

**UNIVERSITY OF ENGINEERING & MANAGEMENT, KOLKATA**

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING (CSE)

SUBJECT NAME: DATA COMMUNICATION AND NETWORKING

SUBJECT CODE: CS 491

2ND YEAR 4TH SEMESTER

**PROJECT REPORT ON:**

“Design a checksum checker. Write a program in any programming language which will check multiple data following the checksum for both sender and receiver side.”

BY

* + Sagnik Bhattacharya

**ACKNOWLEDGEMENTS**

We are very much thankful to our loving Parents and Faculty for their care and responsibility in helping us to achieve this work done. We are also greatly indebted to UEMKOLKATA that has provided a healthy environment to drive us to achieve our ambitions and goals.

Place: UEM-KOLKATA

Date: 1st April, 2019

**TABLE OF CONTENTS**

|  |  |  |
| --- | --- | --- |
|  | **Content** | **Page No.** |
| 1. | Introduction & Objective |  |
| 2. | What is CHECKSUM? |  |
| 3. | Checksum Offloading |  |
| 4. | Error detection by Checksum & Example |  |
| 5. | Source Code |  |
| 6. | Output |  |
| 7. | Conclusion & References |  |

**INTRODUCTION**

When bits are transmitted over the computer network, they are subject to get corrupted due to interference and network problems. The corrupted bits lead to spurious data being received by the receiver and are called errors.

Error detection techniques are responsible for checking whether any error has occurred or not in the frame that has been transmitted via network. It does not consider the number of error bits and the type of error.

For error detection, the sender needs to send some additional bits along with the data bits. The receiver performs necessary checks based upon the additional redundant bits. If it finds that the data is free from errors, it removes the redundant bits before passing the message to the upper layers.

**OBJECTIVE**

Our objective is to design a checksum checker. We’ve to write a code which will represent checksum checker.

We’ve used Python 3 and C for this Project.

**WHAT IS CHECKSUM?**

Checksums are used to ensure the integrity of data portions for data transmission or storage. A checksum is basically a calculated summary of such a data portion.

Network data transmissions often produce errors, such as toggled, missing or duplicated bits. Thus, the data received might not be identical to the data transmitted, which is obviously a bad thing.

Because of these transmission errors, network protocols very often use checksums to detect such errors. The transmitter will calculate a checksum of the data and transmits the data together with the checksum. The receiver will calculate the checksum of the received data with the same algorithm as the transmitter. If the received and calculated checksums don’t match a transmission error has occurred.

Some checksum algorithms can recover (simple) errors by calculating where the expected error must be and repairing it.

If there are errors that cannot be recovered, the receiving side throws away the packet. Depending on the network protocol, this data loss is simply ignored or the sending side needs to detect this loss somehow and retransmits the required packet(s).

Using a checksum drastically reduces the number of undetected transmission errors. However, the usual checksum algorithms cannot guarantee an error detection of 100%, so a very small number of transmission errors may remain undetected.

There are several different kinds of checksum algorithms; an example of an often-used checksum algorithm is CRC32. The checksum algorithm chosen for a specific network protocol will depend on the expected error rate of the network medium, the importance of error detection, the processor load to perform the calculation, the performance needed and many other things.

**CHECKSUM OFFLOADING**

The checksum calculation might be done by the network driver, protocol driver or even in hardware.

For example: The Ethernet transmitting hardware calculates the Ethernet CRC32 checksum and the receiving hardware validates this checksum. If the received checksum is wrong Wireshark won’t even see the packet, as the Ethernet hardware internally throws away the packet.

Higher level checksums are “traditionally” calculated by the protocol implementation and the completed packet is then handed over to the hardware.

Recent network hardware can perform advanced features such as IP checksum calculation, also known as checksum offloading. The network driver won’t calculate the checksum itself but will simply hand over an empty (zero or garbage filled) checksum field to the hardware.

|  |  |
| --- | --- |
|  | **Note** |
| Checksum offloading often causes confusion as the network packets to be transmitted are handed over to Wireshark before the checksums are calculated. Wireshark gets these “empty” checksums and displays them as invalid, even though the packets will contain valid checksums when they leave the network hardware later. |

Checksum offloading can be confusing and having a lot of [invalid] messages on the screen can be quite annoying. As mentioned above, invalid checksums may lead to un-reassembled packets, making the analysis of the packet data much harder.

