LOGIC DESIGN

12 February 2024 12:39 AM

XOR $A \oplus B = A\bar{B} + \bar{A}B$

XNOR $A \odot B = \overline{A \oplus B} = AB + \bar{A}\bar{B}$

$A + AB = A$

$A + \bar{A}B = (A + B)$

$(A+C)(A+B) = A + BC$

$(A+B)(A+\bar{B}) = A$

Consensous Law $F = AC + B\bar{C}$
$= AC + B\bar{C} + AB$

De Morgan's Law $\overline{AB} = \bar{A} + \bar{B}$
$\overline{A+B} = \bar{A}\,\bar{B}$

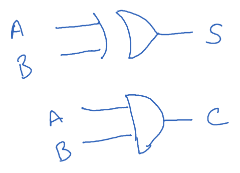
Minterm (m) = SOP Σ

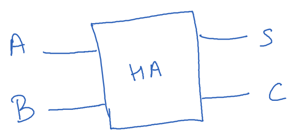
Maxterm (M) = POS Π

Half Adder

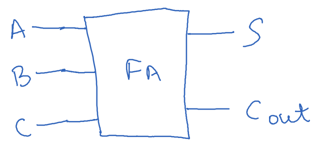
$S = A \oplus B$

$C = AB$





Full Adder



$S = \Sigma_m(1,2,4,7)$

$C_{out} = \Sigma_m(3,5,6,7)$

$S = A \oplus B \oplus C$

$C_{out} = C(A \oplus B) + AB$
$= AC + BC + AB$

K- Maps

AB\CD	00	01	11	10
00				
01				
11				
10				

A\BC	00	01	10	11
0				
1				

essential prime implicants
↳ atleast one element is not common.

* pair in powers of 2

* d ≡ don't care, dummy

Parallel Adder / Ripple Carry Adder RCA

→ n-bit Add^n

→ Variable I/P size

Drawbacks:

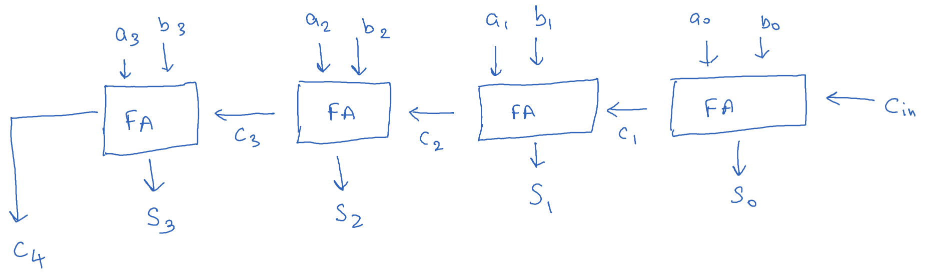
→ Area of design = $n A_{FA}$

→ Delay of design

Delay $T = (n-1) T_c + T_s$ → Delay of producing final sum.

T_c ↓ Delay of carry being rippled through previous stages

Area: if $C_{in} \neq 0$: $n A_{FA}$ if $C_{in} = 0$: $(n-1) A_{FA} + A_{HA}$



$S = S_3 S_2 S_1 S_0$

$C_{out} = C_4$

Carry Look Ahead Adders CLA

$C_{i+1} = a_i b_i + (a_i \oplus b_i) c_i = G_i \oplus P_i C_i$ $G_i = a_i b_i$

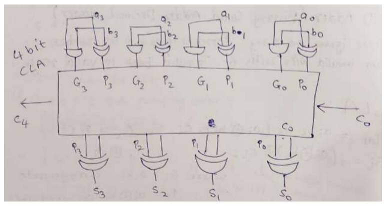
$S_i = a_i \oplus b_i \oplus c_i = P_i \oplus C_i$ $P_i = a_i \oplus b_i$

Adv

→ Can pre calculate carry's.
Hence faster than RCA

Disadv.

→ As n increases, Complexity increases.
only 4 bit CLA available



Binary Coded Decimal Adder.

