PSG COLLEGE OF TECHNOLOGY

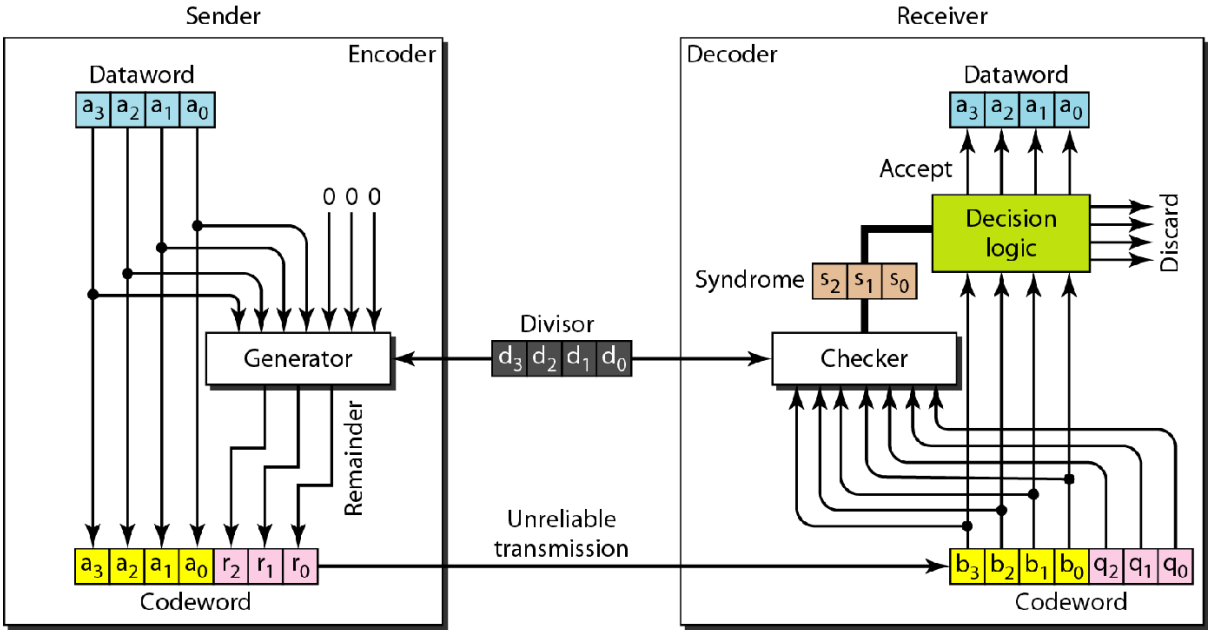
DEPARTMENT OF APPLIED MATHEMATICS AND COMPUTATIONAL SCIENCES

In this lab you will write a program which computes the CRC for a block of data. The cyclic redundancy check, or CRC, is a technique for detecting errors in digital data, but not for making corrections when errors are detected. It is used primarily in data transmission.

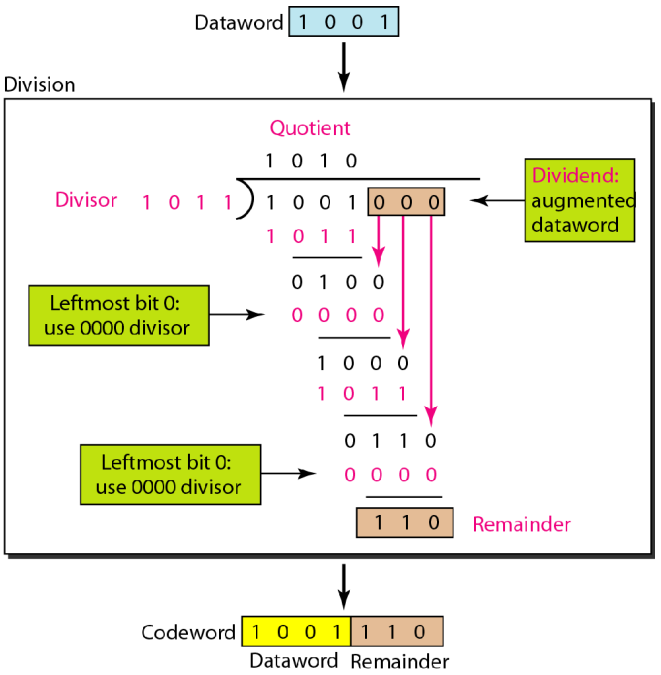
In the CRC method, a certain number of check bits, often called a checksum, are appended to the message being transmitted. The receiver can determine whether or not the check bits agree with the data, to ascertain with a certain degree of probability whether or not an error occurred in transmission. If an error occurred, the receiver sends a "negative acknowledgement" (NAK) back to the sender, requesting that the message be retransmitted.

