CSE1004 - Network and Communication -Embedded Lab[DA-2]

Name:VIKASH KUMAR Reg no.:19BCE0084 Slot: L59+L60

1.EVEN AND ODD PARITY GENERATOR

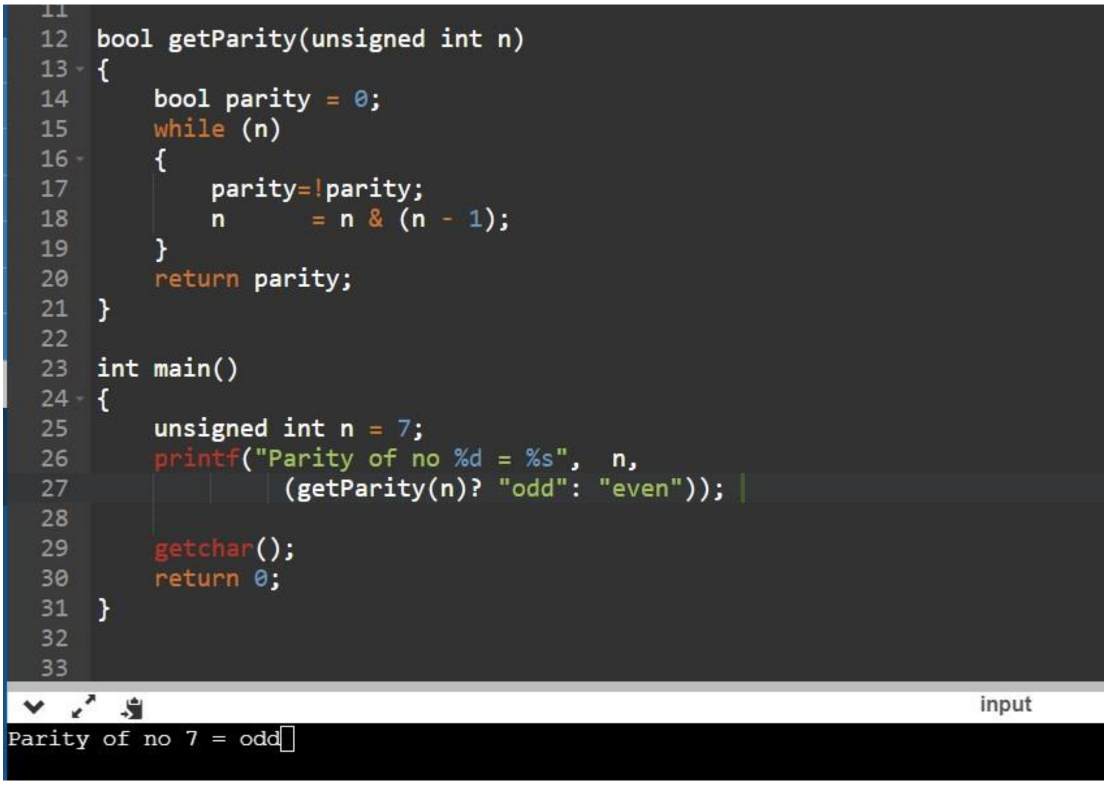
ALGORITH/PESUDOCODE

```
1. Initialize parity = 0 2.
Loop while n != 0
a.   Invert parity
parity = !parity
b.   Unset rightmost set bit
n = n & (n-1)
3. return parity
```

SOURCE CODE

```
#  include  <stdio.h>   #  define

bool    int                 bool

getParity(unsigned int n)

{      bool parity = 0;

while (n)      {

parity=!parity;          n

= n & (n - 1);

    }

return parity;

}     int

main()

{      unsigned int n = 7;

printf("Parity of no %d = %s",  n,

(getParity(n)? "odd": "even"));


getchar();

return 0;
}
```

OUTPUT



```
bool getParity(unsigned int n)
{
    bool parity = 0;
    while (n)
    {
        parity=!parity;
        n      = n & (n - 1);
    }
    return parity;
}

int main()
{
    unsigned int n = 7;
    printf("Parity of no %d = %s",  n,
            (getParity(n)? "odd": "even"));

    getchar();
    return 0;
}
```

```
Parity of no 7 = odd
```