→ Adds two binary number

→ Each digit is represented in a 4 bit number

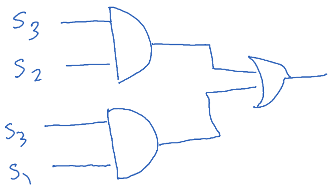
eg: $529 = \underbrace{5}\ \underbrace{2}\ \underbrace{9} =$ 0101 0010 1001

→ Each digit is represented in a 4 bit number

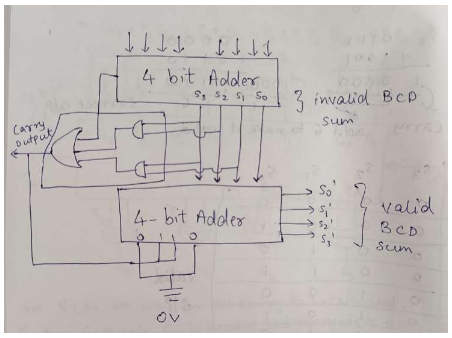
eg: 529 = $\underbrace{5}_{0101}$ $\underbrace{2}_{0010}$ $\underbrace{9}_{1001}$ = 0101 0010 1001

→ Adds each digit (2+9 etc) . If sum exceeds 9, then add 6 more to shift.
↓
ie carry = 1

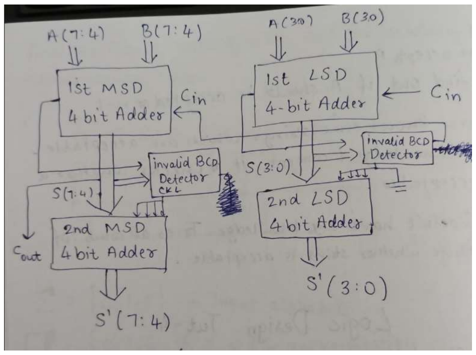
a digit is represent as $S_3 S_2 S_1 S_0$ if $S_3 S_1 = 1$ or $S_3 S_2 = 1$ } invalid O/P Add 6.



invalid O/P detector



for 1 digit

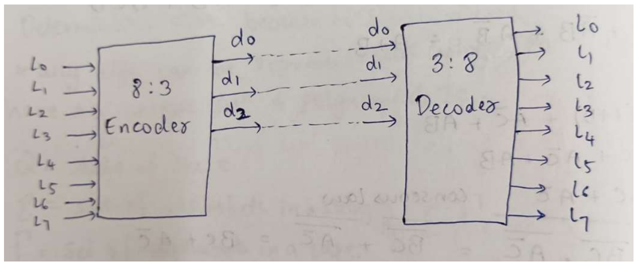


for 2 digit BCD Adder

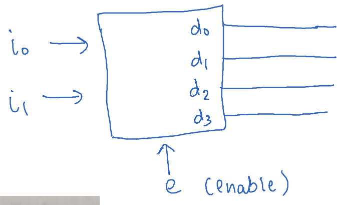
Digital Transmission (Encoder Decoder)

Encoder $(2^n : n)$

Decoder $(n : 2^n)$

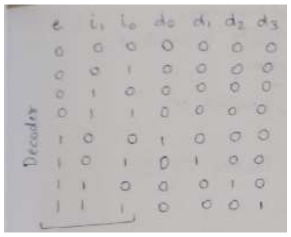


2: 4 Decoder



e = 0 decoder circuit is off

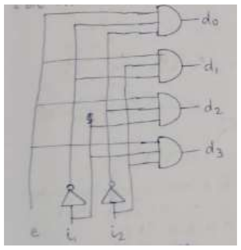
e = 1 decoder circuit in on.



e	i_1	i_0	d_0	d_1	d_2	d_3
0	0	0	0	0	0	0
0	0	1	0	0	0	0
0	1	0	0	0	0	0
0	1	1	0	0	0	0
1	0	0	1	0	0	0
1	0	1	0	1	0	0
1	1	0	0	0	1	0
1	1	1	0	0	0	1

Decoder

2 bit I/P



Priority Encoder

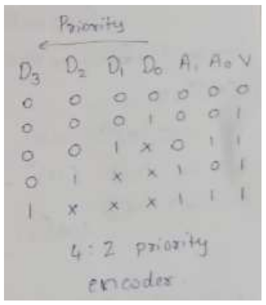
→ Each port assigned a priority.

