DATA LINK LAYER

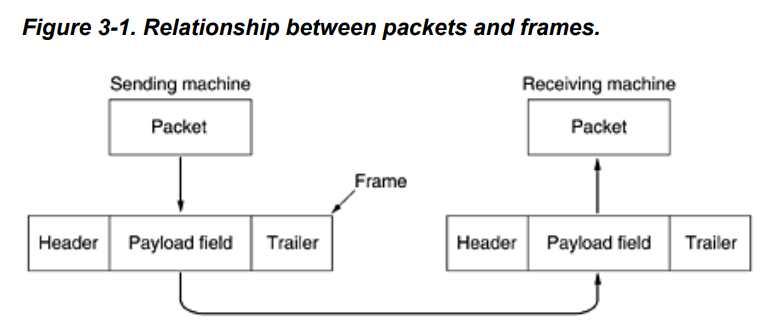
Frames

The **f**unctions of the data link layer are:-

1. Providing a service interface for the network layer.
2. Dealing with the transmission errors.
3. Regulating the flow of data so that the slow receivers are not swamped by the fast senders.

To accomplish these goals, the data link layer , takes the packets from the network layer and encapsulates them into frames . A frame contains

* Header
* Payload field for holding the packet
* Frame trailer.



The principle service is providing data from the network layer of the source machine to the network layer on the destination machine see the figure.

Data

Network Layer

Network Layer

Destination Machine

Source Machine

The services provided by this layer;

1. Unacknowledged connectionless service
2. Acknowledged connectionless service
3. Acknowledged connection-oriented service

Unacknowledged connectionless service(UDP):

* It doesn’t establishes a logical connection.
* The source machine sends independent frames to the destination machine without needing the frames to acknowledge them.
* Even if the frame is lost, no attempt is made to recover it, that is why it is used in real time traffic, such as voice where getting late data would be better than having a bad data.
* A good implementation of it would be LAN cables

Acknowledged connectionless service(WIFI):

* The only difference is that it in it, the acknowledgment is sent back to the user.
* An example is an unreliable wireless network and a reliable fibre network(where the packets loss are minimal)
* As in the example of WIFI, the sender knows whether the frame has arrived. If it hasn’t arrived within a specified time interval, then an attempt can be made to send it again.

Acknowledged connection-oriented service(TCP):

* Here first a logical connection is established between the end users.
* Each frame sent over the connection is numbered and the data link layer makes sure that each frame is indeed received *exactly once*.
* It also makes sure that the packets are received in the right order.
* It is unlike the connectionless service where the unacknowledged packets are sent several times and that results to them being received several times.

**2) ERROR CONTROL**

The problem arrives while the transmission of the frames from source to destination , in a proper order. When the machine is independently outputting the frames without making sure whether they are arriving at the destination or not, this isn’t a problem for the unacknowledged connection oriented service, but it is not fine when we have a reliable connection oriented service.

The usual way to ensure a reliable delivery of frame is to provide the sender with some feedback about what is happening at the other side of the line. This can be done by-:

* Making the receiver sending back special control frames that bear positive and negative acknowledgements with them.
* If a positive acknowledgment has been received to the sender, that means that the frames have arrived safely.
* Else it means that something has gone wrong in the frame transmission.

Another possibility is that a frame can vanish forever in case of a hardware malfunctioning, in such case the receiver can’t react at all.

This possibility is dealt by introducing timers:

* When the sender transmits a frame, it generally starts a timer.
* It is set to expire for the interval long enough till it arrives at the destination, be processed there and then the acknowledgment propagates back to the user.
* Normally the frame is received correctly and the acknowledgement is received before the time and hence the timer is cancelled.

But if the frame and the acknowledgment is lost:

* The timer will go off, which will alert the sender that the frame needs to be retransmitted again, in that case the frame will be sent back again.

Another possibility is when the frames are transmitted multiples time to the destination, then that causes problem. This is solved by associating a serial number to each frame.

**3) FLOW CONTROL**

So the design issue in the data link layer is that when the sender systematically wants to sent the frames at a faster rate than the receiver can receive them. This becomes a problem when the sender is running on a fast (lightly loaded) computer and the receiver is running on a slow (heavily loaded computer) computer. The sender then is pumping out frames at a very higher rate and the receiver gets completely swamped by them. Even if the transmission occurs error free, there is a chance that at some point of time, the receiver starts losing the frame as it might not be able to handle it.

Two approaches are commonly used in flow control:-

* **Feedback based flow control**: the receiver sends some information to the sender giving permission to send more frames or at least telling the sender about how is it doing.
* **Rate based flow control**: This protocol has an in built functionality that limits the rate at the which the sender may transmit the data, it does not require the need of any feedback based system.

The *data link layer talks only about the feedback based flow control*, as the rate based flow control is done at some other layer.

1. **Error detection and Correction**

So the network designers came up with two methods of dealing with the errors-

1. Error-correcting code:

Here enough redundant information is added along with the block of data that enables the receiver to deduce what the transmitted data must have been.

1. Error-detecting codes:

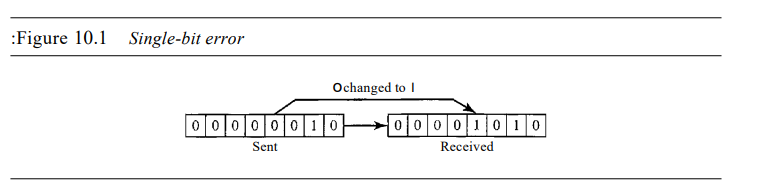
Only that much redundant information is added with the block that receiver will be able to deduce that an error has occurred, but not what error.

Those redundant bits are also known as the Parity bits.

Each of these channels occupy a different ecological niche. On highly reliable channels such as an optical fibre, use of error detecting codes would be cheaper as it may just retransmit the faulty block. But with the unreliable channels such as wireless network that contains so many errors, using error correction codes would be useful.