The technique is also sometimes applied to data storage devices, such as a disk drive. In this situation each block on the disk would have check bits, and the hardware might automatically initiate a reread of the block when an error is detected, or it might report the error to software. The material that follows speaks in terms of a "sender" and a "receiver" of a "message," but it should be understood that it applies to storage writing and reading as well.

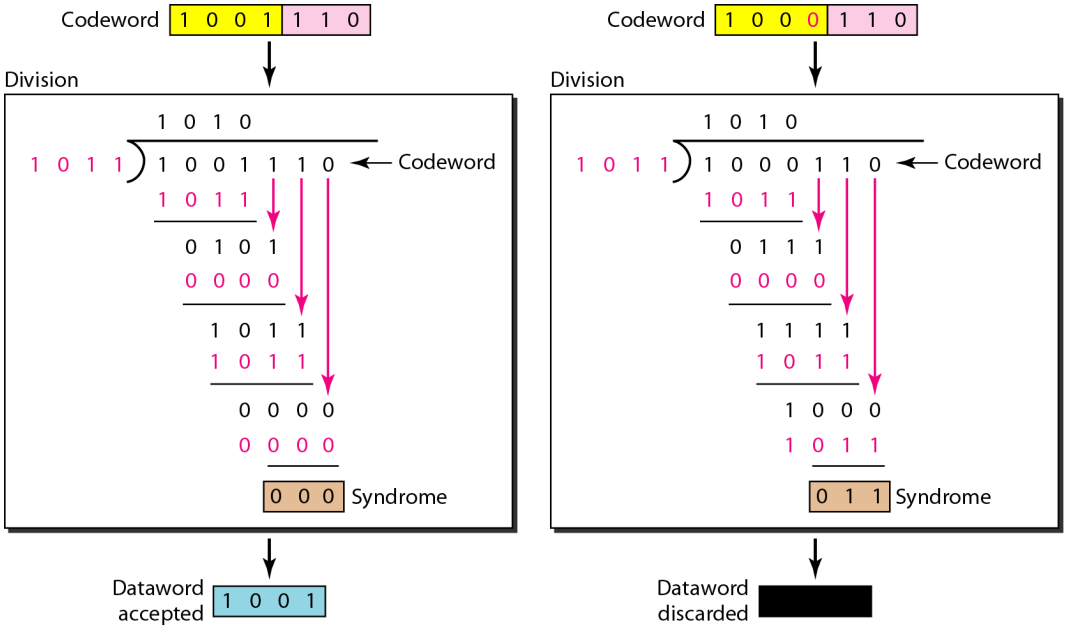
Following Diagram shows the working of CRC.



This image shows division at sender side.



Division at Receiver side.



Write programs to

- Read binary data from a file, compute the checksum and store the data along with checksum in another file.
- Read the file which contains data along with checksum and detect errors if any (Manually insert error in the file by modifying bit(s)).

CRC at sender's side:

```
def xor(a,b):
        result = []
        for i in range(1,len(b)):
        if a[i]==b[i]:
        result.append('0')
        else:
        result.append('1')
        return result

def crcDiv(data, key):
        ld = len(key)
        tempD = data[0:ld]

        while ld<len(data):

        print(tempD)
                if tempD[0] == '1':
        tempD = xor(key,tempD) + data[ld]
        else:
        tempD = xor(0*key, tempD) + data[ld]
        ld+=1

        if tempD[0] == '1':
        tempD = xor(key,tempD)
        else:
        tempD = xor(0*key, tempD)

def encodeData(data,key):
```

```
    codeword = []
    lKey=len(key)
    codeword = data + '0'*(lKey-1)
    print(codeword)
    crcDiv(codeword, key)

encodeData('1001','1011')
```

CRC at receiver's side:

```
def xor(a,b):
    result = []
    for i in range(1,len(b)):
    if a[i]==b[i]:
    result.append('0')
    else:
    result.append('1')
    return result

def crcDiv(data, key):
    ld = len(key)
    tempD = data[0:ld]

    while ld<len(data):
    if tempD[0] == '1':
    tempD = xor(key,tempD) + data[ld]
    else:
    tempD = xor(0*key, tempD) + data[ld]
    ld+=1
```

```
        if tempD[0] == '1':
        tempD = xor(key,tempD)
        else:
        tempD = xor(0*key, tempD)

        print(tempD)
        for i in tempD:
        if i=='1':
        print("Error")
        break

crcDiv('1001110','1011')
```