→ Standard priority : $D_3 > D_2 > D_1 > D_0$

$A_0 = \overline{D_3}\,\overline{D_2}\,D_1 + D_3$

$A_1 = \overline{D_3}\,D_2 + D_3 = D_3 + D_2$

$V = D_3 + D_2 + D_1 + D_0$



Priority ←

D_3	D_2	D_1	D_0	A_1	A_0	V
0	0	0	0	0	0	0
0	0	0	1	0	0	1
0	0	1	x	0	1	1
0	1	x	x	1	0	1
1	x	x	x	1	1	1

4 : 2 priority encoder

x = don't care

Applications :

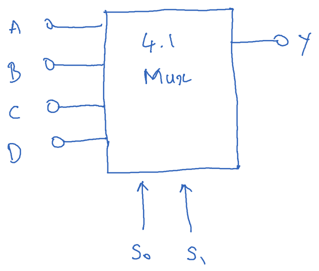
→ fiber optics : encoded at one end, decoded at other

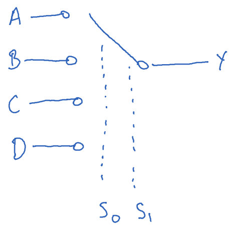
→ computer recieves signals from many devices. The encoder has an inbuilt priority for requests from these devices, and depending on priority, it processes the request.

interrupt : computer currently working on a tasks leaves it for higher priority task

Multiplexer (Mux)

→ Digital circuit with 2^n input lines & 1 O/P



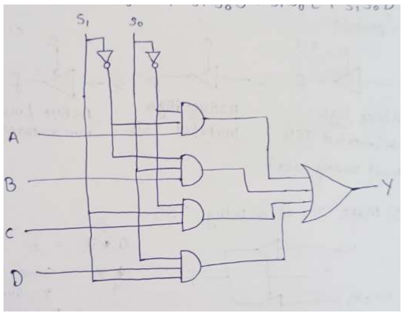


S_1	S_0	Y
0	0	A
0	1	B
1	0	C
1	1	D

$Y = A\,\overline{S_1}\,\overline{S_0} + B\,\overline{S_1}\,S_0 + C\,S_1\,\overline{S_0} + D\,S_1\,S_0$

primary inputs. A, B, C, D, S_1, S_0

↳ come directly from user & not some component



Multiplexer is used when there are resource constraints over system is having too much overhead.

Tri- State Buffer (TSB)

→ 1 I/P (x) 1 O/P (f) 1 Control line (e)

Z = high impedance (open circuit)

X = don't care (conflict)

e	x	f
0	1	Z
0	0	Z
1	0	0
1	1	1

if e = 0 f = Z

if e = 1 f = x

1	0	0
1	1	1

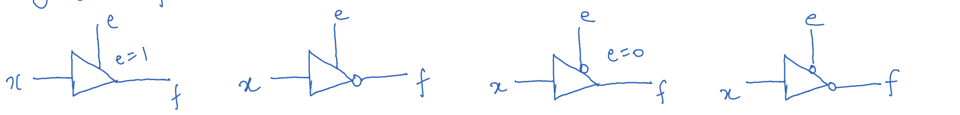
Z = high impedance (open circuit)

X = don't care (conflict)

u = uninitialised I/P values

$\therefore$ 0, 1, z, x, u $\rightarrow$ States

Symbols of TSB:

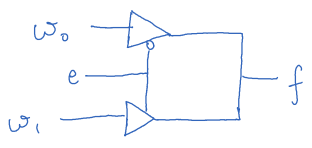


Active High Non Inverted TSB (mostly used) — Active High Inverted TSB — Active Low Non Inverted TSB — Active Low Inverted TSB