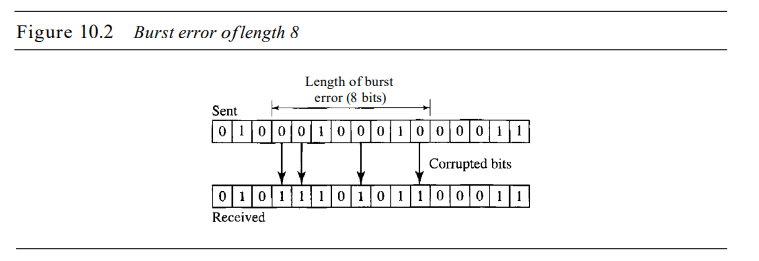
Single-Bit Error

The term single-bit error means that only 1 bit of a given data unit (such as a byte, character, or packet) is changed from 1 to 0 or from 0 to 1.



Burst Error

The term burst error means that 2 or more bits in the data unit have changed from 1 to 0 or from 0 to 1.



* 1. **Hamming distance and Hamming Code**

In order to handle the error, it is really important to understand what an error really is.

Normally a frame consists of:

* m (message) bits
* r redundant /check bits
* so (n=m+r ) would be the total length that can be called as an n-bit codeword.

Given any two codewords say;

Exclusive XOR the two code words, the number of 1 bits are the difference between those code s.words.

1001001

1011010

0010011

It is possible to determine how many corresponding bits differ. In this case the difference is of 3 bits.

So the number of bits the two codewords differ is the hamming distance between them.

Hamming Code | Error detection and Correction

These are the categories of error-correcting codes. These codes were originally designed with dmin=3 (minimum hamming distance) that means that it can detect upto 2 errors and correct 1 error.

In most of the cases , all 2^m data messages are legal, but not all 2^n codewords are used (due to some ways of computing the check bits).

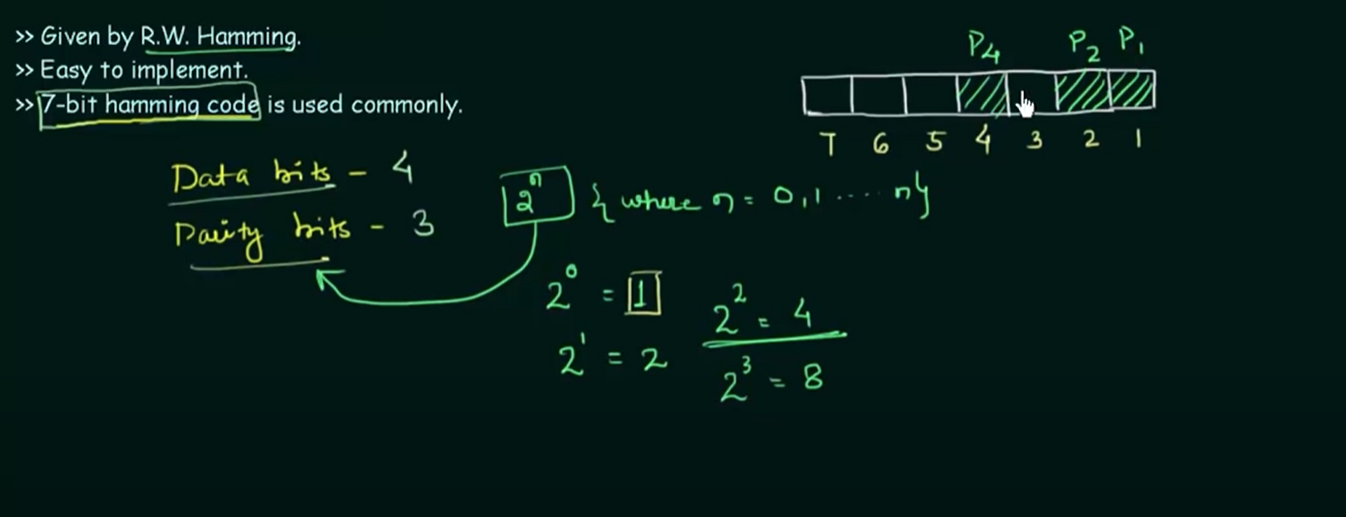
To guarantee the detection of up to s errors in all cases, the minimum Hamming distance in a block code must be dmin =S + 1.

To guarantee correction of up to t errors in all cases, the minimum Hamming distance in a block code must be dmin == 2t + 1

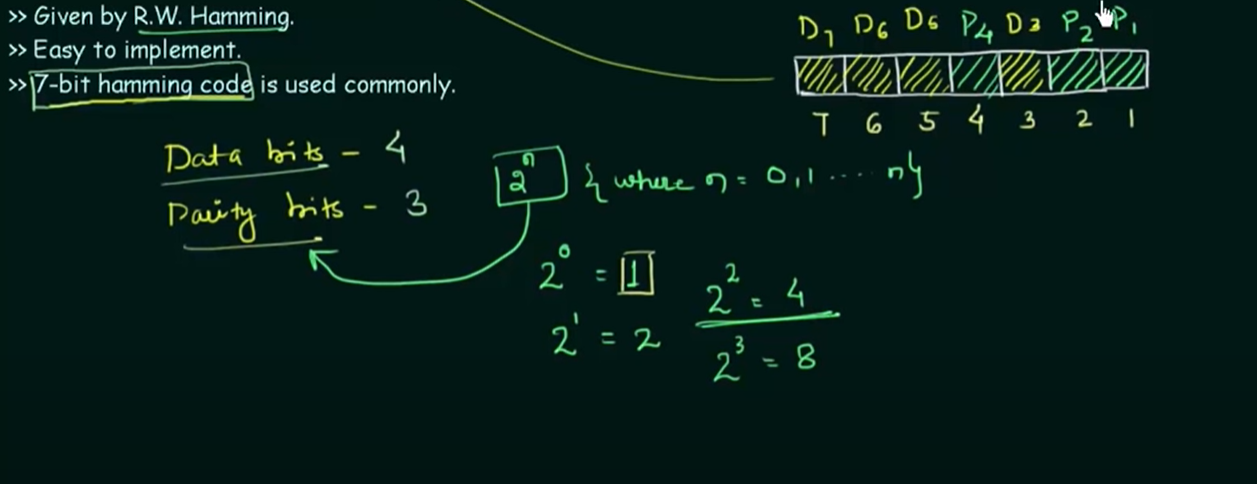
* Usually a 7-bit hamming code is used.
* So in this case we have

