

## CheckSum

A checksum is a small-sized datum derived from a block of digital data for the purpose of detecting errors that may have been introduced during its transmission or storage. It is usually applied to an installation file after it is received from the download server. By themselves, checksums are often used to verify data integrity but are not relied upon to verify data authenticity.

The actual procedure which yields the checksum from a data input is called a checksum function or checksum algorithm. Depending on its design goals, a good checksum algorithm will usually output a significantly different value, even for small changes made to the input. This is especially true of cryptographic hash functions, which may be used to detect many data corruption errors and verify overall data integrity; if the computed checksum for the current data input matches the stored value of a previously computed checksum, there is a very high probability the data has not been accidentally altered or corrupted.

Checksum functions are related to hash functions, fingerprints, randomization functions, and cryptographic hash functions. However, each of those concepts has different applications and therefore different design goals. For instance, a function returning the start of a string can provide a hash appropriate for some applications but will never be a suitable checksum. Checksums are used as cryptographic primitives in larger authentication algorithms. For cryptographic systems with these two specific design goals, see HMAC.

Check digits and parity bits are special cases of checksums, appropriate for small blocks of data (such as Social Security numbers, bank account numbers, computer words, single bytes, etc.).

Some error-correcting codes are based on special checksums which not only detect common errors but also allow the original data to be recovered in certain cases.

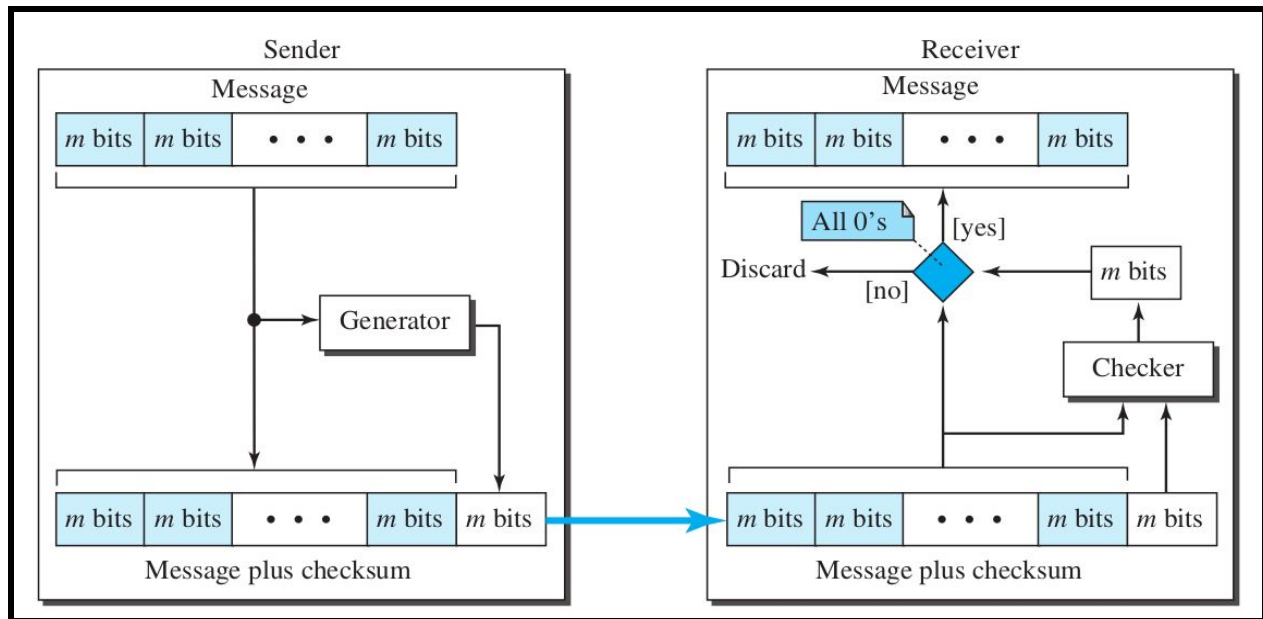


Fig 1. Checksum Flow Diagram

## Algorithms for Calculating CheckSum

### Parity Byte or Parity Word Based Checksum

The simplest checksum algorithm is the so-called longitudinal parity check, which breaks the data into "words" with a fixed number  $n$  of bits, and then computes the exclusive or (XOR) of all those words. The result is appended to the message as an extra word. To check the integrity of a message, the receiver computes the exclusive or of all its words, including the checksum; if the result is not a word consisting of  $n$  zeros, the receiver knows a transmission error occurred.