0 & Z = Z

1 & Z = 1

1 & 0 = X



2:1 Mux

Parity Generator Function

A	B	C	O/P P
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0

$\rightarrow$ code word

data word

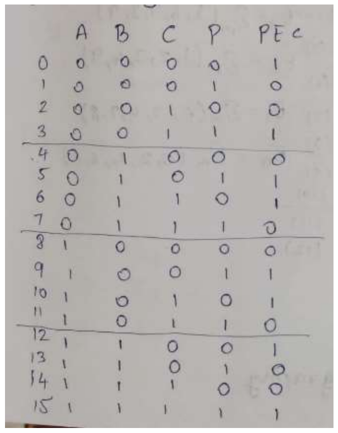
Adding parity bit to code word, such that the code words has odd number of 1's.

$$P(A,B,C) = \sum_m (0,3,5,6)$$

$$= \overline{A \oplus B \oplus C}$$

This is used in bit flip detection but not rectification during transmission.
It can only detect a single bit flip not multiple

Odd Parity Generator



	A	B	C	P	PEC
0	0	0	0	0	1
1	0	0	0	1	0
2	0	0	1	0	0
3	0	0	1	1	1
4	0	1	0	0	0
5	0	1	0	1	1
6	0	1	1	0	1
7	0	1	1	1	0
8	1	0	0	0	0
9	1	0	0	1	1
10	1	0	1	0	1
11	1	0	1	1	0
12	1	1	0	0	1
13	1	1	0	1	0
14	1	1	1	0	0
15	1	1	1	1	1

checks the 4-bit codeword, to check for odd number of 1's

if odd # 1's PEC = 0

if even # 1's PEC = 1

$$PEC = \sum_m (0,3,5,6,9,10,12,15)$$

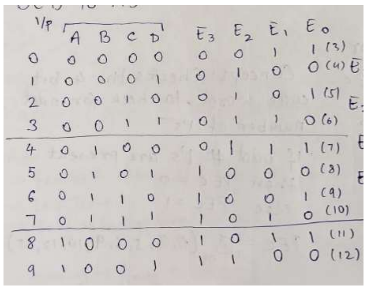
$$= \overline{A \oplus B \oplus C \oplus P}$$

SEU $\rightarrow$ We send an extra bit to realise if bit flip occured during transmission.

Detects but doesn't rectify. Can't detect multiple bit flips.

SEC → We send an [illegible]
transmission.
Detects but doesn't rectify. Can't detect multiple bit flips.

BCD to XS-3 Convertor



I/P	A	B	C	D	E_3	E_2	E_1	E_0
0	0	0	0	0	0	0	1	1 (3)
1	0	0	0	1	0	1	0	0 (4)
2	0	0	1	0	0	1	0	1 (5)
3	0	0	1	1	0	1	1	0 (6)
4	0	1	0	0	0	1	1	1 (7)
5	0	1	0	1	1	0	0	0 (8)
6	0	1	1	0	1	0	0	1 (9)
7	0	1	1	1	1	0	1	0 (10)
8	1	0	0	0	1	0	1	1 (11)
9	1	0	0	1	1	1	0	0 (12)

