⇒ Error Control policies of D.L.L :-

↳ "Error Correction policies" :-

"Hamming Code" :-

Data + Parity bits = Code word.

Let Data = 10011010 → 8 bits

:- $2^r \geq m + r + 1$ [m = message bits, r = Parity bits]

$r = 3$

$2^3 \geq 8 + 3 + 1$

$8 \geq 12$ ✗

$r = 4$

$2^4 \geq 8 + 4 + 1$

$16 \geq 13$ ✓

$r = 4$ ✓

Parity bits will be required

★ :- Parity bits should be placed in power of 2. Positions!

1	2	3	4	5	6	7	8	9	10	11	12
P_1	P_2	1	P_4	0	0	1	P_8	1	0	1	0
		↑		↑	↑	↑					

message bits

P_1 : 1, 3, 5, 7, 9, 11 → get values

↳ 1 difference

0 1 0 1 1 1

$P_1 = 0$, No of Parity bits will be even.

P_2 : 2, 3, 6, 7, 10, 11

✓1 1 0 1 0 1

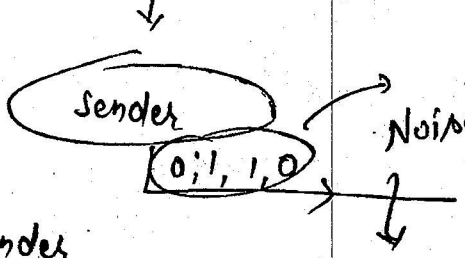
$P_2 = 1$ ↺ No of Parity bits should be even.

P_4 : 1 0 0 1 0 $\quad P_4 = 1$

P_8 : - 8, 9, 10, 11, 12

0 1 0 1 0 $\quad P_8 = 0$

Data = 1 0 0 1 1 0 1 0



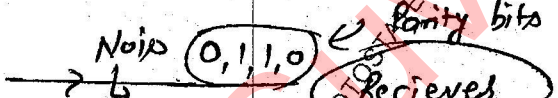
0 1 1 0 | 0 1 1 0 | 0 1 1 0 ← reliable copy of Parity bits

Sender Codeword = 0 1 1 1 0 0 1 0 1 0 1 0 (9 10)

P_1 P_2 P_4 P_8 ↑

[substitution of Parity bits values

Codeword = Data + Parity bits = 8 + 4 = 12 bits



Parity bits copy.

Recieved Codeword = 1 2 3 4 5 6 7 8 9 10 11 12

0 1 1 1 0 0 1 0 1 1 1 0

P_1 P_2 1 P_4 0 0 1 P_8 1 1 1 0

($P_1 = 0$, $P_2 = 1$, $P_4 = 1$, $P_8 = 0$) ↙ Recieved Parities in Recieved codeword.

Reciever Codeword = 1 2 3 4 5 6 7 8 9 10 11 12

P_1 P_2 1 P_4 0 0 1 P_8 1 (1) 1 0

error

P_1 : 1, 3, 5, 7, 9, 11

0 1 0 1 1 1 $\quad P_1 = 0$ calculated ✓ recieved $P_1 = 0$

P_2 : 2, 3, 6, 7, 10, 11

0 1 0 1 1 1 $\quad$ should be even No of 1's so ($P_2 = 0$) ✗ recieved $P_2 = 1$

P_4 : 4, 5, 6, 7, 12

1 0 0 1 0 $\quad P_4 = 1$ ✓

r_8 = 8, 4, 10, 11, 12

= 1 1 1 1 0 $P_8 = 1$ ✗ recieved $P_8 = 0$

:- The bit is modified $\Rightarrow$ By P_2 & P_8 :- 2+8

= 10th bit error

Drawback :- Hamming code will correct only single bit errors.

* If parity bits are modified, reciever will be knowing immidietly by comparing it reliable copy.