2.CHECKSUM

ALGORITH/PESUDOCODE

1. Take 2 binary input strings.

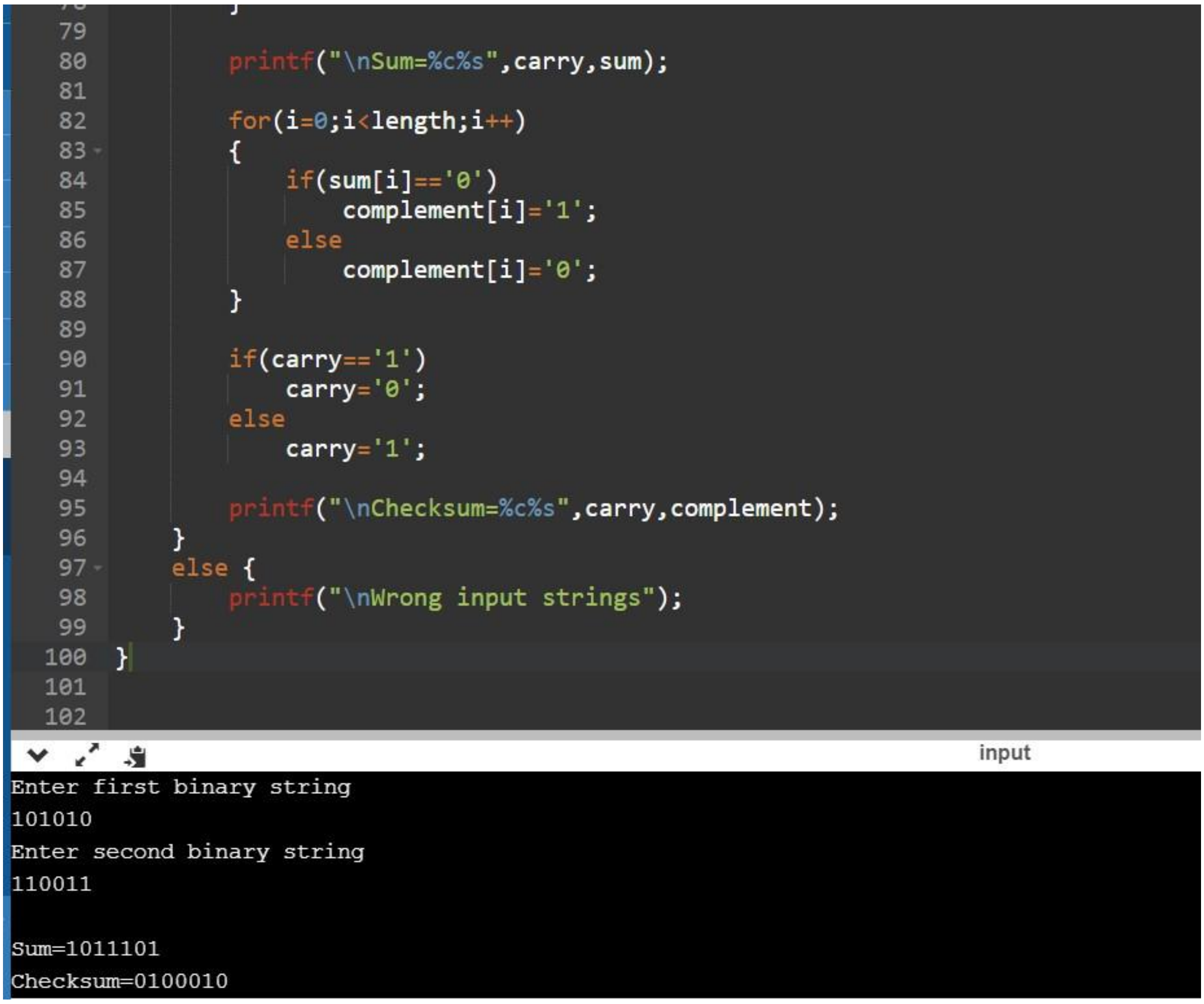
2. Do their binary sum to find out the checksum which will be sent to the destination or to the receiver.
3. In binary sum there are 6 cases:-
    a. If both bits are 0 and carry is 0, sum=0 and carry=0
    b. If both bits are 0 and carry is 1,sum=1 and carry=0
    c. If both bits are 1 and carry is 0,sum=0 and carry=1
    d. If both bits are 1 and carry is 1,sum=1 and carry=1
    e. If either bit is 1 and carry is 0,sum=1 and carry=0
    f. If either bit is 1 and carry is 1,sum=0 and carry=1
4. While doing the addition we have to add the binary strings from rightmost end i.e LSB to MSB.
5. When binary sum is done 1’s complement of it is taken by reversing 1’s to 0’s and vice versa.
6. The resulting 1’s complement is the Checksum.
7. Stop.

