Labsheet 3

Name: Aniketh Vijesh

Roll No: AM.EN.U4AIE22009

In [1]:
```
def calculate_lrc(data_blocks):
    lrc = [0] * len(data_blocks[0])
    for block in data_blocks:
        for i in range(len(block)):
            lrc[i] ^= int(block[i])
    return ''.join(map(str, lrc))

def check_lrc(data_blocks, received_lrc):
    calculated_lrc = calculate_lrc(data_blocks)
    return calculated_lrc == received_lrc

data = ["1010", "1100", "1111"]
lrc = calculate_lrc(data)
transmitted_message = data + [lrc]
print("Original Message:", " ".join(data))
print("LRC:", lrc)
print("Transmitted Message:", " ".join(transmitted_message))

# Simulating error
received_data = ["1010", "1100", "1011"]
received_lrc = lrc

if check_lrc(received_data, received_lrc):
    print("Receiver Check: No Error Detected")
else:
    print("Receiver Check: Error Detected")
```

```
Original Message: 1010 1100 1111
LRC: 1001
Transmitted Message: 1010 1100 1111 1001
Receiver Check: Error Detected
```

In [2]:
```
def calculate_vrc(data):
    parity_bit = str(data.count('1') % 2 == 0).replace('True', '0').replace(
    return data + parity_bit

def check_vrc(received_data):
    return received_data.count('1') % 2 == 0

data = "1101011"
transmitted_message = calculate_vrc(data)
print("Original Message:", data)
print("Even Parity Bit:", transmitted_message[-1])
print("Transmitted Message:", transmitted_message)
```

```
# Simulating error
received_data = transmitted_message[:-1] + "1"
if check_vrc(received_data):
    print("Receiver Check: No Error Detected")
else:
    print("Receiver Check: Error Detected")
```

```
Original Message: 1101011
Even Parity Bit: 1
Transmitted Message: 11010111
Receiver Check: No Error Detected
```

In [3]:
```
def calculate_checksum(words):
    checksum = sum(int(word, 2) for word in words) & 0xFFFF
    checksum = (~checksum) & 0xFFFF
    return format(checksum, '016b')

def verify_checksum(words, received_checksum):
    computed_checksum = calculate_checksum(words)
    return computed_checksum == received_checksum

data = ["1010101010101010", "1100110011001100"]
checksum = calculate_checksum(data)
transmitted_message = data + [checksum]
print("Original Message:", " ".join(data))
print("Checksum:", checksum)
print("Transmitted Message:", " ".join(transmitted_message))

# Simulating error
received_data = ["1010101010101010", "1000110011001100"]
received_checksum = checksum

if verify_checksum(received_data, received_checksum):
    print("Receiver Check: No Error Detected")
else:
    print("Receiver Check: Error Detected")
```

```
Original Message: 1010101010101010 1100110011001100
Checksum: 1000100010001001
Transmitted Message: 1010101010101010 1100110011001100 1000100010001001
Receiver Check: Error Detected
```

In [4]:
```
def xor(a, b):
    return ''.join('0' if i == j else '1' for i, j in zip(a, b))

def divide_crc(data, divisor):
    n = len(divisor)
    temp = data[:n]
    while n < len(data):
        if temp[0] == '1':
            temp = xor(temp, divisor) + data[n]
        else:
            temp = xor(temp, '0'*n) + data[n]
        temp = temp[1:]
        n += 1
    if temp[0] == '1':
        temp = xor(temp, divisor)
```

```
    else:
        temp = xor(temp, '0'*len(divisor))
    return temp[1:]

def encode_crc(data, generator):
    padded_data = data + '0'*(len(generator)-1)
    crc = divide_crc(padded_data, generator)
    return data + crc

data = "101001"
generator = "1101"
transmitted_message = encode_crc(data, generator)
print("Original Message:", data)
print("CRC:", transmitted_message[len(data):])
print("Transmitted Message:", transmitted_message)

# Simulating error
received_data = "111001" + transmitted_message[len(data):]

if divide_crc(received_data, generator) == '0'*(len(generator)-1):
    print("Receiver Check: No Error Detected")
else:
    print("Receiver Check: Error Detected")
```

```
Original Message: 101001
CRC: 001
Transmitted Message: 101001001
Receiver Check: Error Detected
```

In [5]:

```
def hamming_encode(data):
    d = list(data)
    p1 = str((int(d[0]) + int(d[1]) + int(d[3])) % 2)
    p2 = str((int(d[0]) + int(d[2]) + int(d[3])) % 2)
    p4 = str((int(d[1]) + int(d[2]) + int(d[3])) % 2)
    return p1 + p2 + d[0] + p4 + d[1] + d[2] + d[3]

def hamming_decode(received):
    p1 = int(received[0])
    p2 = int(received[1])
    d1 = int(received[2])
    p4 = int(received[3])
    d2 = int(received[4])
    d3 = int(received[5])
    d4 = int(received[6])

    c1 = (p1 + d1 + d2 + d4) % 2
    c2 = (p2 + d1 + d3 + d4) % 2
    c4 = (p4 + d2 + d3 + d4) % 2

    error_pos = c1 * 1 + c2 * 2 + c4 * 4

    if error_pos == 0:
        print("Receiver Check: No Error Detected")
    else:
        print(f"Receiver Check: Error Detected at Bit Position: {error_pos}"
        received = list(received)
```

```
        received[error_pos - 1] = '1' if received[error_pos - 1] == '0' else
        print("Corrected Message:", ''.join(received))

data = "1011"
encoded_message = hamming_encode(data)
print("Encoded Message:", encoded_message)
print("Transmitted Message:", encoded_message)

# Simulating error
received_message = "1100011"

hamming_decode(received_message)
```

```
Encoded Message: 0110011
Transmitted Message: 0110011
Receiver Check: Error Detected at Bit Position: 2
Corrected Message: 1000011
```

In [ ]: