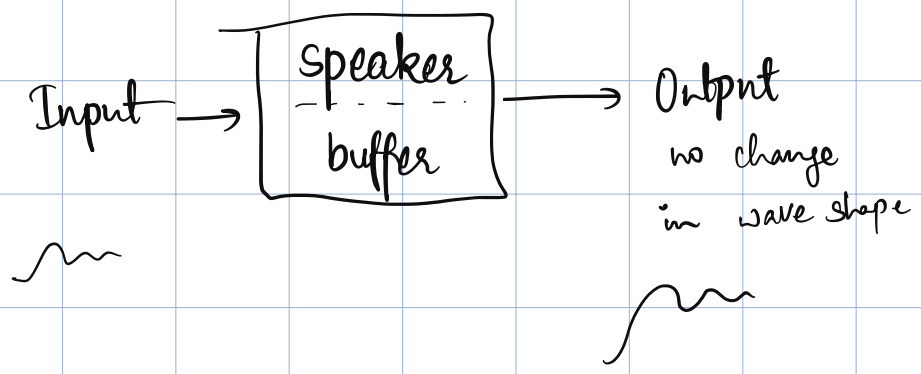
2024/09/09 - Digital Circuits - Week 07

2:00

Buffers

in general → under their presence, ideal systems do not have any effect

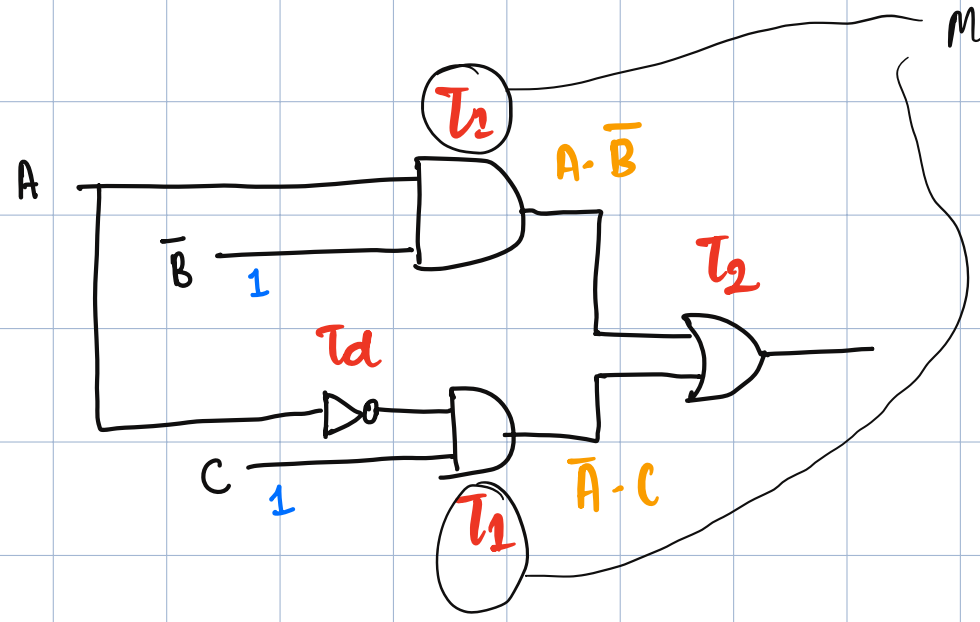
very common
aka repeaters



Solving hazard
① add consensus terms (BC in the previous example)
② add buffers

Timing hazards

- Static '1' hazard → present in SOP
- Static '0' hazard → present in POS



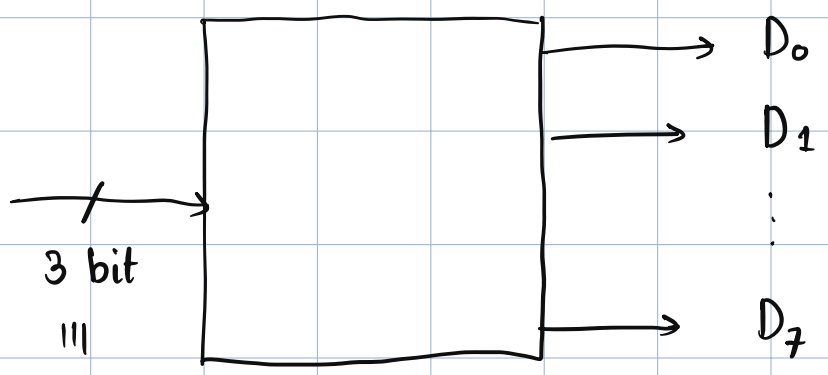
May not be same

dynamic hazards

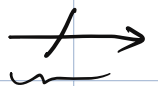
Combinational logic ⟶ O/p only depends on the present combination of input

e.g.: all circuits discussed till now.