With this checksum, any transmission error which flips a single bit of the message, or an odd number of bits, will be detected as an incorrect checksum. However, an error which affects two bits will not be detected if those bits lie at the same position in two distinct words. Also swapping of two or more words will not be detected. If the affected bits are independently chosen at random, the probability of a two-bit error being undetected is  $1/n$ .

### Sum Complement Based Checksum

A variant of the previous algorithm is to add all the "words" as unsigned binary numbers, discarding any overflow bits, and append the two's complement of the total as the checksum. To validate a message, the

receiver adds all the words in the same manner, including the checksum; if the result is not a word full of zeros, an error must have occurred. This variant too detects any single-bit error, but the pro modular sum is used in SAE J1708.

## Position Dependent Checksum

The simple checksums described above fail to detect some common errors which affect many bits at once, such as changing the order of data words, or inserting or deleting words with all bits set to zero. The checksum algorithms most used in practice, such as Fletcher's checksum, Adler-32, and cyclic redundancy checks (CRCs), address these weaknesses by considering not only the value of each word but also its position in the sequence. This feature generally increases the cost of computing the checksum.

## Types of Checksum Based on Number of Bits

### 8-bit Checksum

The 8-bit checksum is the 2's complement of the sum of all bytes. The checksum value, when added to the sum of all bytes produces a result of zero. Bytes are provided as two-character strings.

### 16-bit Checksum

Traditionally, the Internet has used a 16-bit checksum. The checksum value, when added to the sum of all bytes produces a result of zero. Bytes are provided as four-character strings. It is also known as Internet Checksum.

### 32-bit Checksum

The 32-bit checksum is the 2's complement of the sum of all bytes. The checksum value, when added to the sum of all bytes produces a result of zero. Bytes are provided as eight-character strings.

## Algorithm to Calculate Checksum

The flow diagram given below can be used as an algorithm for the calculation of the checksum. A program in any language can easily be written based on the algorithm. Note that the first loop just calculates the sum of the data units in two's complement; the second loop wraps the extra bits created from the two's complement calculation to simulate the calculations in one's complement. This is needed because almost all computers today do calculation in two's complement.