$E_3 = \sum m(5,6,7,8,9)$

$E_2 = \sum m(1,2,3,4,9)$

$E_1 = \sum m(0,3,4,7,8)$

$E_0 = \sum m(0,2,4,6,8)$

Application
→ Cryptography

Implementation of Boolean function using MUX

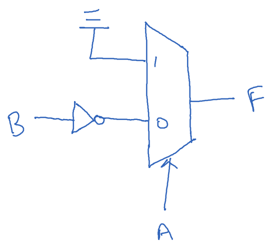
1) NOR Gate

A	B	F
1	1	0
1	0	0
0	1	0
0	0	1

NOR

A	F
0	$\overline{B}$
1	0

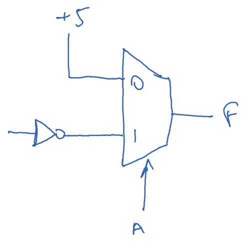
MUX



2) Implementing NAND

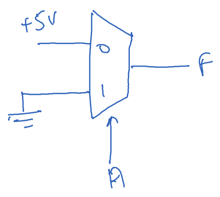
A	B	F
1	1	0
1	0	1
0	1	1
0	0	1

A	F
0	1
1	$\overline{B}$



3) Invertor

A	F
0	1
1	0

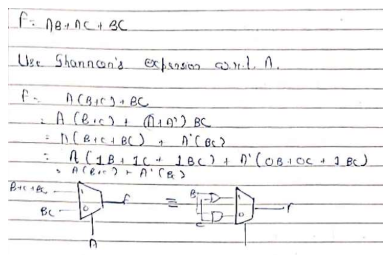


Shanon's Expression

$F = x F_x + \bar{x} \overline{F_x}$

F_x = +ve shanon factor @ $x=1$

$\overline{F_x}$ = -ve shanon factor @ $x=0$

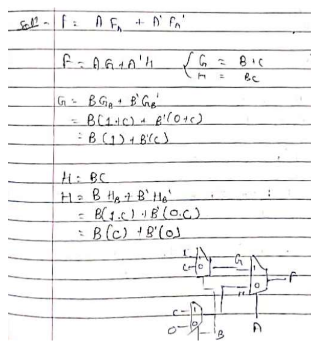


$F = AB + AC + BC$

Use Shannon's expansion w.r.t. A.

$F = A(B+C) + BC$
$= A(B+C) + (A+A')BC$
$= A(B+C+BC) + A'(BC)$
$= A(1B + 1C + 1BC) + A'(0B + 0C + 1BC)$
$= A(B+C) + A'(BC)$

B+C+BC
BC
1
0
F
A
≡
B
C
1
0
F



Sol^n - $F = A F_A + A' F_A'$

$F = AG + A'H \quad \begin{cases} G = B + C \\ H = BC \end{cases}$

$G = B G_B + B' G_B'$
$= B(1+C) + B'(0+C)$
$= B(1) + B'(C)$

$H = BC$
$H = B H_B + B' H_B'$
$= B(1.C) + B'(0.C)$
$= B(C) + B'(0)$

1
C
1
0
G
1
0
F
H
C
0
1
0
B
A

Comparator

1) 1-bit Comparator

A_i	B_i	$\overset{A=B}{E_i}$	$\overset{A<B}{L_i}$	$\overset{A>B}{G_i}$
0	0	1	0	0
1	1	1	0	0
1	0	0	0	1
0	1	0	1	0

$$E_i = A_i \odot B_i = A_i B_i + \overline{A_i}\,\overline{B_i}$$

$$L_i = \overline{A_i}\, B_i$$

$$G_i = A_i\, \overline{B_i}$$

2) 4-Bit Comparator.

$A = A_3 A_2 A_1 A_0$

$B = B_3 B_2 B_1 B_0$

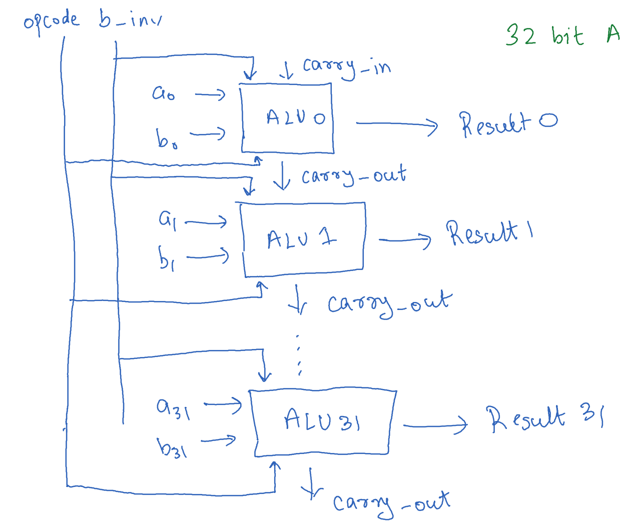
$$E = (A_3 \odot B_3)(A_2 \odot B_2)(A_1 \odot B_1)(A_0 \odot B_0)$$