Ex:-

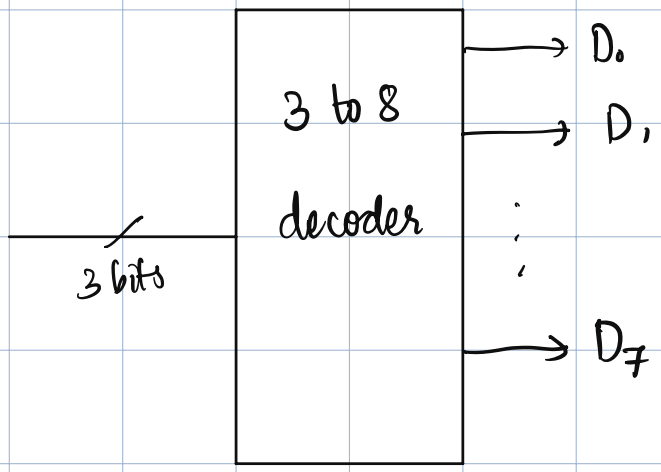


roll numbess for each roll no. ⟶ one bulb glows



A	B	C	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1

00 01 11 10



decoder → o/p and input have same information

— information content does not change, but it might be useful to expand 3 bits to 8 bits

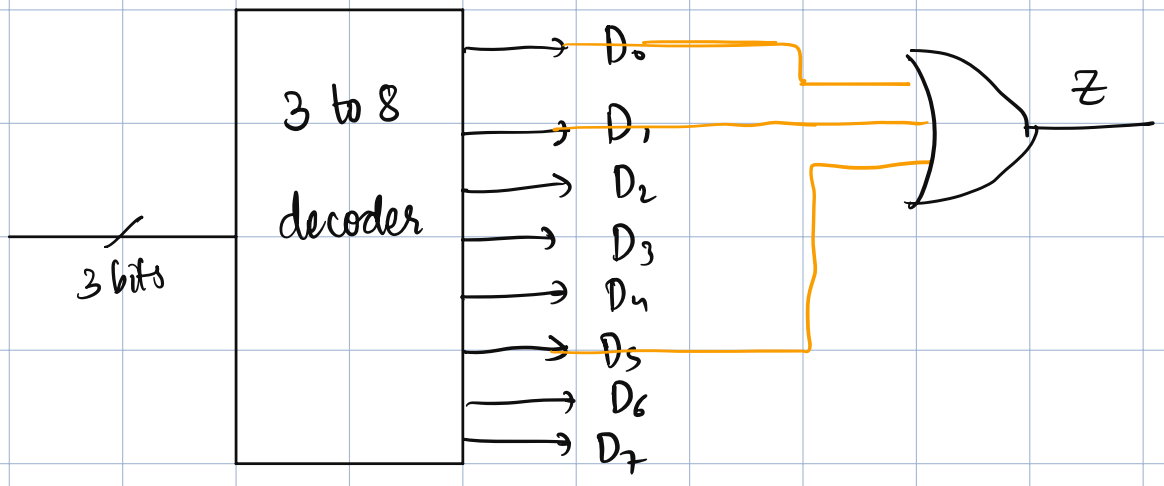
Decoder → OR to get a function

↳ outputs minterms

Why is it called decoder?

