**Batch: A-3 Roll No.: 16010122104**

**Experiment / assignment / tutorial No. 4**

**Grade: AA / AB / BB / BC / CC / CD /DD**

**Signature of the Staff In-charge with date**

**Experiment No.:4**

|  |
| --- |
| **TITLE:** Implementation of CRC & Checksum for Computer Networks |

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**AIM:** To implement Layer 2 Error Control schemes: CRC & Checksum.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Expected Outcome of Experiment:**

**CO:**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Books/ Journals/ Websites referred:**

1. A. S. Tanenbaum, “Computer Networks”, Pearson Education, Fourth Edition
2. B. A. Forouzan, “Data Communications and Networking”, TMH, Fourth Edition

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Pre Lab/ Prior Concepts:**

Data Link Layer, Error Correction/Detection, Types of Errors

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**New Concepts to be learned:** Checksum.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**CRC(Cyclic Redundancy Check):**

Cyclic Redundancy Check (CRC) is another error detection technique to detect errors in data that has been transmitted on a communications link. A sending device applies a 16 or 32 bit polynomial to a block of data that is to be transmitted and appends the resulting cyclic redundancy check (CRC) to the block. The receiving end applies the same polynomial to the data and compares its result with the result appended by the sender. If they agree, the data has been received successfully. If not, the sender can be notified to resend the block of data.

**At Sender Side:**

* Sender has a generator G(x) polynomial.
* Sender appends (n-1) zero bits to the data.

Where, n= no of bits in generator

* Dividend appends the data with generator G(x) using modulo 2 division (arithmetic).
* Remainder of (n-1) bits will be CRC.

**Codeword:** It is combined form of Data bits and CRC bits i.e. Codeword = Data bits + CRC bits.

**Example**

Assume that –

(a) data is 10110.

(b) code generator is 1101.

(Code generator can also be mentioned in polynomial : x3+x2+1 )

**Calculate CRC Bits:** While calculating the CRC bits, we pad (n-1) 0’s to the message bits, where ‘n’ = no of bits in the code generator.

Cyclic Redundancy check will be generated as shown below –

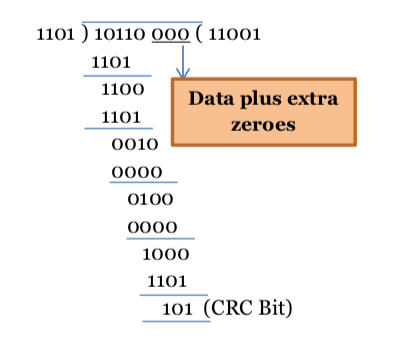


Figure 1: CRC calculation by sender

**At Receiver Side**

* Receiver has same generator G(x).
* Receiver divides received data (data + CRC) with generator.
* If remainder is zero, data is correctly received.
* Else, there is error.

Assume the received message is 10110110.

**Calculate CRC Bits:** It does not add any padding bits, rather calculates from the entire received code word.

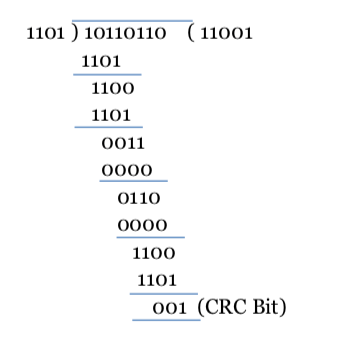


Figure 2: CRC calculation by receiver

The CRC bits are calculated to be different. Thus, there is an error detected.