$$L = \overline{A_3} B_3 + (A_3 \odot B_3)\overline{A_2} B_2 + (A_3 \odot B_3)(A_2 \odot B_2)\overline{A_1} B_1 + (A_3 \odot B_3)(A_2 \odot B_2)(A_1 \odot B_1)\overline{A_0} B_0$$

$$G = A_3 \overline{B_3} + (A_3 \odot B_3) A_2 \overline{B_2} + (A_3 \odot B_3)(A_2 \odot B_2) A_1 \overline{B_1} + (A_3 \odot B_3)(A_2 \odot B_2)(A_1 \odot B_1) A_0 \overline{B_0}$$

Arithematic Logic Unit (ALU)

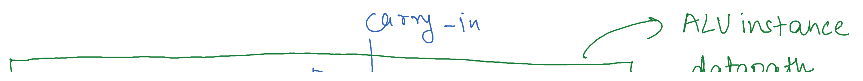
32 bit ALU



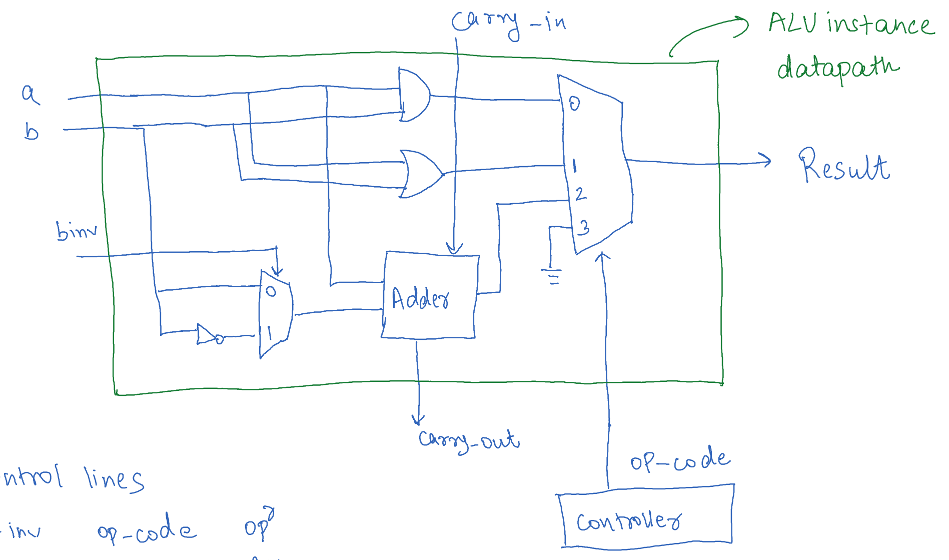
Controller generates the op-code to synchronize ALU

Datapath → internal mech of ALU, responsible for all computations.

Controller is responsible for providing timming info ie at what time which signals are activated/deactivated.



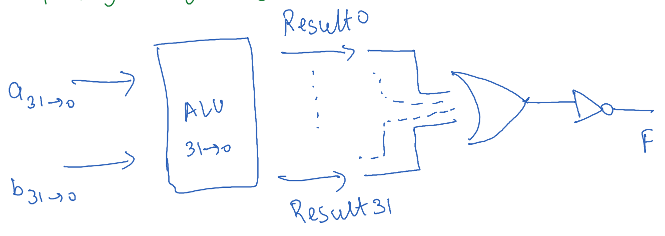
are activated/deactivated.



control lines

b-inv	op-code	op?
0	0 0	AND
0	0 1	OR
0	1 0	ADD
1	1 0	SUBTRACT

Test for equality



if A − B = 0
F = 1
else
F = 0

op-code = 10

b_inv = 1

Hamming Code

for a 4-bit data word, we have 3 parity bits . ∴ total 7 bit codeword

1	2	3	4	5	6	7
P_1	P_2	D_1	P_3	D_2	D_3	D_4

parity bits = 2^i position $i = 0, 1, 2 \ldots$

rest is data word bit.