What are encoders?

reverse output

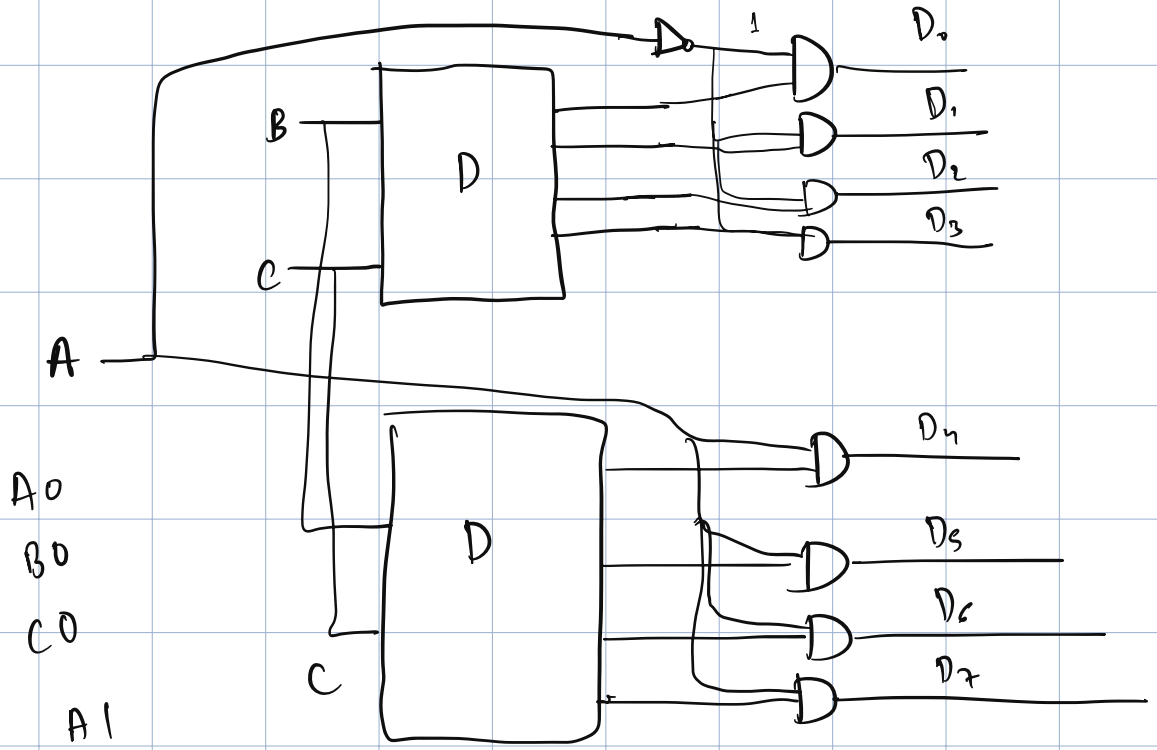


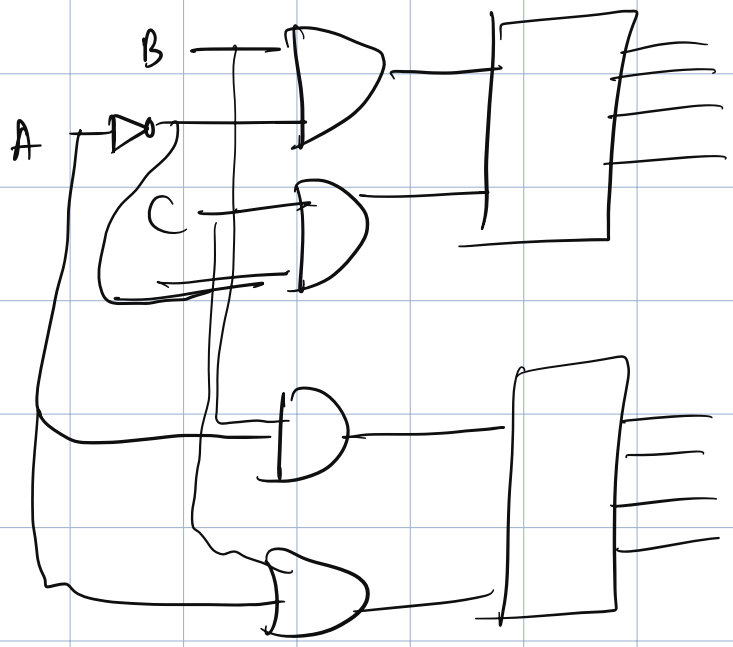
works for general case

extra logic, may not be always useful

A	B	C	Z
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	0

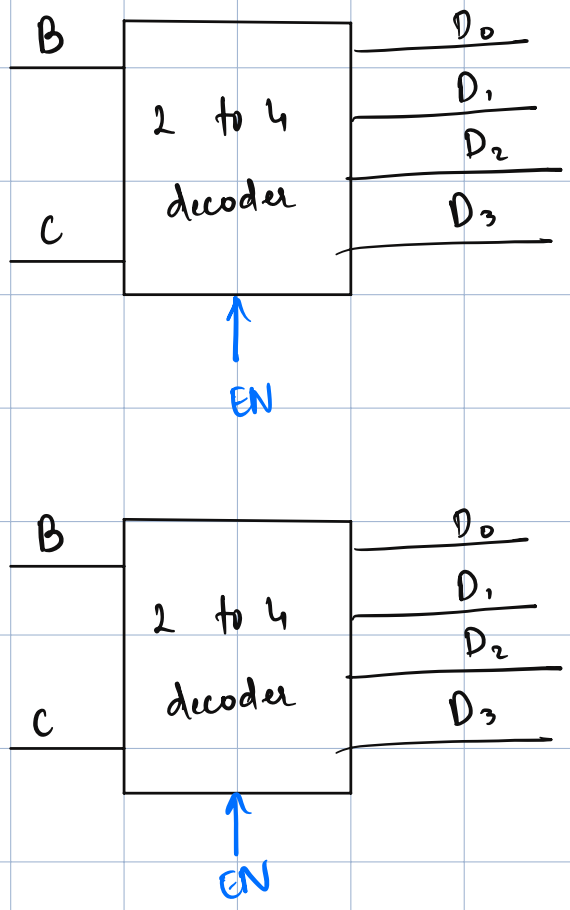
Ex: Design a 3 to 8 decoder using 2 to 4 decoders





HDL ⟶ edaplayground ⟶ Mini Project

2024|09|10