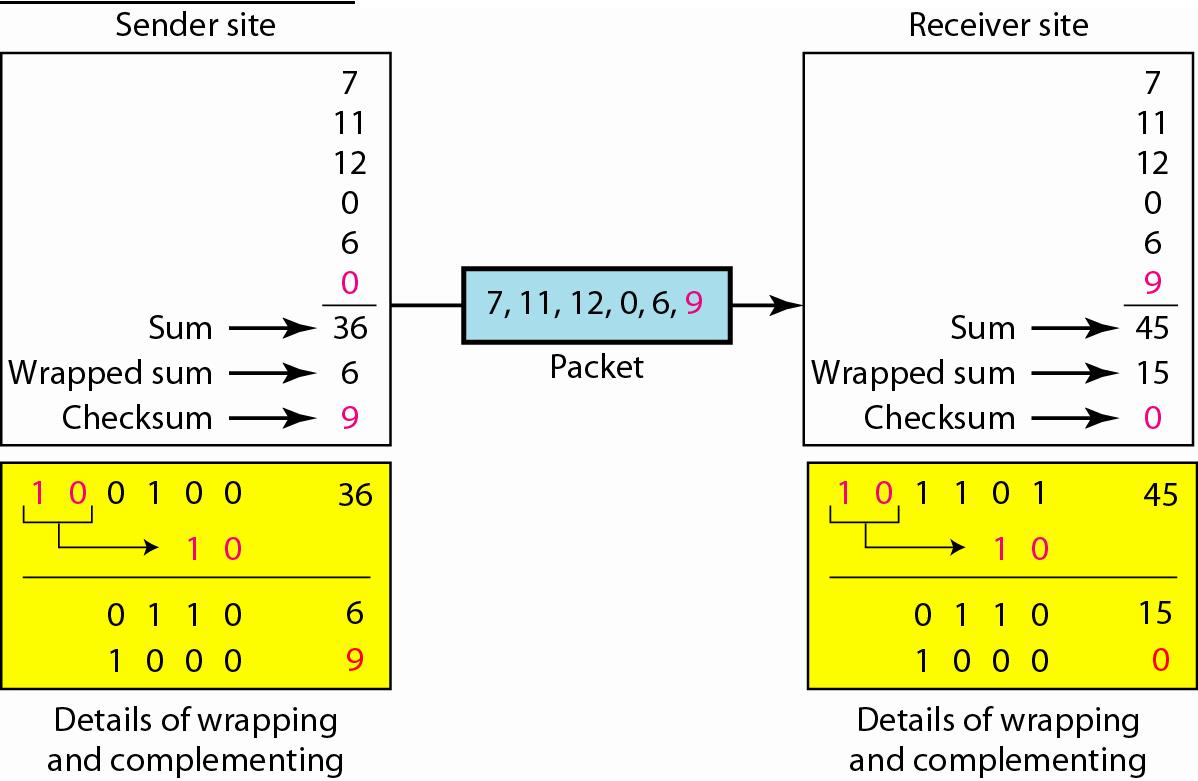
**Internet Checksum :**

A checksum is a simple type of redundancy check that is used to detect errors in data.

Errors frequently occur in data when it is written to a disk, transmitted across a network or otherwise manipulated. The errors are typically very small, for example, a single incorrect bit, but even such small errors can greatly affect the quality of data, and even make it useless.

In its simplest form, a checksum is created by calculating the binary values in a packet or other block of data using some algorithm and storing the results with the data. When the data is retrieved from memory or received at the other end of a network, a new checksum is calculated and compared with the existing checksum. A non-match indicates an error; a match does not necessarily mean the absence of errors, but only that the simple algorithm was not able to detect any.

**Simple Checksum:**



**Internet Checksum**

The following process generates Internet Checksum

Assume the packet header is: 01 00 F2 03 F4 F5 F6 F7 00 00

(00 00 is the checksum to be calculated)

The first step is to form 16-bit words.

0100 F203 F4F5 F6F7

The second step is to calculate the sum using 32-bits.

0100 + F203 + F4F5 + F6F7 = 0002 DEEF

The third step is to add the carries (0002) to the 16-bit sum.

DEEF + 002 = DEF1

The fourth step is to take the complement. (1s becomes 0s and 0s become 1s)

~DEF1 = 210E

So the checksum is 21 0E.

The packet header is sent as: 01 00 F2 03 F4 F5 F6 F7 21 0E

\* At the receiver, the steps are repeated.

The first step is to form 16-bit words.

0100 F203 F4F5 F6F7 210E

The second step is to calculate the sum using 32-bits.

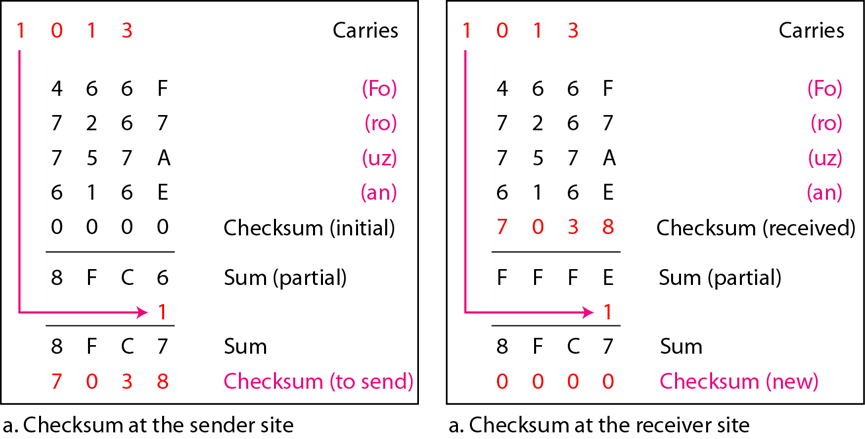
0100 + F203 + F4F5 + F6F7 + 210E = 0002 FFFD

The third step is to add the carries (0002) to the 16-bit sum.