Procedure:

1) Mark all bits with position 2^i as parity bits
2) Mark all remaining position as data word bits
3) Each parity bit calc parity of some bits

Position 1 P1 = Check 1 bit , skip 1-bit , Check 1-bit, skip 1 bit ....

Position 2 P2 = Check 2 bit , skip 2-bit , Check 2-bit, skip 2 bit ....

Position 4 P4 = Check 4 bit , skip 4-bit , Check 4-bit, skip 4 bit ....

Position 8 P8 = Check 8 bit , skip 8-bit , Check 8-bit, skip 8 bit ....

Starting at P_i of the codeword.

4) Set the parity bit to 1 if there are odd # of 1
(generate even parity)

This is how we create the hamming code for generating

Now for detecting, check in reverse order, & create new parity bits.
& combine them to detect the flipped position.

~~eg~~ transmitted C.W. = $0111001010\boxed{0}10$ at P_{10} corruption
recieved C.W. = $0111001010\boxed{1}10$

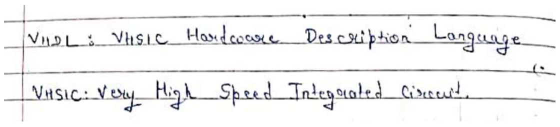
Check for even parity at 8,9,10,11
fails due to ODD #1 $x_1 = 1$

Check for even parity at 4,5,6,7, 12
passes due to EVEN #1 $x_2 = 0$

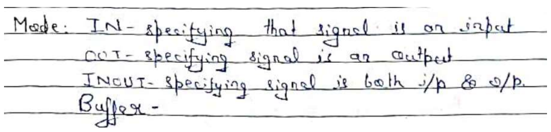
Check for even parity at 2,3, 6, 7, 11
fails due to ODD #1 $x_3 = 0$

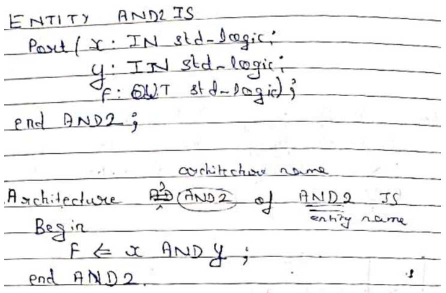
Check for even parity at 1, 3, 5,7,11
passes due to EVEN #1 $x_4 = 0$

$\therefore$ corrupted bit = $x_1 x_2 x_3 x_4$ = $\underbrace{1010}_{\text{binary}} = \underbrace{10}_{\text{decimal}}$

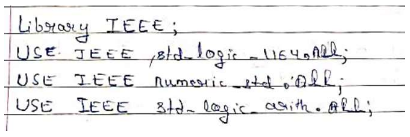

VHDL : VHSIC Hardware Description Language

VHSIC: Very High Speed Integrated Circuit.

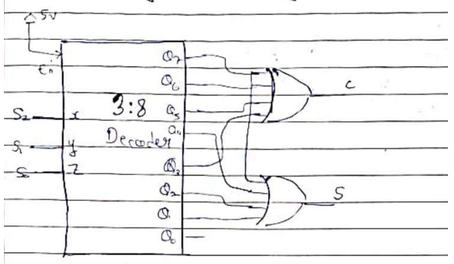

Mode: IN - specifying that signal is an input
OUT - specifying signal is an output
INOUT - specifying signal is both i/p & o/p.
Buffer -


```
ENTITY  AND2 IS
  Port ( x : IN std-logic;
         y : IN std-logic;
         F : OUT std-logic);
end AND2 ;

Architecture ~~AD~~ AND2 of AND2 IS
  Begin
     F <= x AND y ;
  end AND2.
```
architechture name
entity name


```
Library IEEE;
USE IEEE ,std_logic_1164.All;
USE IEEE numeric_std.All;
USE IEEE std_logic_arith.All;
```