A	B	C	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7
0	0	0	1							
0	0	1		1						
0	1	0			1					
0	1	1				1				
1	0	0					1			
1	0	1						1		
1	1	0							1	
1	1	1								1

2 to 4 decoder

2 to 4 decoder

decoders have AND gate internally

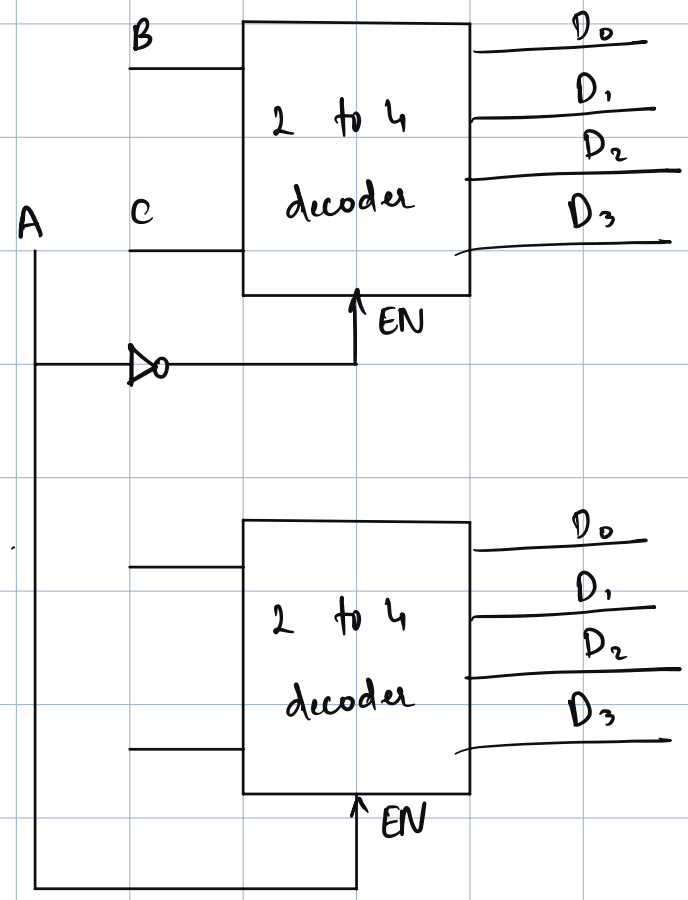
↓

connected to ENable input

EN	A	B	D_0	D_1	D_2	D_3
1	0	0	1	0	0	0
1	0	1	0	1	0	0
1	1	0	0	0	1	
1	1	1	0	0	0	1
0	X	X	0	0	0	0
0	X	X				
0	X	X				
0	X	X				

0 ...