FFFD + 0002 = FFFF which means that no error was detected.

(In 1s complement, zero is 0000 or FFFF.)

**Example:**

****

**IMPLEMENTATION:** (printout of codes)

**CHECKSUM**

print("SENDER - ")

word = input("Enter the word: ")

hexcode= []

if len(word)%2 == 0:

    for i in range (int(len(word)/2)):

        hexcode.append(hex(ord(word[2\*i])) + hex(ord(word[2\*i+1])))

else:

    hexcode.append("00"+hex(ord(word[0])))

    for i in range (1,int(len(word)/2)+1):

        hexcode.append(hex(ord(word[2\*i-1])) + hex(ord(word[2\*i])))

print("Hex strings for the input: ")

for i in range (len(hexcode)):

    hexcode[i].replace('0x',"")

    print(hexcode[i])

sum = "0000"

for i in range (len(hexcode)):

    hexcode1 = str(hexcode[i]).replace("0x","")

    temp = int(sum,16) + int(hexcode1,16)

    sum = hex(temp)

sum = str(sum.replace("0x",""))

print("Sum: ",sum)

if len(sum)>4:

    n = len(sum)-4

    sum1 = (sum[0:n])

    sum2 = (sum[n:])

    temp = int(sum1,16) + int(sum2,16)

    sum = hex(temp)

comp = int("FFFF",16) - int(sum,16)

sum = hex(comp)

print("The checksum is: ",sum )

x = sum

print("RECEIVER - ")

word = input("Enter the word: ")

hexcode = []

if len(word)%2 == 0:

    for i in range (int(len(word)/2)):

        hexcode.append(hex(ord(word[2\*i])) + hex(ord(word[2\*i+1])))

else:

    hexcode.append("00"+hex(ord(word[0])))

    for i in range (1,int(len(word)/2)+1):

        hexcode.append(hex(ord(word[2\*i-1])) + hex(ord(word[2\*i])))

print("Hex strings for the input: ")

for i in range (len(hexcode)):

    hexcode[i].replace('0x',"")

    print(hexcode[i])

sum = "0000"

for i in range (len(hexcode)):

    hexcode1 = str(hexcode[i]).replace("0x","")

    temp = int(sum,16) + int(hexcode1,16)

    sum = hex(temp)

sum = str(sum.replace("0x",""))

print("Sum: ",sum)

if len(sum)>4:

    n = len(sum)-4

    sum1 = (sum[0:n])

    sum2 = (sum[n:])

    temp = int(sum1,16) + int(sum2,16)

    sum = hex(temp)

y = sum

comp = int("FFFF",16) - int(sum,16)

sum = hex(comp)

final = hex(int(x,16) + int(y,16))