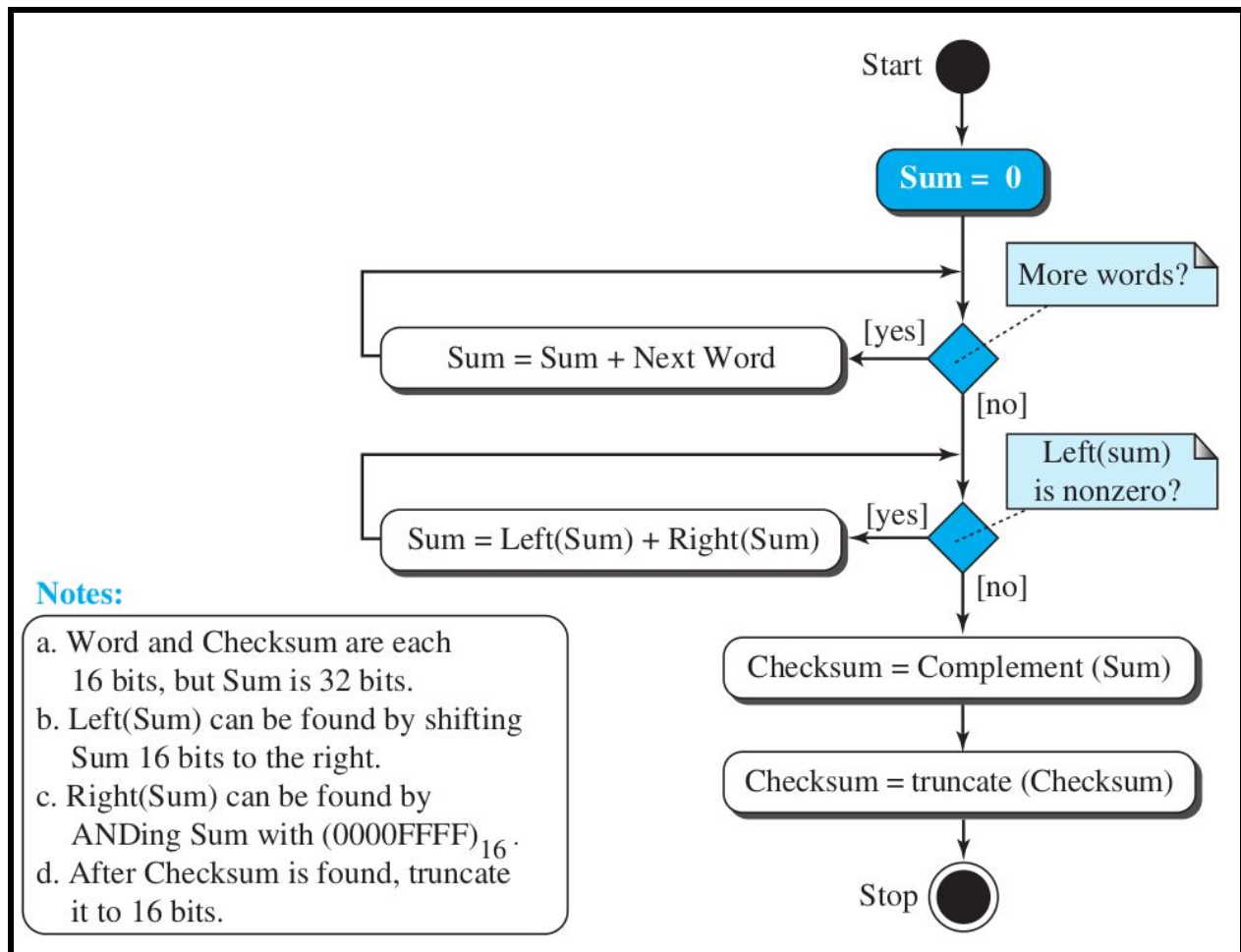


Fig 2. Checksum Calculation Algorithm

## Other Approaches to Checksum

As mentioned before, there is one major problem with the traditional checksum calculation. If two 16-bit items are transposed in transmission, the checksum cannot catch this error. The reason is that the traditional checksum is not weighted: it treats each data item equally. In other words, the order of data

items is immaterial to the calculation. Several approaches have been used to prevent this problem. Two of them here: Fletcher and Adler.

## Fletcher Checksum

The Fletcher checksum was devised to weight each data item according to its position. Fletcher has proposed two algorithms: 8-bit and 16-bit. The first, 8-bit Fletcher, calculates on 8-bit data items and creates a 16-bit checksum. The second, 16-bit Fletcher, calculates on 16-bit data items and creates a 32-bit checksum.

The 8-bit Fletcher is calculated over data octets (bytes) and creates a 16-bit checksum. The calculation is done modulo 256 ( $2^8$ ), which means the intermediate results are divided by 256 and the remainder is kept. The algorithm uses two accumulators, L and R. The first simply add data items together; the second adds weight to the calculation. There are many variations of the 8-bit Fletcher algorithm. The 16-bit Fletcher checksum is similar to the 8-bit Fletcher checksum, but it is calculated over 16-bit data items and creates a 32-bit checksum. The calculation is done modulo 65,536.