⇒ Error Detection policies :-

(1.) Parity scheme :-

x	y	output (even parity)
0	0	0
0	1	1
1	0	1
1	1	0

Ex-OR

⇒ valid code (0,0,0)

→ invalid code (0,0,1)

↑ Ckt can't generate

→ 3 bit code

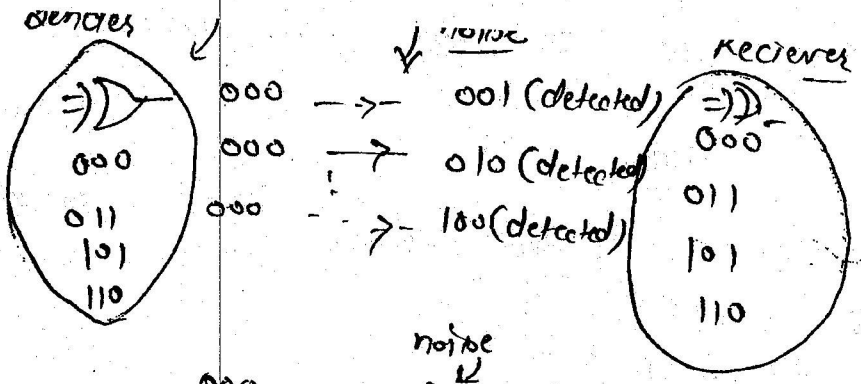
x 0,0 0,1

y 0,1

← A ckt

Codeword = Data + Parity bits.

* Codewords which are generated from a circuit are known as valid codewords and remaining all are invalid codewords.



000 —noise→ 011 (Not detected) another valid Codeword.

★ A valid codeword ip converted into an invalid codeword by noise then errors can be detected.

★ A valid codeword ip converted into another valid codeword then errors cannot be detected.

:- Hamming distance :- The Number of bits, the codewords differ from each-other ip the hamming distance.

★ To detect "d" errors, the minimum hamming distance ip d+1

★ To detect "1" error, the minimum hamming distance ip 2

★★ To correct d errors
min. hamming distance
= 2d+1

2 { 000 } 2
{ 011 } 2
{ 101 } 2
{ 110 } 2
hamming distance = 2

So 1 error can be detected only.

Drawback :-

★ Note :- :- The Parity scheme can detect only odd no of errors.

:- the ~~odd~~ Even No of errors (2, 4, 6 :-) can't be detected by Parity scheme.

Hence, ... scheme can not be apply in LAN Network for detecting the errors.

(1)

Sender

Reciever

0000 0000 ... 1111 1111

0000 0000 } Hamming distance
0000 1111 } 4
1111 0000 } 8
1111 1111 } 4

min. hamming distance = 4

$d+1 = 4$

$d = 3$

:- only 3 bit error can be detected.

"Checksum" :-

Data + checksum bits = Codecword

Sender

115265 70

Data checksum

11
52
65
1 28
+1
29

9's compliment

99
-29
70

Sender's checksum

Case (i)

noise

Data bits are modified

115963 70

Recieved checksum = 70

11
59
63
1 33
+1
34

checksum 65 ≠ 70

Both are not same so errors can be detected.

9's compliment

99
-34
65

Calculated checksum at recieves

115 26570 — — → — — ↓ — 115265 95

checksum bits are modified.

↑ Recieved checksum.

11
52
65
128
+1
29

9's compliment
99
-29
70

Calculated checksum :- 70

Both checksum are not equal, so data will not be accepted.

115 265 70 — — — → ↓ noise — — — 11506 70

↑ Case(iii)