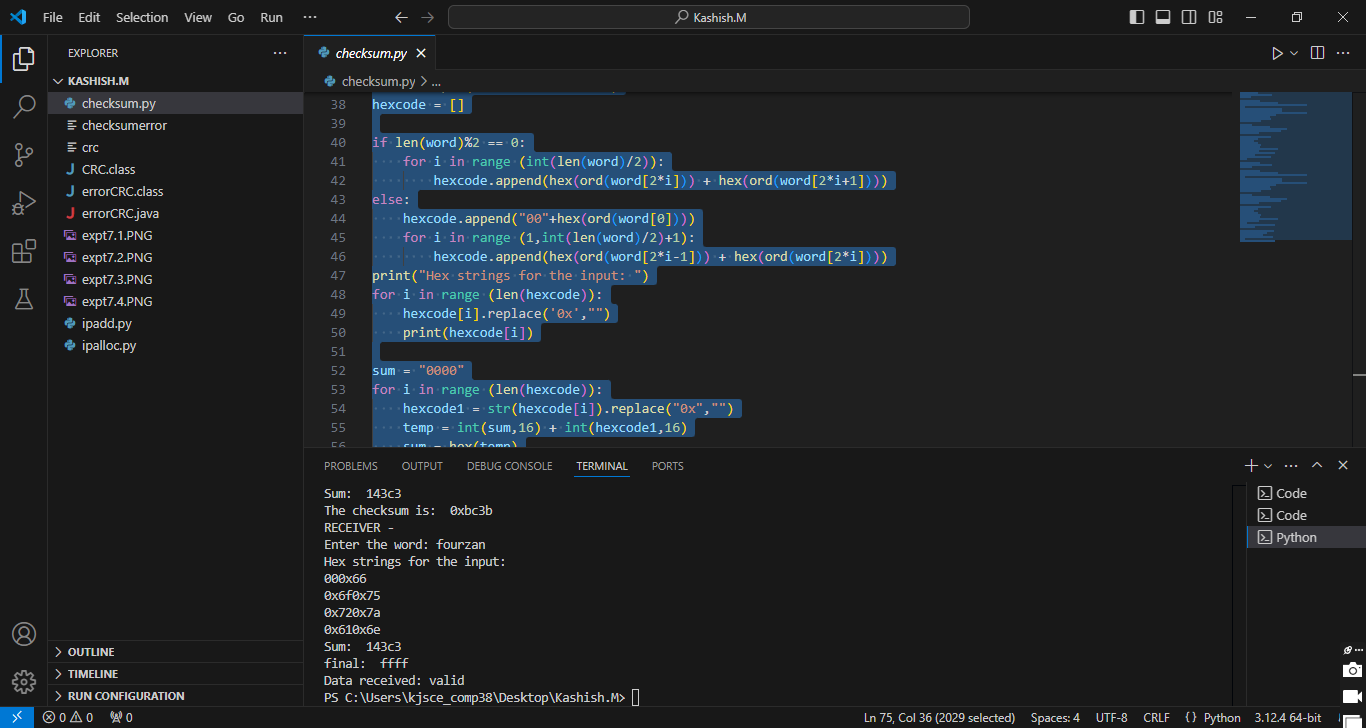
print("final: ", final[2:])

if final[2:] == 'ffff':

    print("Data received: valid")

else:

    print("Data received: invalid")



**CRC**

def xor(a, b):

    # XOR operation for two binary strings

    result = ""

    for i in range(1, len(b)):

        if a[i] == b[i]:

            result += "0"

        else:

            result += "1"

    return result

def crc(data, generator):

    # Append zeros to the data according to the length of the generator polynomial

    data = data + '0' \* (len(generator) - 1)

    # Extract the initial substring of the same length as the generator

    remainder = data[:len(generator)]

    for i in range(len(generator), len(data) + 1):

        # Perform XOR operation only if the leftmost bit is 1

        if remainder[0] == '1':

            remainder = xor(remainder, generator) + data[i:i + 1]

        else:

            remainder = remainder[1:] + data[i:i + 1]

    return remainder[:-1]  # Remove the last bit

def calculate\_syndrome(received\_data, generator):

    # Calculate the syndrome for the received data

    syndrome = crc(received\_data, generator)

    return syndrome

def main():

    # Input data in binary format

    data = input("Enter data in binary format: ")

    # Input the generator polynomial in binary format

    generator = input("Enter generator polynomial in binary format (e.g., 1101): ")

    # Calculate CRC at the sender side

    sender\_crc = crc(data, generator)

    print(f"CRC value to be appended to the data: {sender\_crc}")

    # Append the CRC to the data for transmission

    transmitted\_data = data + sender\_crc

    print(f"Transmitted data (Data + CRC): {transmitted\_data}")

    # At the receiver side, calculate the syndrome

    received\_data = transmitted\_data  # Simulate the received data

    syndrome = calculate\_syndrome(received\_data, generator)

    print(f"Syndrome (CRC check at receiver side): {syndrome}")

    # If syndrome is all zeros, data is correct

    if int(syndrome) == 0:

