 0 1 0 0 0
Type generator bits: 1 1 0 0 1 1 0 0
No error is detected! (according to generator)
Done...
Do you want to continue? (y/n): n
○ (OSWF) (base) userui-noteubug:Error-detection-implement mac$
```

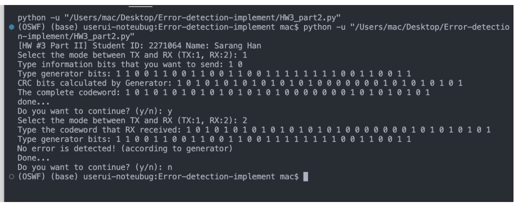
generator: 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 1 1 1 1 1 1 0 0 1 1 0 0 1 1

info7: 1 0

crc7: 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 0 0 0 0 0 0 1 0 1 0 1 0 1 0 1

codeword7: 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 0 0 0 0 0 0 1 0 1 0 1 0 1 0 1

An error is not detected!



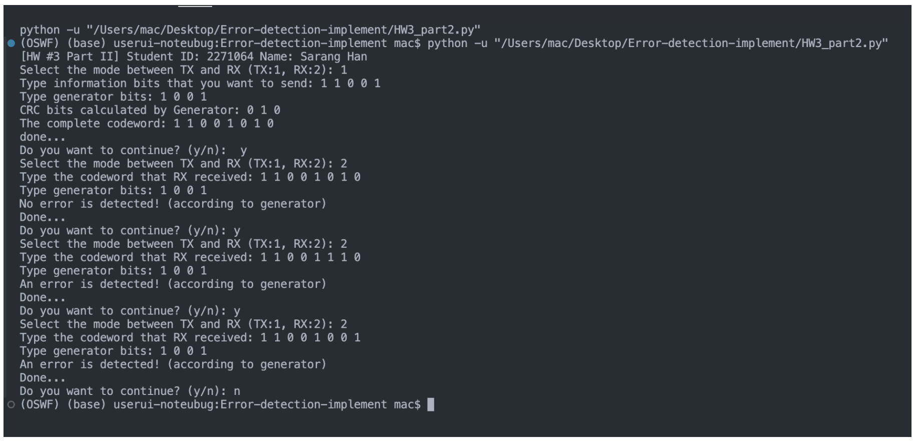
```
python -u "/Users/mac/Desktop/Error-detection-implement/HW3_part2.py"
● (OSWF) (base) userui-noteubug:Error-detection-implement mac$ python -u "/Users/mac/Desktop/Error-detectio
n-implement/HW3_part2.py"
[HW #3 Part II] Student ID: 2271064 Name: Sarang Han
Select the mode between TX and RX (TX:1, RX:2): 1
Type information bits that you want to send: 1 0
Type generator bits: 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 1 1 1 1 1 1 0 0 1 1 0 0 1 1
CRC bits calculated by Generator: 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 0 0 0 0 0 0 1 0 1 0 1 0 1 0 1
The complete codeword: 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 0 0 0 0 0 0 1 0 1 0 1 0 1 0 1
done...
Do you want to continue? (y/n): y
Select the mode between TX and RX (TX:1, RX:2): 2
Type the codeword that RX received: 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 0 0 0 0 0 0 1 0 1 0 1 0 1 0 1
Type generator bits: 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 1 1 1 1 1 1 0 0 1 1 0 0 1 1
No error is detected! (according to generator)
Done...
Do you want to continue? (y/n): n
○ (OSWF) (base) userui-noteubug:Error-detection-implement mac$
```

Error Detection Test

Data: 1 1 0 0 1

Generator: 1 0 0 1

Codeword: 1 1 0 0 1 0 1 0



```
python -u "/Users/mac/Desktop/Error-detection-implement/HW3_part2.py"
● (OSWF) (base) userui-noteubug:Error-detection-implement mac$ python -u "/Users/mac/Desktop/Error-detection-implement/HW3_part2.py"
[HW #3 Part II] Student ID: 2271064 Name: Sarang Han
Select the mode between TX and RX (TX:1, RX:2): 1
Type information bits that you want to send: 1 1 0 0 1
Type generator bits: 1 0 0 1
CRC bits calculated by Generator: 0 1 0
The complete codeword: 1 1 0 0 1 0 1 0
done...
Do you want to continue? (y/n):  y
Select the mode between TX and RX (TX:1, RX:2): 2
Type the codeword that RX received: 1 1 0 0 1 0 1 0
Type generator bits: 1 0 0 1
No error is detected! (according to generator)
Done...
Do you want to continue? (y/n): y
Select the mode between TX and RX (TX:1, RX:2): 2
Type the codeword that RX received: 1 1 0 0 1 1 1 0
Type generator bits: 1 0 0 1
An error is detected! (according to generator)
Done...
Do you want to continue? (y/n): y
Select the mode between TX and RX (TX:1, RX:2): 2
Type the codeword that RX received: 1 1 0 0 1 0 0 1
Type generator bits: 1 0 0 1
An error is detected! (according to generator)
Done...
Do you want to continue? (y/n): n
○ (OSWF) (base) userui-noteubug:Error-detection-implement mac$
```

result:

1. No error → No error is detected!
2. 1 Error → An error is detected!
3. 2 Error → An error is detected!