Recieved Checksum

11
50
+2 67
128
+1
29

Vertical bit errors

9's compliment
99
-29
70

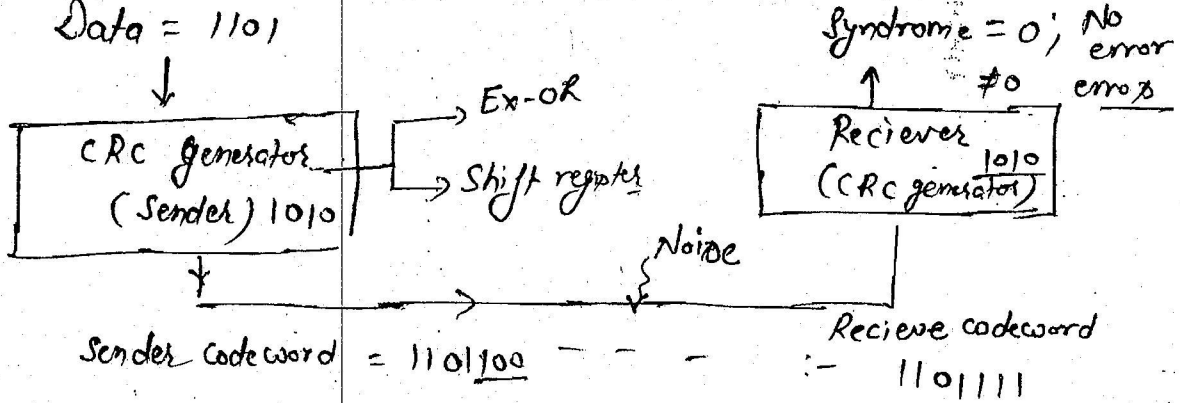
Calculated Checksum

error not detected

Drawback :-

★ If Noise modifies the data in such a manner, the vertically placed bits cancel each-other then the calculated checksum will be equal to that of recieved checksum then such errors can't be detected, those errors are known as vertical bit errors.

"CRC (Cyclic Redundancy check)" :-



CRC (Cyclic redundancy check) :-

CRC generator = 1010

CRC generator polynomial = 1010

$= x^3 + x$

Sender :-

```
1010 ) 1101 000 ( 1110
       1010
       ----
        1110
        1010
        ----
         1000
         1010
         ----
          0100
          0010
          ----
           110
```

Padding bits = generator bits − 1

4 − 1 = 3

No of remainder bits = padding bits

= 3

Reciever :-

```
1010 ) 1101 011 ( 1110
       1010
       ----
        1111
        1010
        ----
         1011
         1010
         ----
          0011
          0000
          ----
           011
```

← syndrom ≠ 0 i.e errors

Note :- A good generator can detect all types of errors in CRC.

CRC Answers

* CRC generator should not contain "x".

Data ⇒ 1101

CRC generator = $x = 1 \cdot x^1 + 0 \cdot x^0$

$= 10$

```
10 ) 11010 ( 1101
     10
     ---
     10
     10
     --
     01
     00
     --
     10
     10
     --
     0
```

↳ Padding bit

Sender Codeword : 11010 ——— Noise ——— errors : -10010 → Recieved Codeword

```
10 ) 10010 ( 1001
     10
     --
     00
     00
     --
     01
     00
     --
     10
     10
     --
     0   ⇐ Syndrom
```

So no errors

So, bad generator

imp.

* x+1 is a generator if can detect odd no of errors.

Data = 1101

CRC generator = $x+1 = 1 \cdot x^1 + 1 \cdot x^0$

$= 11$

```
11 ) 11010 ( 1001
     11
     --
     00
     00
     --
     01
     00
     --
     10
     11
     --
     1
```

```
11 ) 11010 ( 1001
     11
     --
     00
     00
     --
     01
     00
     --
     10
     11
     --
     1   → errors
```

Sender Codeword : 11011 ——— Noise case(i) ——— 1 bit error ——— Recieved codeword : 11010

* CRC-32 is used for detecting all types of errors in the LAN Network.

11011 ——— → (Noise) ↓ (capacity) — 11000

2 bit errors

↳ can't detect

11) 11000 (1000
11
00
00
00
01
0 ⇐ syndrom

i.e. No error

11011 ——— ↓ Noise ——— 111000

↳ 3 bit error

↳ detected

11) 111000 (1011
11
11
10
00
00
01
↑
0 syndrom

error detected

"Framing" :-

* The Efficiency of any error detection scheme decreases as the length of data increases.

* Dividing the large amount of data into small parts, so that errors can be detected easily, by the error detection policies is known as 'framing'.

S
1000 ——→ R
1001 ↩ 1 bit modified

4c, ⇒ 0001
1101
1011
1000

1000 ——→ 1011 ↩ 2 bit modified

4c. = 6

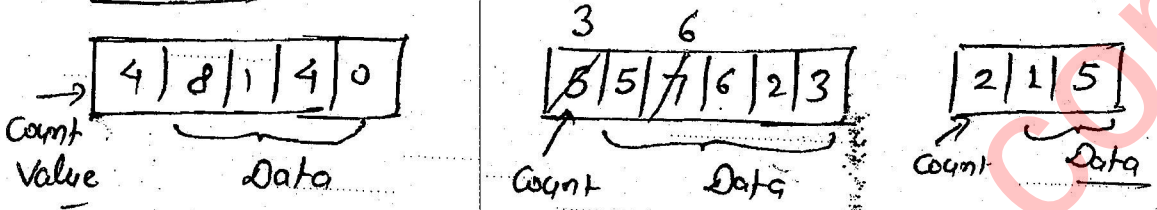
9999 — ÷ 2 → 9999 C_2 ⇒ large

2 bit modified

★ Character Count

N.L. :- 8 1 4 0 5 7 6 2 3 15

S'DLL



★ In character count technique, count value indicates the size of the frame.