SOURCE CODE

```
#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



```c
        }

        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");
    }
}


```

```
Enter first binary string
101010
Enter second binary string
110011

Sum=1011101
Checksum=0100010
```

3.CYCLIC REDUNDANCY CHECK

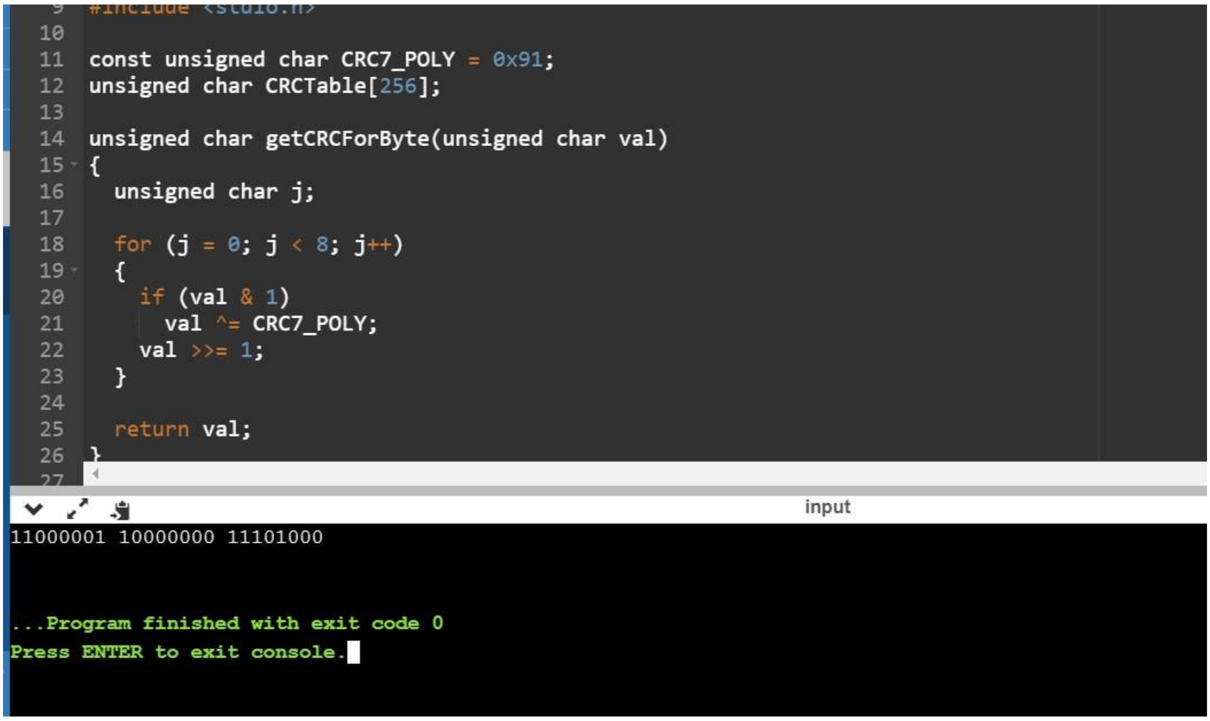
ALGORITH/PESUDOCODE

- The communicating parties agrees upon the size of message,$M(x)$ and the generator polynomial, $G(x)$.
- If r is the order of $G(x)$,r, bits are appended to the low order end of $M(x)$. This makes the block size bits, the value of which is $x^rM(x)$.
- The block $x^rM(x)$ is divided by $G(x)$ using modulo 2 division.
- The remainder after division is added to $x^rM(x)$ using modulo 2 addition. The result is the frame to be transmitted, $T(x)$. The encoding procedure makes exactly divisible by $G(x)$.

SOURCE CODE