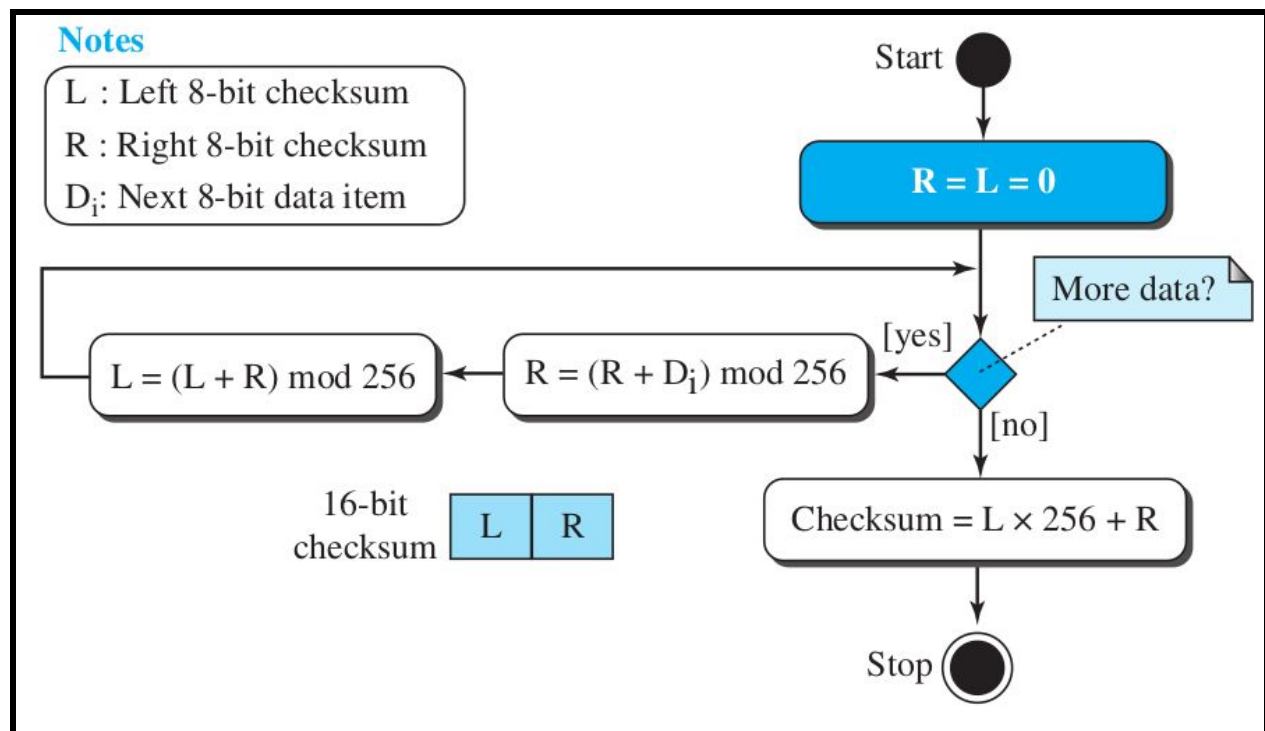


Fig 3. Algorithm to calculate an 8-bit Fletcher checksum

## Adler Checksum

The Adler checksum is a 32-bit checksum. It is similar to the 16-bit Fletcher with three differences. First, the calculation is done on single bytes instead of 2 bytes at a time. Second, the modulus is a prime number

(65,521) instead of 65,536. Third, L is initialized to 1 instead of 0. It has been proved that a prime modulo has a better detecting capability in some combinations of data.