★ If Noise modifies the data CRC can protect the data.

★ But if Noise modifies the count value both sender and Receiver are out of synchronization.

★ Character Stuffing :-

N.L. ⇒ AZ

S'DLL :- FLAG AZ FLAG ——— R'DLL: FLAG AZ FLAG

N.L → AZ

FLAG 01111110

N.L A FLAG B Data

S'DLL :- FLAG A FLAG B FLAG —— R'DLL : FLAG A FLAG B FLAG

N.L. → A

S·DLL :- FLAG A ESC FLAG B FLAG —— R'DLL :- FLAG A ESC FLAG B FLAG

N.L. → A FLAG B

after character stuffing.

N.L. :- A ESC FLAG

D.L.L. :- FLAG A (ESC) ESC (ESC) FLAG P FLAG

↑ character stuffing ↑

Drawback :- for every FLAG one (ESC) should be added so overhead size increases.

* "Bit stuffing" :-

N.L. :- A FLAG B

A ⇒ 65
01000001
B ⇒ 66
01000010
FLAG ⇒ 01111110

D.L.L. :- FLAG A (ESC) FLAG B FLAG

01111110 01000001 01111110 01000010 01111110

01111101 0

↑ stuffed ~~data~~ bit → to understand flag.

* :- after 5, 1's 0 will be stuffed. ↑ to identify flag in data.

(2) FLAG = 01111110

Data at N.L. = 01111101111110

Data at DLL after Bit stuffing :- 01111101001111100

↑ stuffed bit ↑ stuffed bit

(3) N.L. = 01111101111110

FLAG = 01111110 ← after 4, 1's 0 will be stuffed to identify flag.

Data at DLL after bit stuffing :-

01111010111100

↑ stuffed bit ↑ stuffed bit

FLAG = 1000001 ← stuff 1 after 4 zeros

Data at DLL after Bit stuffing :- 10000101 000011
↳ stuffed bits

Dated :- 13/06/2017

① → ② ↳ Serial Transfer

→ (i) Synchronous serial Transmission
→ (ii) Asynchronous serial transmission.

(i) "Synchronous Serial Transfer" -

(1) In Synchronous serial transmissions, If three 8 bit sync characters are included in 30 eight bit information characters: the Bandwidth of the channel is 1200 bps. then what is the data rate of recievers :?

3 × 8 ← 3 8 bit sync chars — included → 30 eight bit information chars ⇒ 30 × 8

24 sync bits ⟶ 240 bits

Bandwidth = 1200 bits/sec ← 1200 bits

$\frac{24 * 1200}{240}$

⇒ 120 sync bits

data rate at reciever = 1200 − 120 bits/sec. (sync bits are not recieved by recievers, these are for alert)

= 1080 bits/sec. = $\frac{1080}{8}$ char/sec. = 135 char/sec

★ In Synchronous serial transmission sync bits are added for group of characters

★ Sync bits are only to alert the reciever that the data is coming.

(ii) "Asynchronous Serial transfer" :-

(2) In Asynchronous serial transmission, 1 start bit, 2 parity bits ~~+ 1 char~~, 1 stop bit are added for a character. Bandwidth of the channel is 1200 bits/sec then what is the data rate of reciever

:- 1 start bit + 2 parity bits + 1 char + 1 stop bit → = 1 char

:- Bandwidth = 1200 bits/sec.

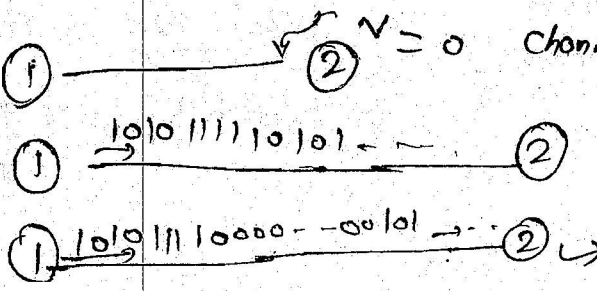
i.e. data rate at reciever = $\frac{1200 \text{ bits}}{1+8+2+1}$ → Part of character = 100 char/sec. = 100 char/sec.