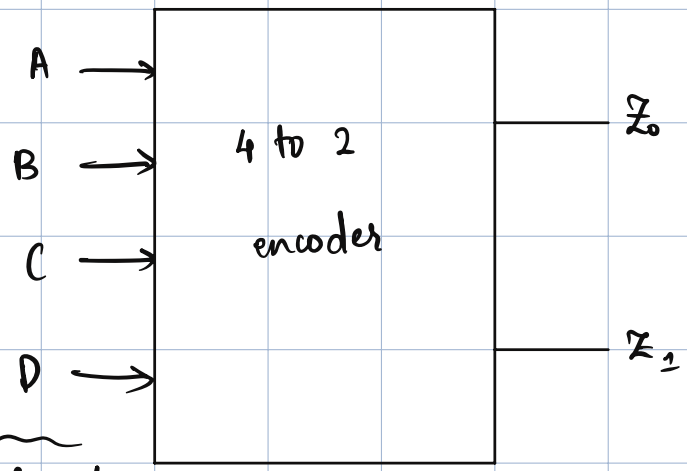
...



Decoders discussed here → aka binary decoder

Encoders

reverse



works only when on of the input is high

one hot input

	A	B	C	D	Z_0	Z_1
m_1	0	0	0	1	0	0
m_2	0	0	1	0	0	1
m_4	0	1	0	0	1	0
m_8	1	0	0	0	1	1

$$\bar{A}\bar{B}\bar{C}D + \bar{A}\bar{B}C\bar{D} + \bar{A}B\bar{C}\bar{D} + A\bar{B}\bar{C}\bar{D}$$

Priority encoder

→ A is highest in priority

if A is 1, ignore other inputs, output 11

Z_0

AB \ CD	$\bar{C}\bar{D}$	$\bar{C}D$	CD	$C\bar{D}$
$\bar{A}\bar{B}$	0	1	3	2
$\bar{A}B$	4	5	7	6
AB	12	13	15	14
$A\bar{B}$	8	9	11	10

simplicity comes with a tradeoff

Priority encoding

A	B	C	D	Z_0	Z_1
0	0	0	1	0	0
0	0	1	x	0	1
0	1	x	x	1	0
1	x	x	x	1	1

circuit that realises this = priority encoder

$Z_0 = A + B$

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

$= AB + A\bar{B} + \bar{A}B$

$Z_1 = A + C$

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

$= AB + A\bar{B} + CD + C\bar{D}$

Z_0

AB \ CD	$\bar{C}\bar{D}$	$\bar{C}D$	CD	$C\bar{D}$
$\bar{A}\bar{B}$	0 (0)	0 (1)	0 (3)	0 (2)
$\bar{A}B$	1 (4)	X (5)	X (7)	X (6)
AB	1 (12)	X (13)	X (15)	X (14)
$A\bar{B}$	1 (8)	X (9)	X (11)	X (10)