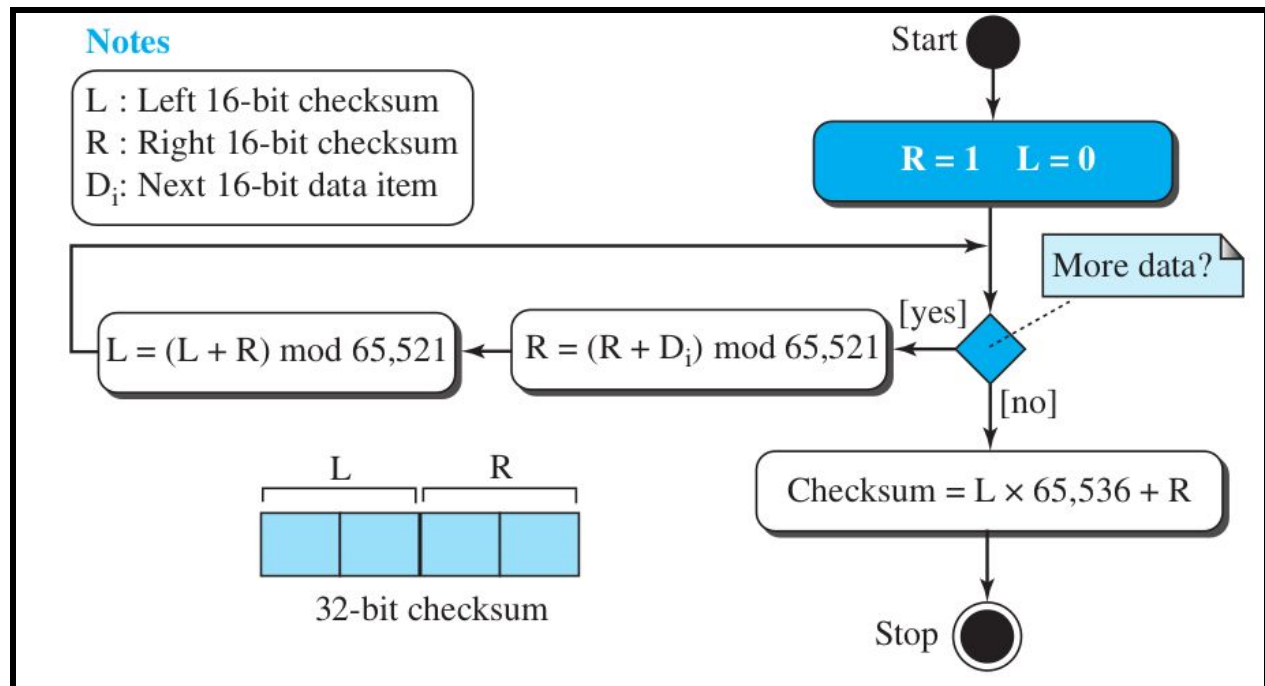


Fig 4. Algorithm to calculate an Adler Checksum

## Checksum Implementation

The checksum algorithm was implemented in MATLAB for calculating 8-bit, 16-bit and 32-bit checksums.

### Source Code

---

```

% CheckSum Implementation for 8,16 and 32 bit numbers.
optckLength = input("8-bit,16-bit or 32 bit checksum? ");

n = input("Enter total number of hex codes: ");

datawords = strings([1,n]);
binDatawords = strings([1,n]);
  
```

```

fprintf("Enter hex codes: \n");
for i=1:n
    datawords(i) = input("Enter code: ','s');
    binDatawords(i) = dec2bin(hex2dec(datawords(i)));
end

% optckLength = 0;
% % dWordLength = strlength(datawords(1));
% if(dWordLength == 1 || dWordLength == 2)
%     optckLength = 8;
% elseif(dWordLength ==3 || dWordLength == 4)
%     optckLength = 16;
% elseif(dWordLength>=5 && dWordLength<=8)
%     optckLength = 32;
% end

sentChecksum = checksumFunc(binDatawords,n,optckLength);
fprintf("Sent Checksum = %s\n",sentChecksum);

fprintf("*****AT RECEIVER'S END: *****\n");
fprintf("Received Data:\n");
receivedDatawords = [datawords, sentChecksum];
binreceivedDatawords = [binDatawords, dec2bin(hex2dec(sentChecksum))];

for i=1:n+1
    fprintf("%s\n",receivedDatawords(i));
end

receivedChecksum = checksumFunc(binreceivedDatawords, n+1, optckLength)
fprintf("Received Checksum = %s\n", receivedChecksum);

if(hex2dec(receivedChecksum)==0)
    fprintf("Data Successfully Transmitted\n");
else
    fprintf("Data corrupted\n");
end

function checksumHex = checksumFunc(binDatawords,n,optckLength)
    checksum = "0";
    for i=1:n