Implementing Adder Using Decoder:-

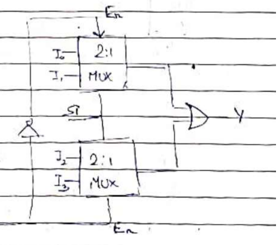


$C = \Sigma_m(3,5,6,7)$

$S = \Sigma_m(1,2,4,7)$

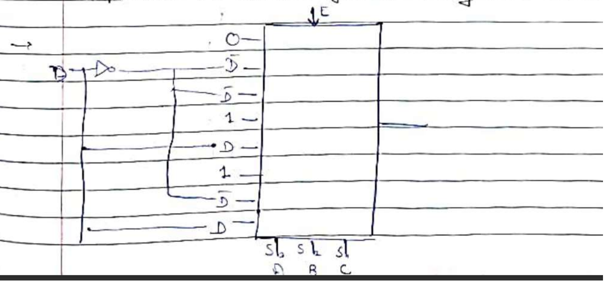
Designing 4:1 MUX :-

S_0	S_1	Y
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3



Ques: $f(A,B,C,D) = \Sigma_m(2,4,6,7,9,10,11,12,15)$

Implement the boolean function using 8:1 MUX.



- ➢ A digital circuit can be implemented :
- (1) as an application specific integrated circuit (ASIC)
- (2) on a programmable device like field programmable gate array (FPGA)

➢ A **field-programmable gate array (FPGA)** is an integrated circuit designed to be configured by a customer or a designer after manufacturing – hence the term field-programmable.

➢ A **configurable logic block (CLB)** is the basic repeating logic resource on an FPGA. When linked together by routing resources, the components in CLBs execute complex logic functions, implement memory functions, and synchronize code on the FPGA.

➢ CLBs contain smaller components, including flip-flops, look-up tables (LUTs), and multiplexers.

Synthesis is used to create a netlist from HDL (e.g. VHDL or Verilog). In electronic design, a netlist is a description of the connectivity of an electronic circuit.

A **bitstream** is a file that contains the configuration information for an FPGA. It is also known as a bit file or programming file because by streaming it to the FPGAs configuration port, we can program the FPGA. The bitstream is a binary format, although sometimes it's stored as a human-readable hex file.

The **behavioral simulation** is performed at RTL-level. It is typically performed to verify code syntax, and to confirm that the code is functioning as intended.

Functional and timing simulations are performed post-synthesis or post-implementation. After synthesis or implementation, RTL is transformed to structural netlist, in which the lowest level of hierarchy consists of primitives and macro primitives.

Different models of architecture

Behavioural model

Data flow model

Structural model

- **Behavioural model**

-> Describes how the output is derived from the inputs using structured statements. Behavioral modelling executes statements sequentially.
They are written inside a process statement. Statements like if-else , switch case, loops are part of behavioral modelling.

Although we never use looping statements while programming hardware as it is difficult to implement on board.

- **Data flow model**

-> Describes how the data flows from the inputs to the output most often using NOT, AND and OR operations.

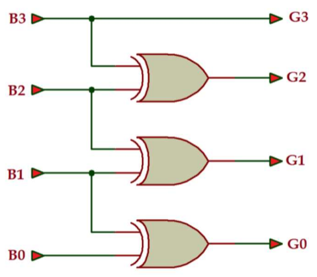
Ex: y <= a & b ;

Statements are executed concurrently.

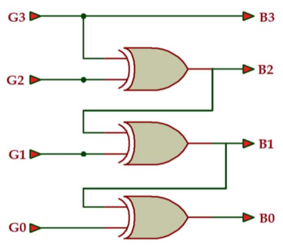
- **Structural model**

-> Structural modelling uses logic diagrams. The concept is similar to functions. Here we create components of each logic gate which is used multiple times in the code.

The 4-bit, binary-to-gray code converter



The gray-to-binary code converter circuit


G3
B3
B2
G2
G1
B1
G0
B0