Notes:-

★ In Asynchronous serial transmission, Extra bits are added for every character

★ These extra bits are treated as part of data.

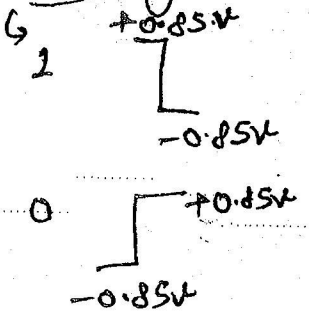
1 ⇒ +5V
0 ⇒ 0V

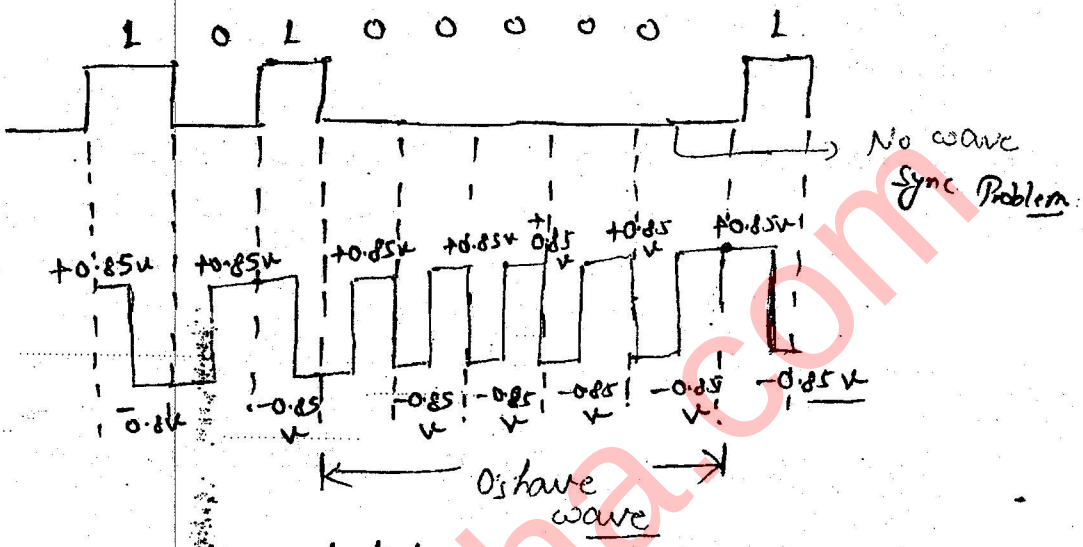


Signal → (1) → (2) High direct current (low)

"Encoding" :-

"Manchester Encoding" :-





1 → High to Low, 0 → low to high

Note :- The Advantage of menchester encoding is :-

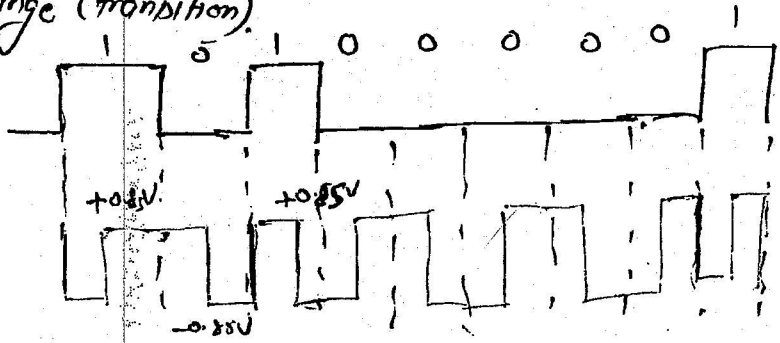
★ 1) To provide syncronization.

★ 2) Eliminating the high D.C (Direct current) components.

:- Differential menchester encoding :-

↳ 1 ⇒ transition (change)

0 ⇒ no change (transition)



The Advantage of differential menchester encoding :-

1) To provide synchronization

2) eliminating the High D.C component.