1. 4 data bits
2. 3 parity bits

Position of parity bits



The rest are the data bits



So,

Depends on

P1 🡪 D3 D5 D7

0 1 1 -> lets say the value of those data bits

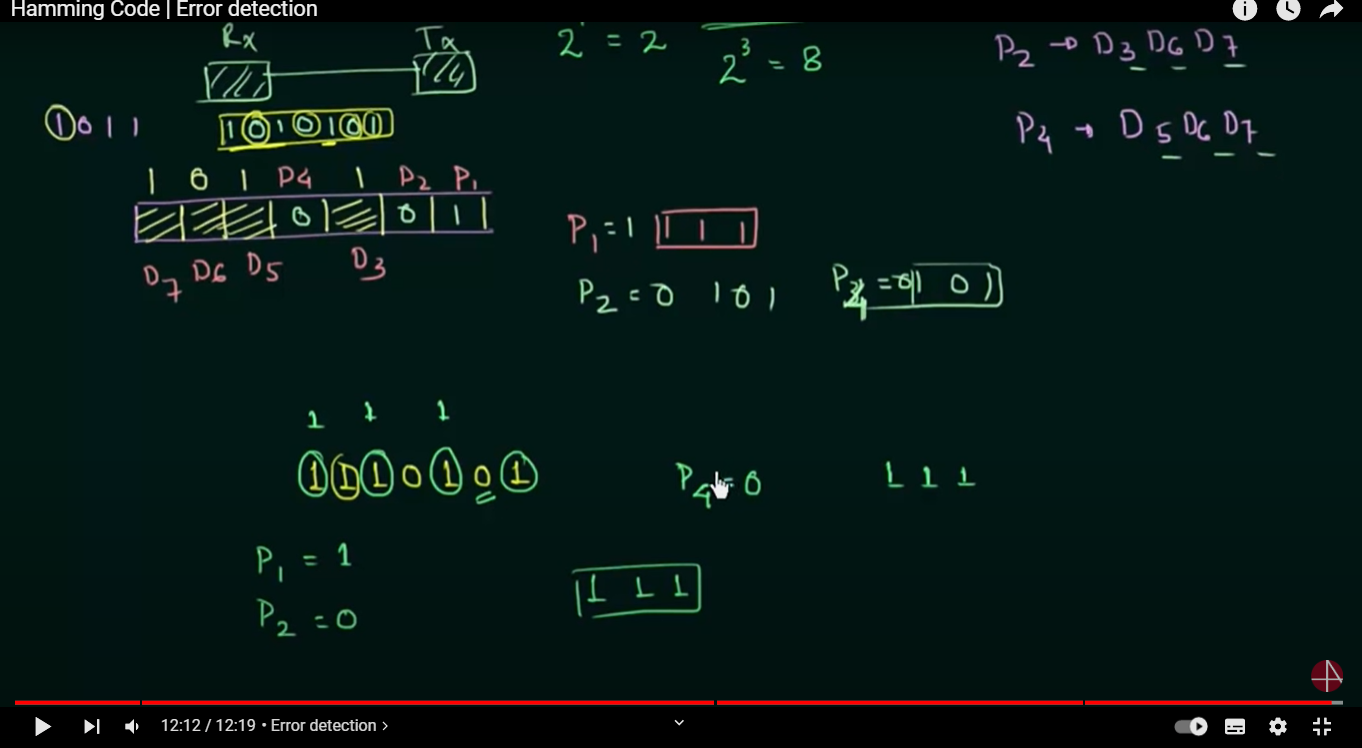
So to make it of even parity, P1=0