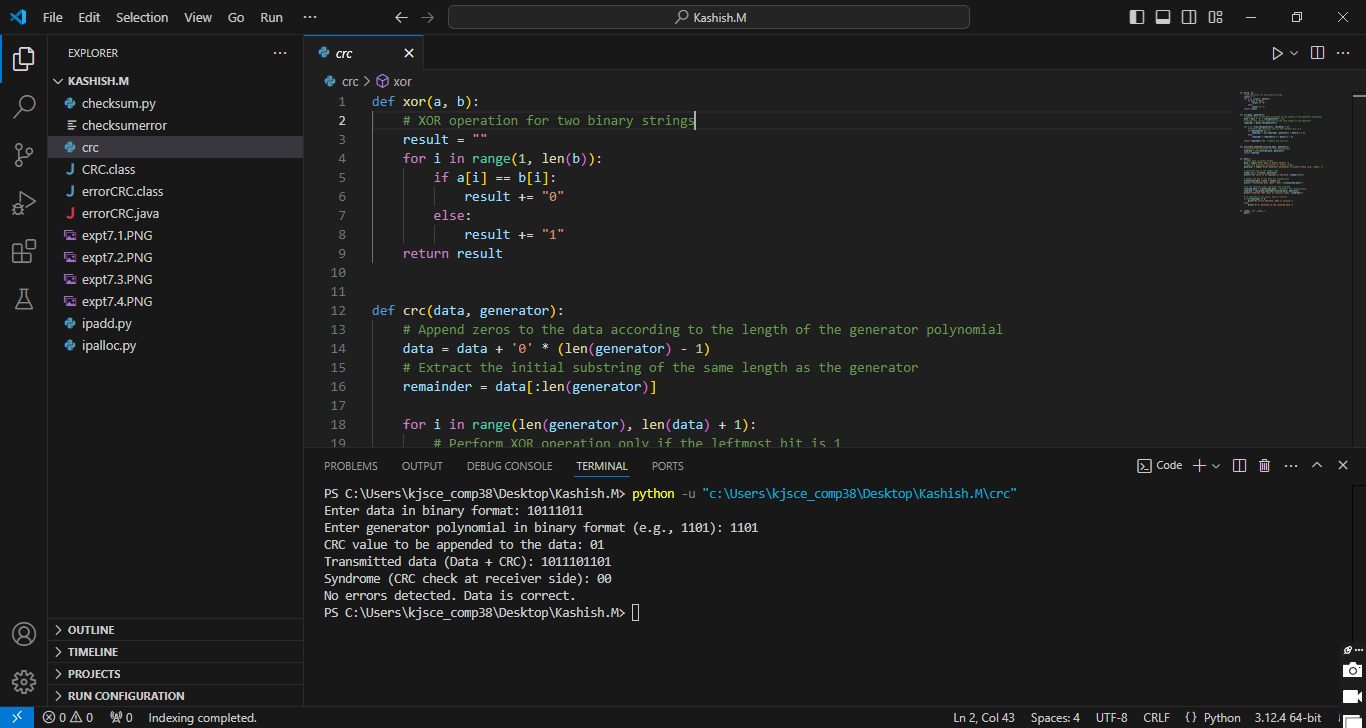
        print("No errors detected. Data is correct.")

    else:

        print("Error detected in the received data.")

if \_\_name\_\_ == "\_\_main\_\_":

    main()



**CONCLUSION:**

We implemented crc and checksum error detection methods.

**Post Lab Questions**

1. Discuss about the rules for choosing a CRC generator.

Ans:

 **Polynomial Should Be Irreducible**: The generator polynomial should not be factorable to ensure high error detection capability.

 **Highest Order Coefficient is 1**: This ensures that the polynomial covers all data bits in the frame.

 **Detect Common Errors**: The polynomial should detect common types of errors, such as single-bit errors, burst errors, and double-bit errors.

 **Length of Polynomial**: Choose a generator polynomial that is suitable for the message size. Longer polynomials can detect larger burst errors.

 **Efficient Calculation**: The generator should allow for fast and efficient CRC calculation during data transmission.

1. State the advantages and disadvantages of Internet Checksum.

Ans:

1. **Advantages**:
   * **Simple Calculation**: Internet checksum uses basic arithmetic (1's complement addition), making it easy and quick to compute.
   * **Lightweight**: It is a simple and efficient method, requiring fewer computational resources compared to more complex error-detection algorithms like CRC.
   * **Good for Small Errors**: It can detect single-bit errors and some simple multi-bit errors effectively.

**Disadvantages**:

* + **Weak Error Detection**: It is not as robust as CRC, and may fail to detect some multi-bit errors, burst errors, or patterns of errors.
  + **Not Suitable for High-Integrity Applications**: Due to its limitations in detecting more complex error patterns, it’s not ideal for systems that require high data integrity (e.g., file transfers or critical communication protocols).
  + **Limited Error Coverage**: The detection capability of the Internet checksum is significantly weaker for certain error patterns, making it less reliable for large amounts of data.

**Date :21/10/2024 Signature of Faculty In-charge**