choose 1

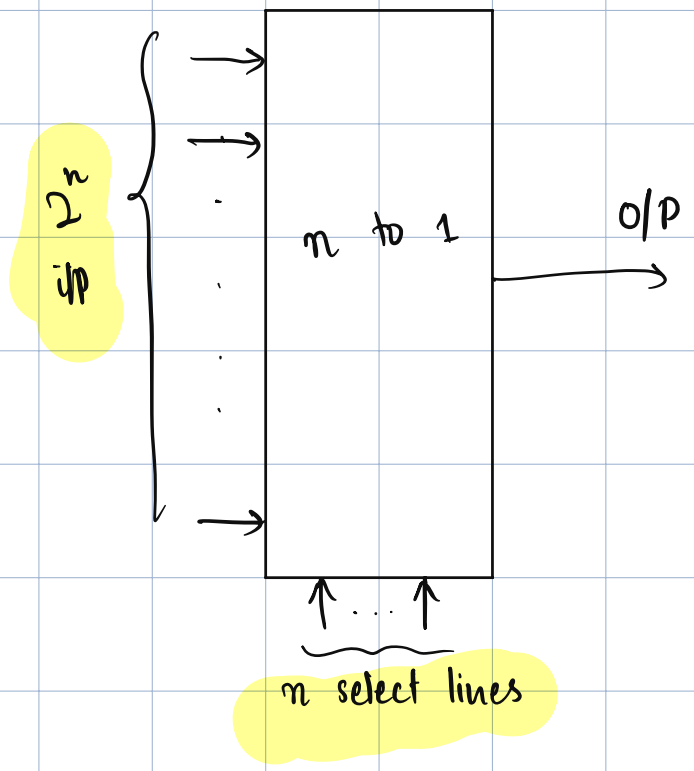
$$Z_0 = A + \bar{A} \cdot B$$

$$= A + B$$

Z_1

AB \ CD	$\bar{C}\bar{D}$	$\bar{C}D$	CD	$C\bar{D}$
$\bar{A}\bar{B}$	X (0)	0 (1)	X (3)	1 (2)
$\bar{A}B$	X (4)	X (5)	X (7)	X (6)
AB	X (12)	X (13)	X (15)	X (14)
$A\bar{B}$	1 (8)	X (9)	X (11)	X (10)

$$Z_1 = A + C$$



← 2^n to 1 multiplexer (MUX)

2^n to n
n to 2^n
Encoder, decoder

select lines needed for n outputs = $\log_2 n$

How to implement a multiplexer?

Next class

Tomorrow: demo of hardware description language. bring laptop

2024/09/12

C – functions

EDA → modules

Design

```
module all_gates (
    input A, B;
    output AND_out;
    output NOT_outA;
    :
);
:
:
endmodule
```

assign is a continuous statement

```
assign AND_out = A & B;
```

& → bit-wise and

```
assign NAND_out = -(A & B)
assign OR_out = A | B;
```

Testbench

```
module testbench;
    // declare input variables
    reg  A, B;

    // declare output variables to capture the output of the gates
    wire AND_out, NOR_out, . . . .

    all_gates uut {
        . A (A);
        . B (B);   . AND_out (.AND_out); . . . .
    }
```

logic ≡ auto

instantiation: positional, named

A (A): variable in design → test bench register

Test bench + design

↓

Verilog

Tools & simulators

↓

check EPWave

```
initial begin    // initial block
    $dumpfile ("gates.vcd");
    $dumpvar (testbench);
    $monitor (
```