```

```

        checksum = dec2bin(bin2dec(checksum)+bin2dec(binDataawords(i)));
    end

    cksumLength = strlen(checksum);
    if(cksumLength < optckLength)
        checksum = [repmat('0', 1, optckLength-cksumLength), checksum];
    elseif(cksumLength > optckLength)
        tempLength = cksumLength - optckLength;
        temp = extractBetween(checksum, 1,tempLength);
        checksum = extractBetween(checksum, tempLength+1,cksumLength);
        checksum = dec2bin(bin2dec(checksum) + bin2dec(temp));
    end

    for i=1:optckLength
        if(checksum(i)=='1')
            checksum(i)='0';
        else
            checksum(i)='1';
        end
    end

    checksumHex = dec2hex(bin2dec(checksum));
end

```

---

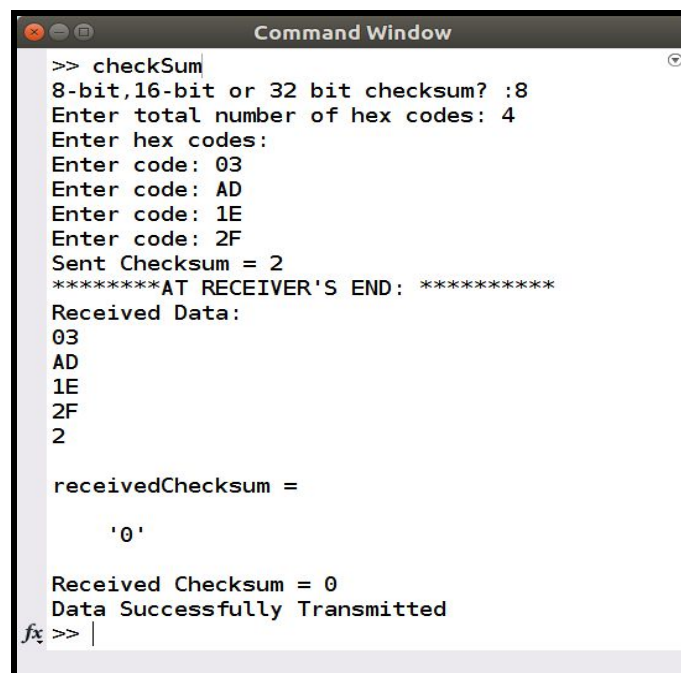


## Observations and Results

**Input-** The input to checksum program is the type of checksum required(8-bit, 16-bit or 32-bit) and the list of data words.

**Output-** The program generates checksum and then on the receiver's side, whether the correct message has been passed or not.

### 8-bit Checksum



```
>> checksum
8-bit,16-bit or 32 bit checksum? :8
Enter total number of hex codes: 4
Enter hex codes:
Enter code: 03
Enter code: AD
Enter code: 1E
Enter code: 2F
Sent Checksum = 2
*****AT RECEIVER'S END: *****
Received Data:
03
AD
1E
2F
2

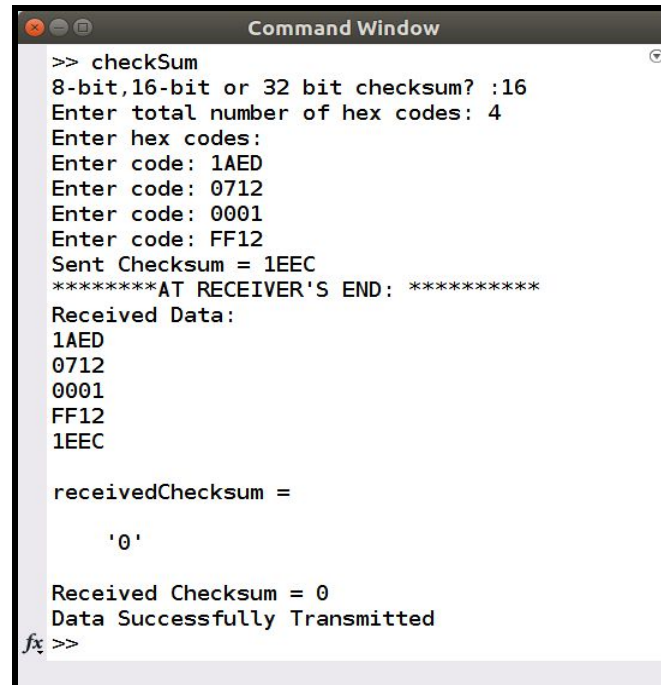
receivedChecksum =

'0'

Received Checksum = 0
Data Successfully Transmitted
fx >> |
```