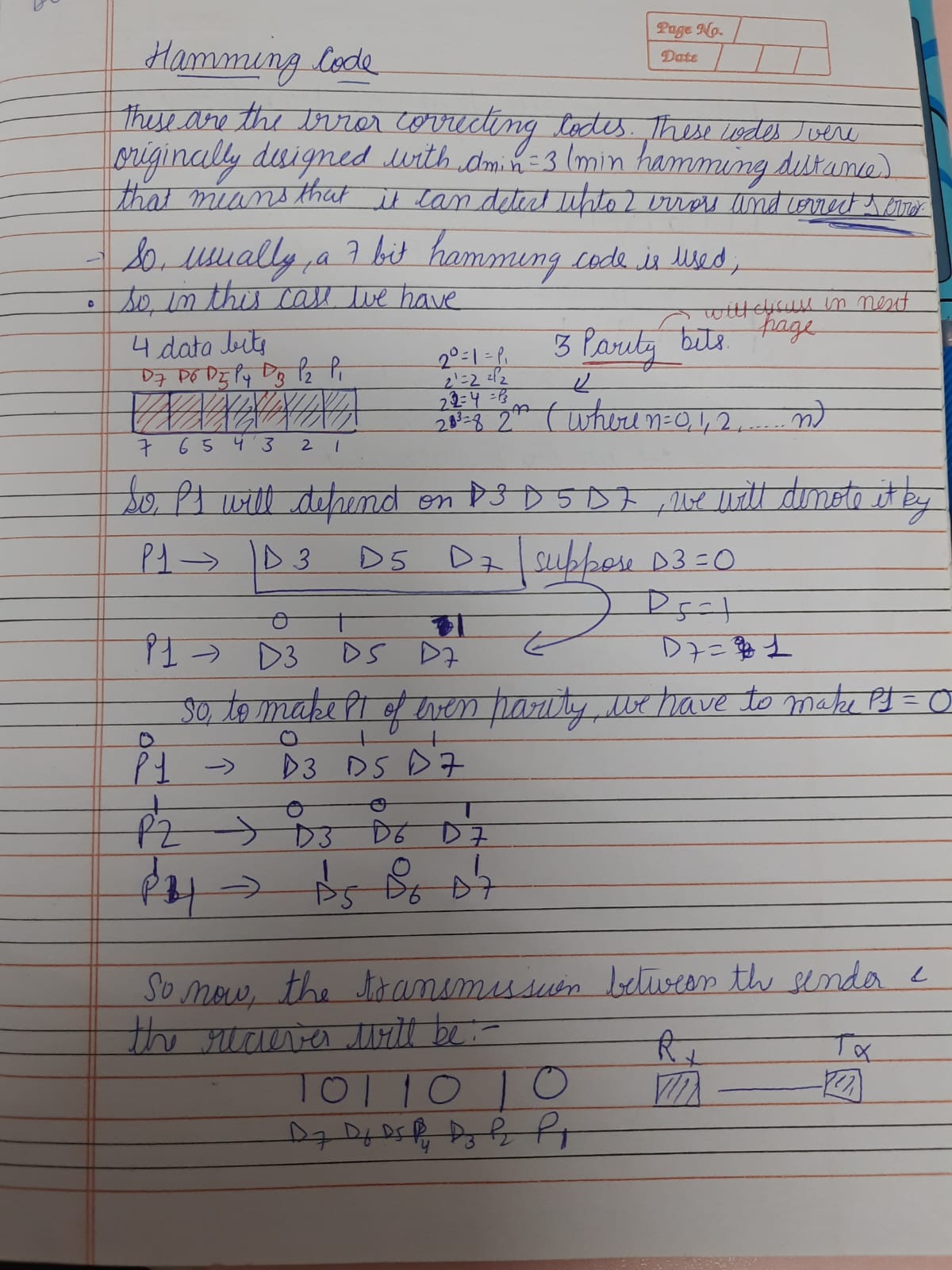
P1 🡪 D3 D5 D7

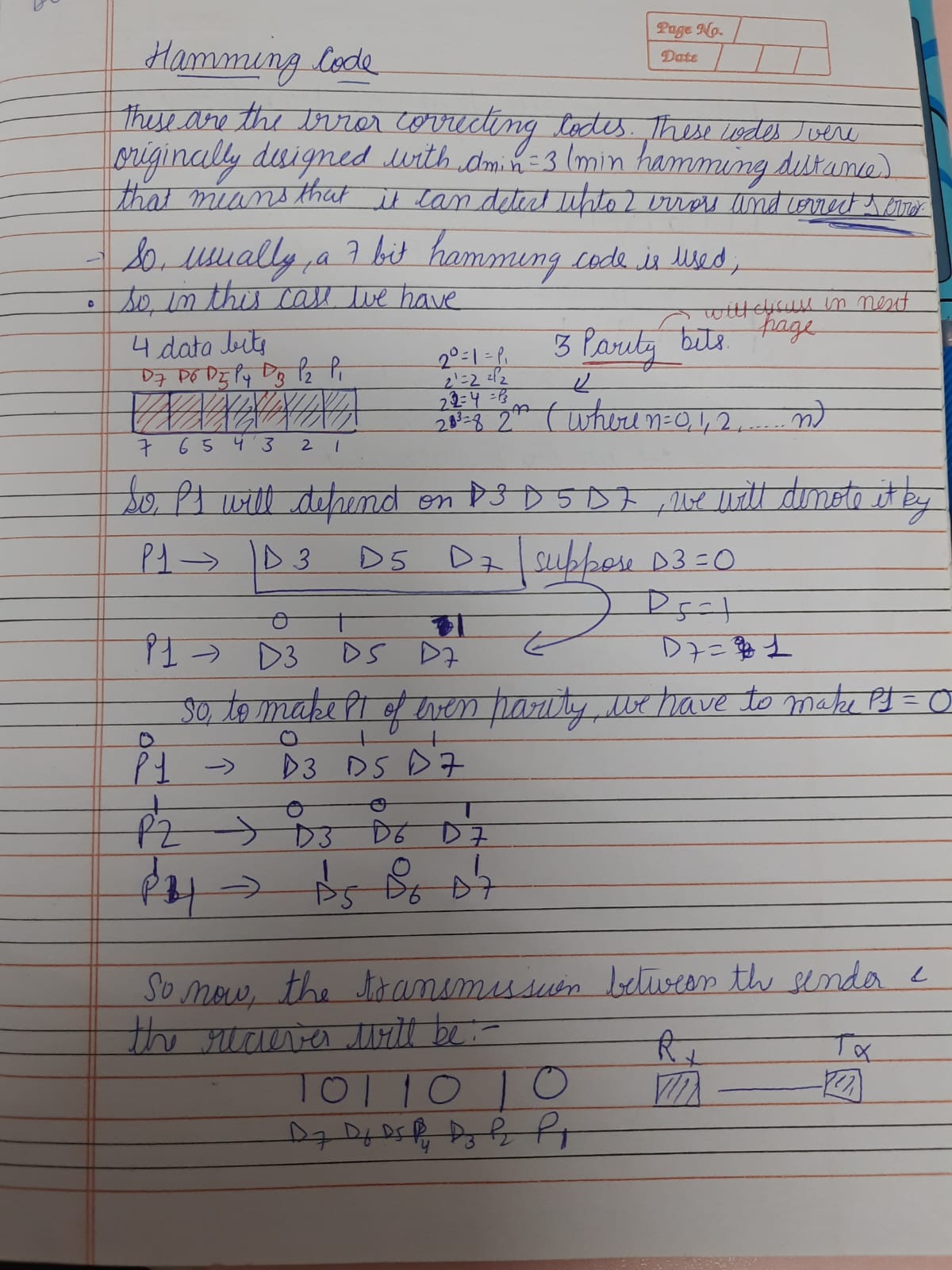
0 1 1 1

P2 🡪 D3 D6 D7

P3 🡪 D5 D6 D7







* 1. Parity |Error detection

It is a concept to detect error.

A single bit error is detected by it.