```
#include <stdio.h>

const unsigned char CRC7_POLY = 0x91; unsigned
char CRCTable[256];
  unsigned char getCRCForByte(unsigned char
val)
{
  unsigned char j;
    for (j = 0; j < 8;
j++)
  {
    if (val & 1)
val ^= CRC7_POLY;
val >>= 1;
  }
return val;
}   void
buildCRCTable()
{   int
i;

  // fill an array with CRC values of all 256 possible bytes
for (i = 0; i < 256; i++)
  {
    CRCTable[i] = getCRCForByte(i);
  }
}   unsigned char getCRC(unsigned char message[], unsigned char
length)
{
  unsigned char i, crc = 0;
for (i = 0; i < length; i++)
crc = CRCTable[crc ^
message[i]];   return crc;
}   int
main() {
  unsigned char message[3] = {0x83, 0x01, 0x00};
int i, j;

  buildCRCTable();
  message[2] = getCRC(message, 2);
    for (i = 0; i < sizeof(message);
i++)
  {
    for (j = 0; j < 8; j++)
      printf("%d", (message[i] >> j) % 2);
printf(" ");
  }
  printf("\n");
}
```

OUTPUT



```
#include <stdio.h>

const unsigned char CRC7_POLY = 0x91;
unsigned char CRCTable[256];

unsigned char getCRCForByte(unsigned char val)
{
  unsigned char j;

  for (j = 0; j < 8; j++)
  {
    if (val & 1)
      val ^= CRC7_POLY;
    val >>= 1;
  }

  return val;
}
```

```
11000001 10000000 11101000


...Program finished with exit code 0
Press ENTER to exit console.
```

4.HAMMING CODE

ALGORITH/PESUDOCODE

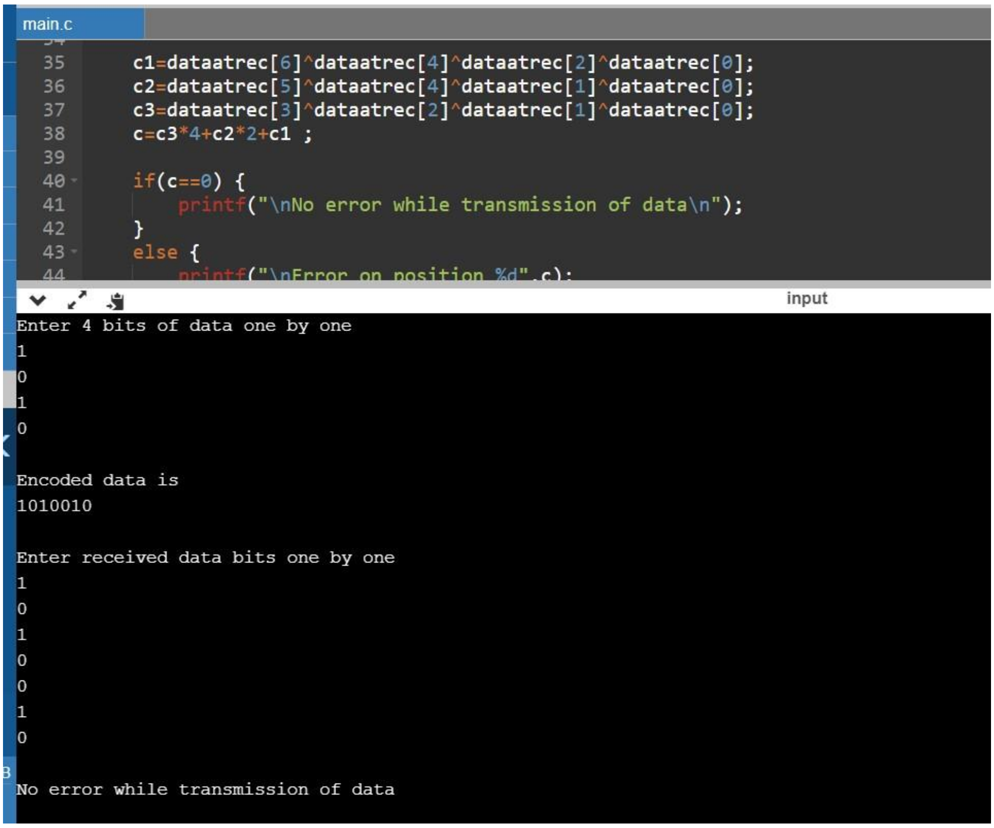
1. Write the bit positions starting from 1 in binary form (1, 10, 11, 100, etc).

2. All the bit positions that are a power of 2 are marked as parity bits (1, 2, 4, 8, etc).
3. All the other bit positions are marked as data bits.
4. Each data bit is included in a unique set of parity bits, as determined its bit position in binary form.
   a. Parity bit 1 covers all the bits positions whose binary representation includes a 1 in the least significant position (1, 3, 5, 7, 9, 11, etc).
   b. Parity bit 2 covers all the bits positions whose binary representation includes a 1 in the second position from the least significant bit (2, 3, 6, 7, 10, 11, etc).
   c. Parity bit 4 covers all the bits positions whose binary representation includes a 1 in the third position from the least significant bit (4–7, 12–15, 20–23, etc).
   d. Parity bit 8 covers all the bits positions whose binary representation includes a 1 in the fourth position from the least significant bit bits (8–15, 24–31, 40–47, etc).
   e. In general each parity bit covers all bits where the bitwise AND of the parity position and the bit position is non-zero.
5. Since we check for even parity set a parity bit to 1 if the total number of ones in the positions it checks is odd.
6. Set a parity bit to 0 if the total number of ones in the positions it checks is even.

SOURCE CODE