only when variable chages

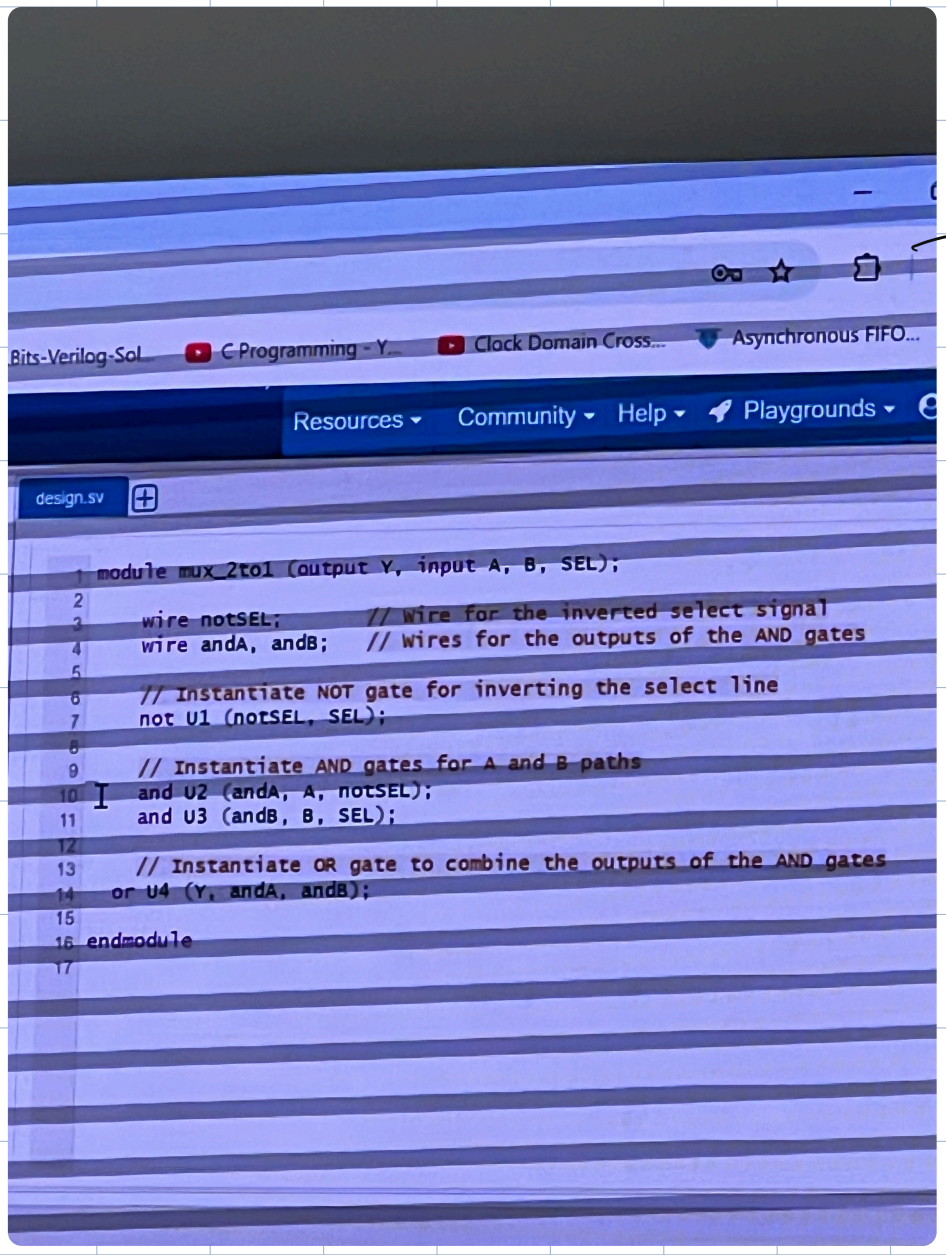
$display

// Test cases:

A = 0, B = 0, #10

#100 → 100 ns

$finish → end test



simpler

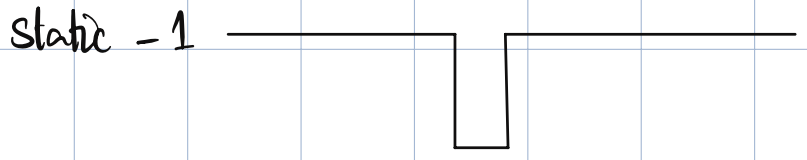
① behavioral

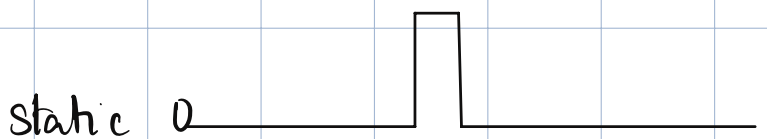
② structural

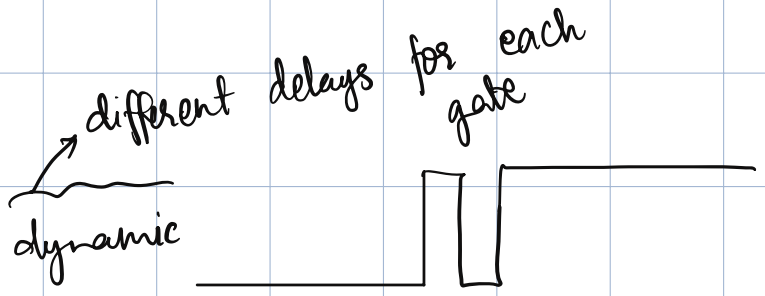
ⓑ assign

ⓐ always
control
driven

Timing hazards and glitches







Timing diagrams

Solution

* instead of only essential prime implicant, take all prime implicants } better

* use buffers

assign with delay:

```
assign #1 nc = ~c;
```

1ns delay

HDL bits

NPTEL
Introduction
modelling using
verilog till
22 lectures