Transmission end

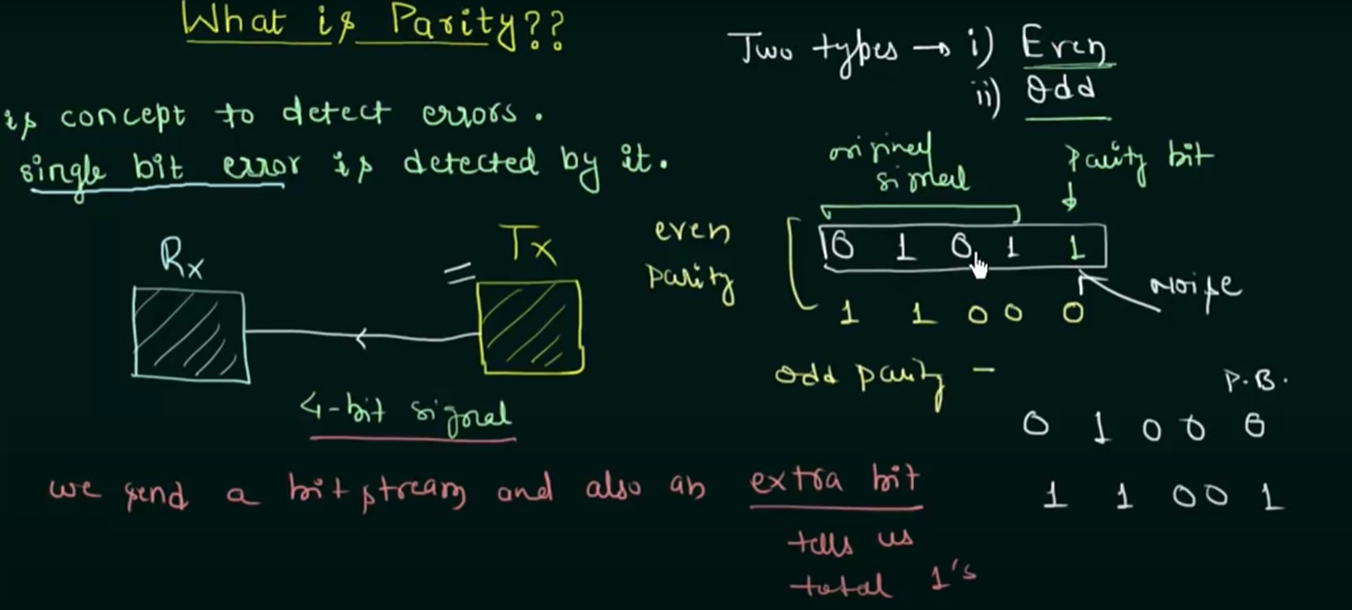
Receiving end

4 bit stream

When there is a noise in the transmission bit stream, some zeroes get converted to 1 and vice versa, so it becomes low to high and high to low.

Parity is the extra bit that is added along with the bit stream to make the number of 1 bits even or odd according to the implementation.

It has the minimum hamming distance of dmin=2, that means that it is a single bit error detecting code, and it can correct no error.



As you can see in the figure above, there are two types of parity, in

* Odd
* Even

In Even parity, we have to add an extra parity bit such that the total number of 1 bits become even

In odd parity, , we have to add an extra parity bit such that the total number of 1 bits become odd.

So in hamming code The total number of 1s in each 4-bit combination (3 dataword bits and 1 parity bit) must be even.

In other words, each of the parity-check bits handles 3 out of the 4 bits of the dataword.

* 1. CRC – Cyclic Redundancy Checker| Error detection

Cyclic codes are special block codes with one additional property. In a codeword, if we shift it by to clockwise left or right, the resulting word is also a code word. Say a codeword 100111 is when shifted cyclically to the left it becomes 001111, a new code word. These codewords can be used to check errors by the following;

Lets say we call the bits in the first codeword to a0 to a6  and the bits in the second codeword - b0 to b6 ,So now on applying right circular shift ,

b0 = a6 b1 = a0 b2 = a1 b3 =a2 b4 = a3 b5 = a4 b6 = a5

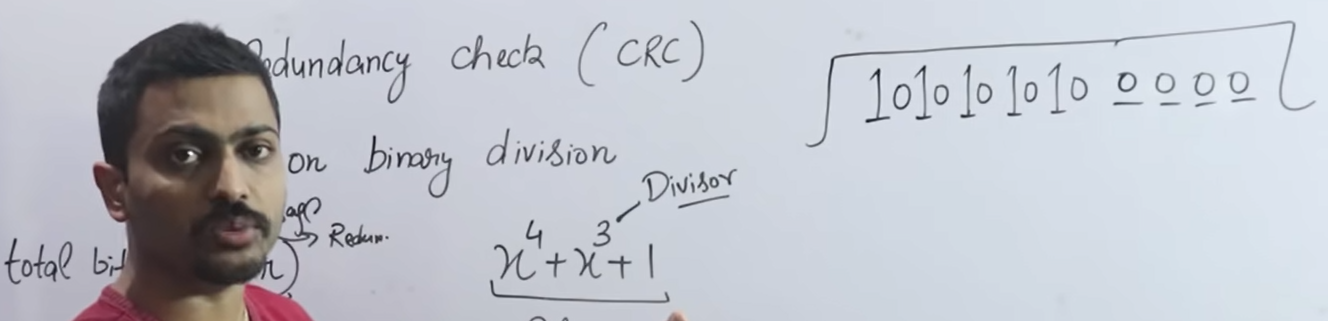
So CRC is widely used error checker methods and is widely used in WANs, LANs.

It can detect all odd errors, single bit errors, burst errors of length equal to polynomial degree.

For solving the numerical ,

1. We will append r bits to the dividend

r=highest degree of the given polynomial (divisor)



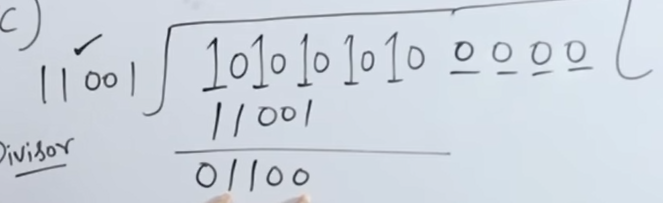
1. For deducing the divisor in the binary form, take out the coefficients from the polynomial.



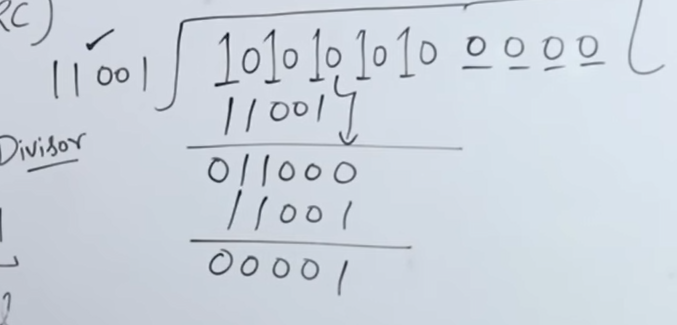
(in case you are provided with the binary form of the divisor only, then you don’t have to append that length, but have to subtract -1 from it)

1. Now we have to perform binary division, for that we use XOR operation.

Here we do not require a quotient, we just require a remainder.