```
#include<iostream>
  using namespace
std;
  int main() {
int data[10];
    int dataatrec[10],c,c1,c2,c3,i;
      cout<<"Enter 4 bits of data one by
one\n";     cin>>data[0];     cin>>data[1];
cin>>data[2];     cin>>data[4];

    //Calculation of even parity
    data[6]=data[0]^data[2]^data[4];
data[5]=data[0]^data[1]^data[4];
data[3]=data[0]^data[1]^data[2];


       cout<<"\nEncoded data is\n";
       for(i=0;i<7;i++)
        cout<<data[i];
       cout<<"\n\nEnter received data bits one by one\n";
for(i=0;i<7;i++)         cin>>dataatrec[i];
c1=dataatrec[6]^dataatrec[4]^dataatrec[2]^dataatrec[0];
c2=dataatrec[5]^dataatrec[4]^dataatrec[1]^dataatrec[0];
c3=dataatrec[3]^dataatrec[2]^dataatrec[1]^dataatrec[0];
       c=c3*4+c2*2+c1 ;
      if(c==0)
{
              cout<<"\nNo error while transmission of data\n";
    }
       else {
              cout<<"\nError on position "<<c;

              cout<<"\nData sent : ";
              for(i=0;i<7;i++)
              cout<<data[i];

              cout<<"\nData received : ";
        for(i=0;i<7;i++)
cout<<dataatrec[i];

              cout<<"\nCorrect message is\n";

              //if errorneous bit is 0 we complement it else vice versa
              if(dataatrec[7-c]==0)
                     dataatrec[7-c]=1;
        else
                     dataatrec[7-c]=0;
for (i=0;i<7;i++) {
                     cout<<dataatrec[i];
              }
       }

       return 0;
}
```

OUTPUT



```
    c1=dataatrec[6]^dataatrec[4]^dataatrec[2]^dataatrec[0];
    c2=dataatrec[5]^dataatrec[4]^dataatrec[1]^dataatrec[0];
    c3=dataatrec[3]^dataatrec[2]^dataatrec[1]^dataatrec[0];
    c=c3*4+c2*2+c1 ;

    if(c==0) {
        printf("\nNo error while transmission of data\n");
    }
    else {
        printf("\nError on position %d",c);
```

```
Enter 4 bits of data one by one
1
0
1
0

Encoded data is
1010010

Enter received data bits one by one
1
0
1
0
0
1
0

No error while transmission of data
```