You can do two things to avoid this checksum offloading problem:

* Turn off the checksum offloading in the network driver, if this option is available.
* Turn off checksum validation of the specific protocol in the Wireshark preferences. Recent releases of Wireshark disable checksum validation by default due to the prevalence of offloading in modern hardware and operating systems.

**ERROR DETECTION BY CHECKSUM**

For error detection by checksums, data is divided into fixed sized frames or segments.

* **Sender’s End** − The sender adds the segments using 1’s complement arithmetic to get the sum. It then complements the sum to get the checksum and sends it along with the data frames.
* **Receiver’s End** − The receiver adds the incoming segments along with the checksum using 1’s complement arithmetic to get the sum and then complements it.

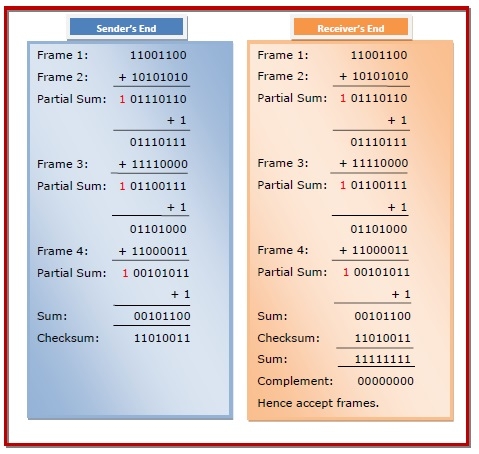
If the result is zero, the received frames are accepted; otherwise they are discarded.

**EXAMPLE:**

Suppose that the sender wants to send 4 frames each of 8 bits, where the frames are 11001100, 10101010, 11110000 and 11000011.

The sender adds the bits using 1s complement arithmetic. While adding two numbers using 1s complement arithmetic, if there is a carry over, it is added to the sum.

After adding all the 4 frames, the sender complements the sum to get the checksum, 11010011, and sends it along with the data frames.

The receiver performs 1s complement arithmetic sum of all the frames including the checksum. The result is complemented and found to be 0. Hence, the receiver assumes that no error has occurred.

**Source code: Python**

print("\n\n----- >>>>> Checksum Generator & Checker <<<<< -----\n\n")

n = int(input("Number of Data : "))

bits = int(input("Number of Bits per Data : "))

data = []

data\_y = []

sum\_x = '0000'

sum\_y = '0000'

# Generating Checksum

print("--- >>> Checksum Generator <<< ---")

print(' ')

# Calculating the Sum

for i in range(n):

data.append(input("Enter Data No." + str(i+1) + " : ")[:bits])

sum\_x = bin(int(data[i], 2) + int(sum\_x, 2))[2:]

if len(sum\_x) > bits:

sum\_x = bin(int('1', 2) + int(sum\_x, 2))[3:]

# 1's Compliment of the Sum

for i in range(bits):

if sum\_x[i:i+1] == '0':

sum\_x = sum\_x[:i] + '1' + sum\_x[i+1:]

else:

sum\_x = sum\_x[:i] + '0' + sum\_x[i+1:]

print('\nChecksum : ' + sum\_x)

# Checksum checker

print("\n\n\n--- >>> Checksum Checker <<< ---\n")

# Calculating the Sum

for i in range(n):

data\_y.append(input("Enter Data No." + str(i+1) + " : ")[:bits])

sum\_y = bin(int(data\_y[i], 2) + int(sum\_y, 2))[2:]

if len(sum\_y) > bits:

sum\_y = bin(int('1', 2) + int(sum\_y, 2))[3:]

# Checking Data Validity

result = bin(int(sum\_x, 2) + int(sum\_y, 2))[2:]

check = '1'\*bits

if result == check:

print("\nData is Valid")

else:

print("\nData is corrupted")