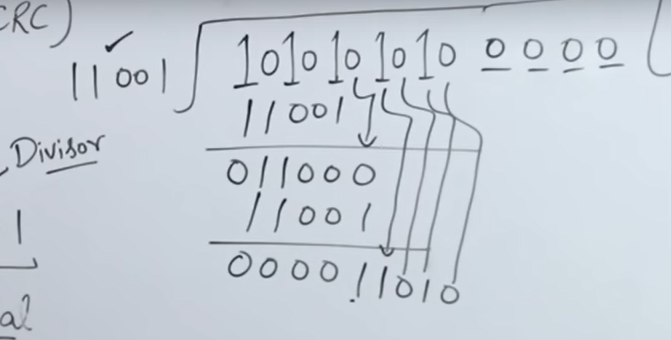


1. Next we will ignore the 0’s from the left and continue from the leading 1, and then carry 0 from the dividend.



In this step, in the remainder, we have 4 zeroes, so will start from the 5th bit , i.e 1and since the divisor is of length 5, we will carry 4 more bits from the dividend.

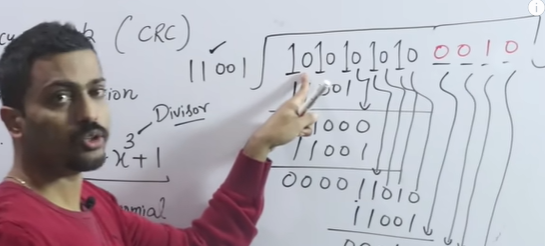
1. So we will have



1. The remainder we will get, we have to append it in the place of those appended zeroes



Like in this examples, the remainder contained length of 6, so remember to append the last 4 bits only (as the appended 0’s were also 4)



1. So for checking the error, the receiver will divide the dividend 10101010100010 , so if the remainder contains all zero, means that the message is error free, and if the remainder has any of 1 bit, means there is an error.
2. To check the efficiency or channel utilisation, you can

Dataword (m:message)

codeword (n=m + r , r=redundant bits)

* 1. Checksum | Error detection

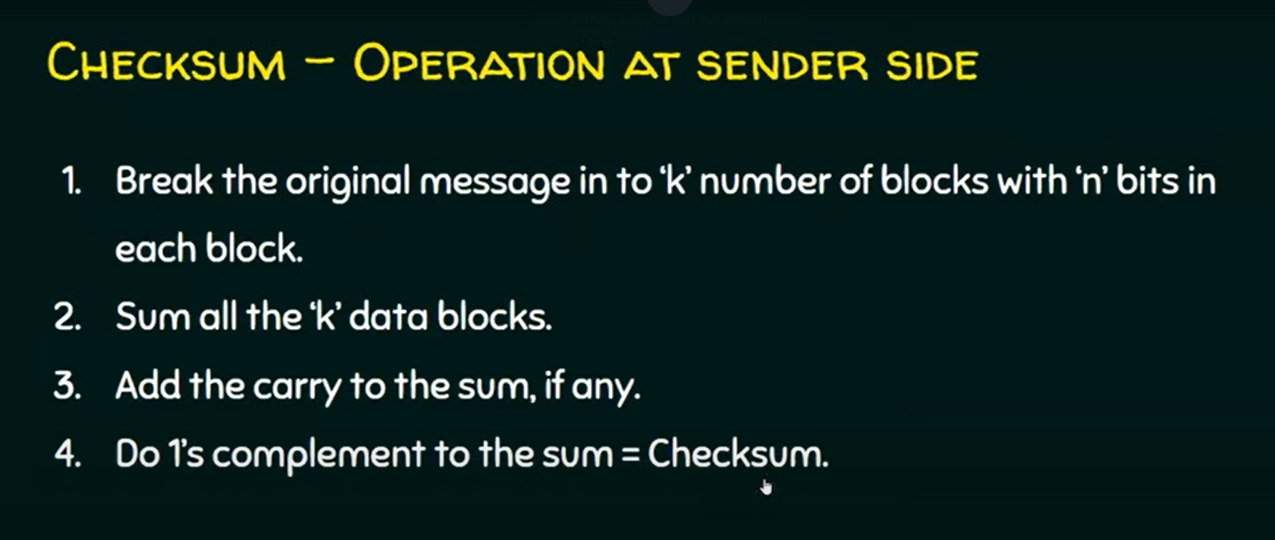
Checksum is another error detection method .It is used in various protocols, but not in the data link layer. Like linear and cyclic codes, checksum is based on the concept of redundancy.

**Example 4.4.1**

Suppose our data is a list of five 4-bit numbers that we want to send to a destination. In addition to sending these numbers, we send the sum of the numbers. For example, if the set of numbers is (7, 11, 12, 0, 6), we send (7, 11, 12,0,6,36), where 36 is the sum of the original numbers. The receiver adds the five numbers and compares the result with the sum. If the two are the same, the receiver assumes no error, accepts the five numbers, and discards the sum. Otherwise, there is an error somewhere and the data are not accepted.