Fig 5. 8-bit Checksum Output

## 16-bit Checksum



```
>> checksum
8-bit,16-bit or 32 bit checksum? :16
Enter total number of hex codes: 4
Enter hex codes:
Enter code: 1AED
Enter code: 0712
Enter code: 0001
Enter code: FF12
Sent Checksum = 1EEC
*****AT RECEIVER'S END: *****
Received Data:
1AED
0712
0001
FF12
1EEC

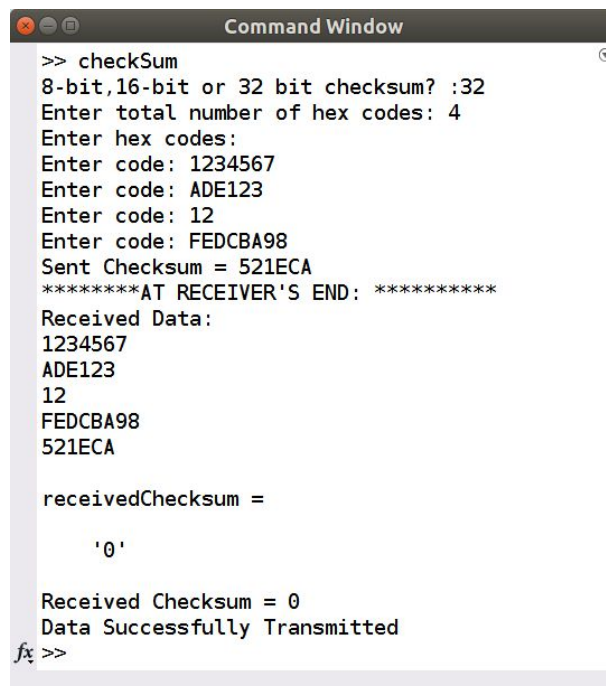
receivedChecksum =

'0'

Received Checksum = 0
Data Successfully Transmitted
fx >>
```

Fig 6. 16-bit Checksum Output

## 32-bit Checksum



```
>> checksum
8-bit,16-bit or 32 bit checksum? :32
Enter total number of hex codes: 4
Enter hex codes:
Enter code: 1234567
Enter code: ADE123
Enter code: 12
Enter code: FEDCBA98
Sent Checksum = 521ECA
*****AT RECEIVER'S END: *****
Received Data:
1234567
ADE123
12
FEDCBA98
521ECA

receivedChecksum =

'0'

Received Checksum = 0
Data Successfully Transmitted
fx >>
```

Fig 7. 32-bit Checksum Output

## Appendix

MATLAB programming language was used in this project, and its inbuilt programming functions were used to implement this project. Some of the functions used are-

1. **dec2bin** - Convert decimal number to character array representing the binary number.
2. **hex2dec** - Convert text representation of the hexadecimal number to decimal number.
3. **bin2dec** - Convert text representation of the binary number to decimal number.
4. **strlength** - This function returns the number of characters in the string.
5. **repmat** - Repeat copies of the array as specified in the format
6. **dec2hex**-.Convert decimal number to a hexadecimal text representation.
7. **extractBetween** - Extract substrings between start and endpoints.

## References

1. Behrouz A. Forouzan (2013), Data Communications and Networking, 5th Edition, McGraw-Hill Publications.
2. Checksum- Wikipedia [<https://en.wikipedia.org/wiki/Checksum>], wikipedia.org, Retrieved on 26th October 2019.
3. MATLAB Documentation [<https://in.mathworks.com/help/matlab/ref/>] Software Version-R2019a