**Source code: C**

#include<stdio.h>

#include<string.h>

int main()

{

char a[20],b[20];

char sum[20],complement[20];

int i,length;

printf("Enter first binary string\n");

scanf("%s",&a);

printf("Enter second binary string\n");

scanf("%s",&b);

if(strlen(a)==strlen(b)){

length = strlen(a);

char carry='0';

for(i=length-1;i>=0;i--)

{

if(a[i]=='0' && b[i]=='0' && carry=='0')

{

sum[i]='0';

carry='0';

}

else if(a[i]=='0' && b[i]=='0' && carry=='1')

{

sum[i]='1';

carry='0';

}

else if(a[i]=='0' && b[i]=='1' && carry=='0')

{

sum[i]='1';

carry='0';

}

else if(a[i]=='0' && b[i]=='1' && carry=='1')

{

sum[i]='0';

carry='1';

}

else if(a[i]=='1' && b[i]=='0' && carry=='0')

{

sum[i]='1';

carry='0';

}

else if(a[i]=='1' && b[i]=='0' && carry=='1')

{

sum[i]='0';

carry='1';

}

else if(a[i]=='1' && b[i]=='1' && carry=='0')

{

sum[i]='0';

carry='1';

}

else if(a[i]=='1' && b[i]=='1' && carry=='1')

{

sum[i]='1';

carry='1';

}

else

break;

}

printf("\nSum=%c%s",carry,sum);

for(i=0;i<length;i++)

{

if(sum[i]=='0')

complement[i]='1';

else

complement[i]='0';

}

if(carry=='1')

carry='0';

else

carry='1';

printf("\nChecksum=%c%s",carry,complement);

}

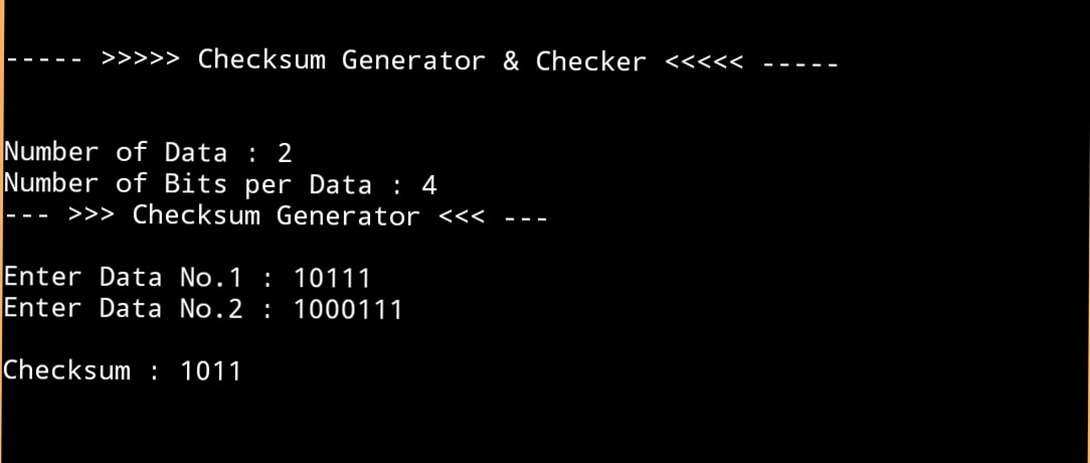
else {

printf("\nWrong input strings");

}

}

**OUTPUT**

****

**CONCLUSION**

From this given project, we learnt how to build a checksum generator and its functions. We learnt Checksums are used to ensure the integrity of data portions for data transmission or storage. Network data transmissions often produce errors, such as toggled, missing or duplicated bits. Thus, the data received might not be identical to the data transmitted, which is obviously a bad thing.

Because of these transmission errors, network protocols very often use checksums to detect such errors.

**REFERANCES**

* [www.wikipedia.com](http://www.wikipedia.com)
* [www.geekofgeeks.com](http://www.geekofgeeks.com)
* [www.techopedia.com](http://www.techopedia.com)