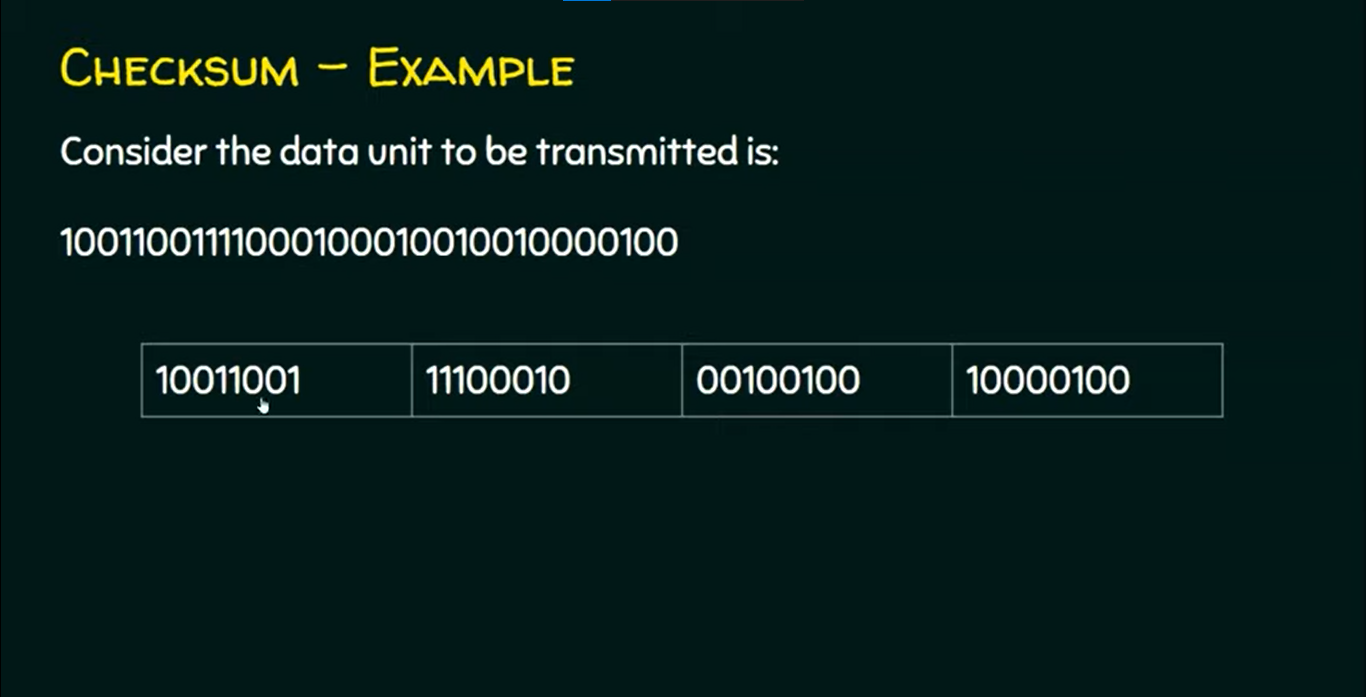
**Example 4.4.2**

We can make the job of the receiver easier if we send the negative (complement) of the sum, called the checksum. In this case, we send (7, 11, 12,0,6, -36). The receiver can add all the numbers received (including the checksum). If the result is 0, it assumes no error; otherwise, there is an error.

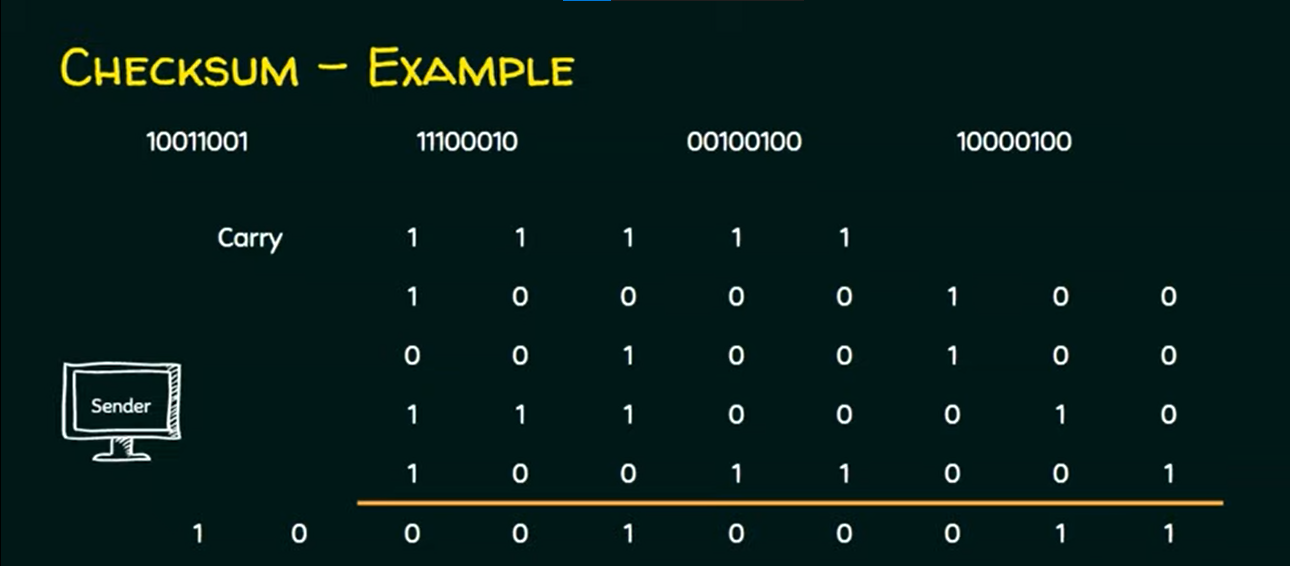




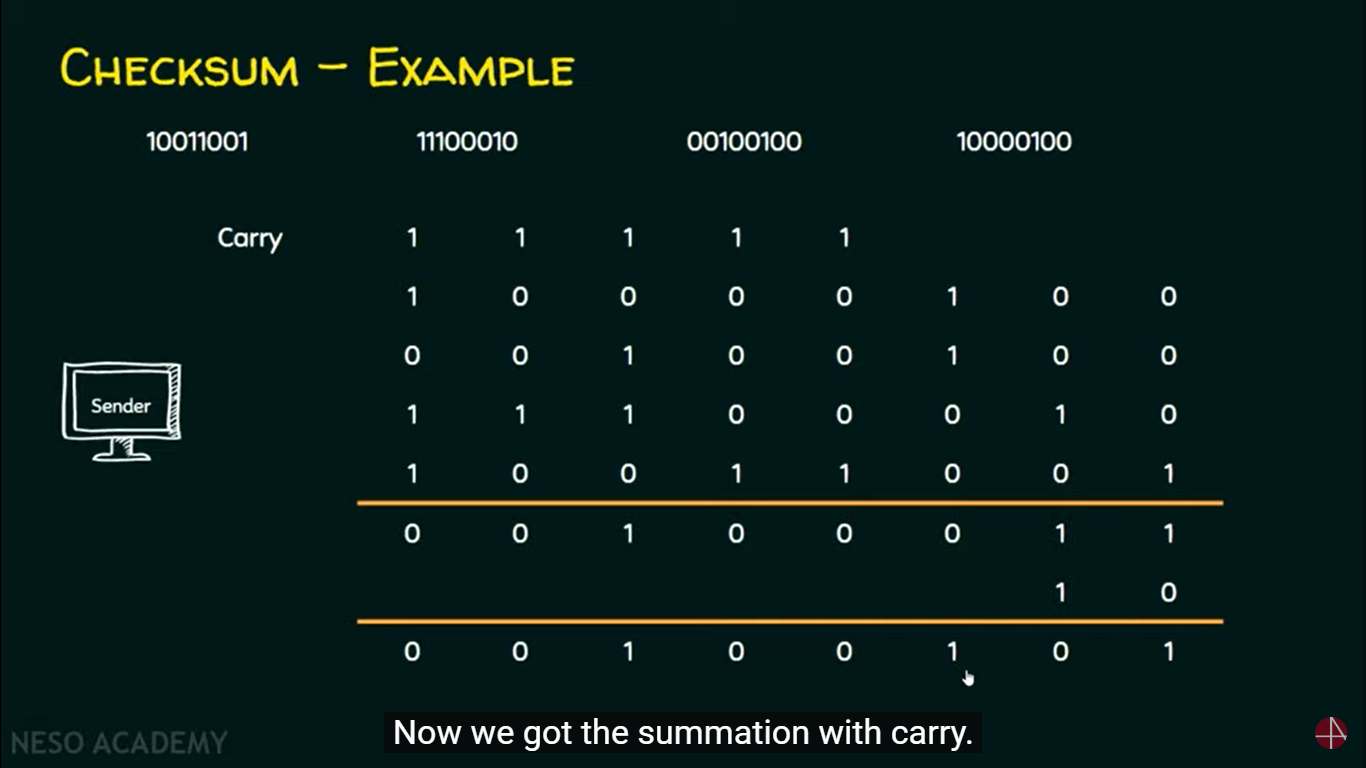
Break down the message into bits,



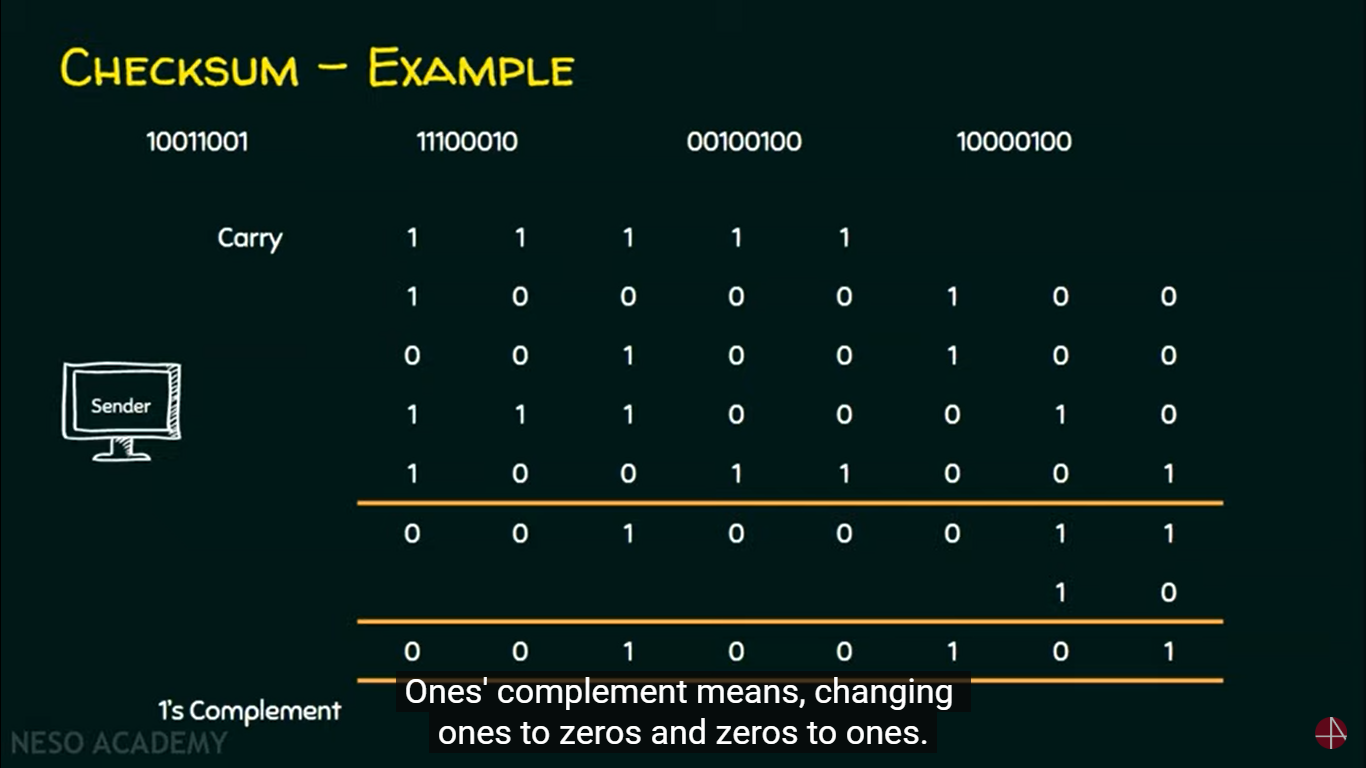
Perform the binary addtion

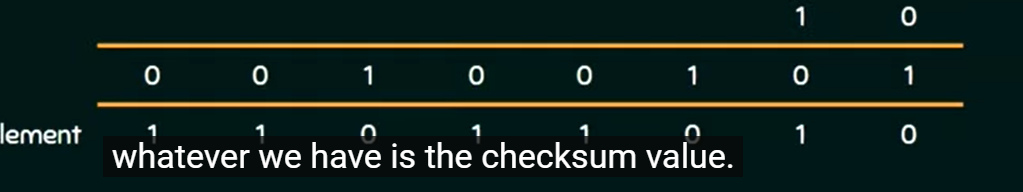


If there are extra bit in the carry, add them at the last of the answer like;



Now we will perform the 1’s complement





We have got 11011010 as the checksum value, we will append it to the message and send to the receiver, The receiver will perform the binary addition and if

It sum is all 1’s

* Accepted

Else

* Rejected

4.4.3) Performance of the checksum

It is not as strong as CRC in the error checking capability, as say if the value of one word is incremented and the value of other word is decremented by the same value, the sum and the checksum will remain the same so the error wont be detected.

Also if the values of several words are incremented but the total change is a multiple of 65535, the sum and the checksum does not change, which means the errors are not detected.