

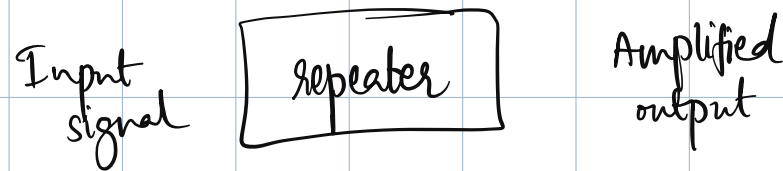
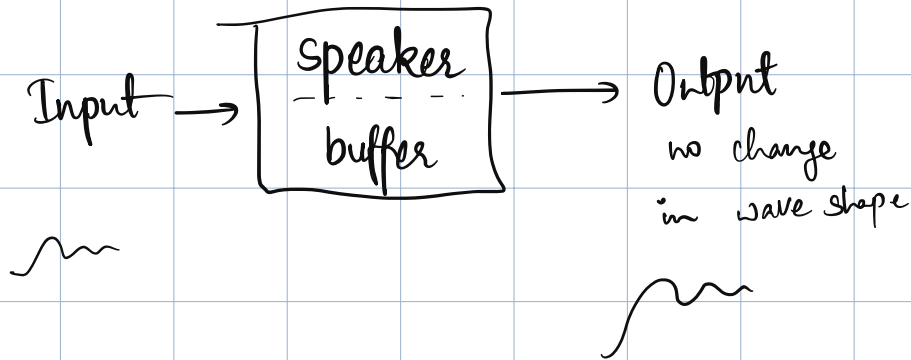
2024/09/09 - Digital Circuits - Week 07

Buffers

very common
aka repeaters

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

2:00



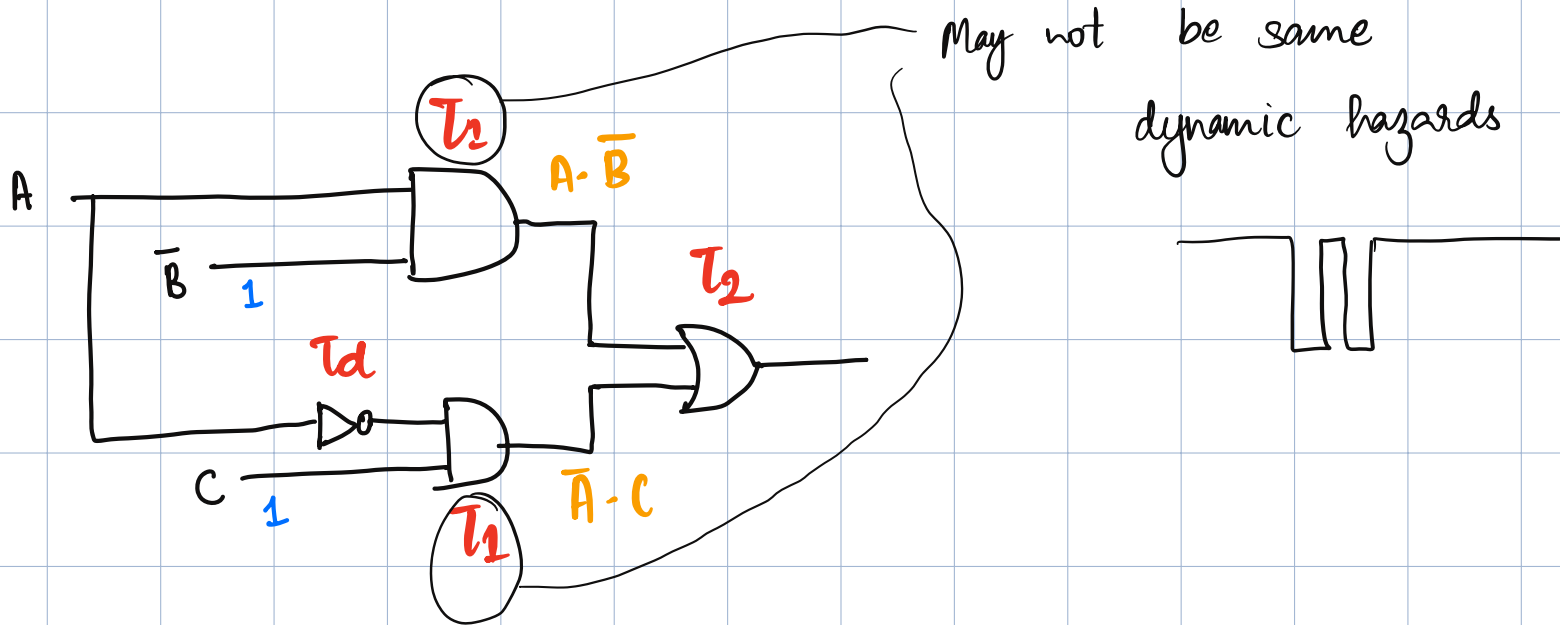
Solving hazard

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

Timing hazards

→ Static '1' hazard → present in SOP

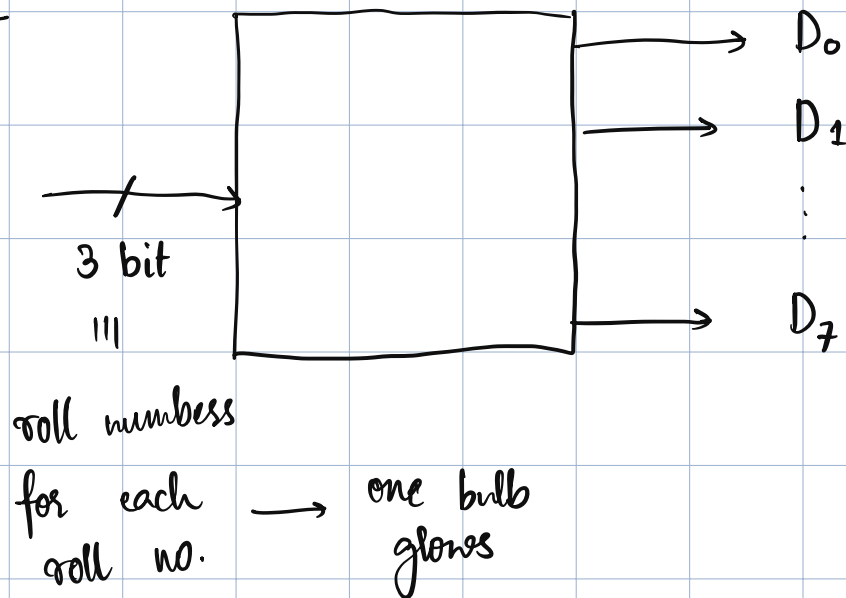
→ Static '0' hazard → present in POS



Combinational logic

→ o/p only depends on the
present combination of input
e.g.: all circuits discussed till now.

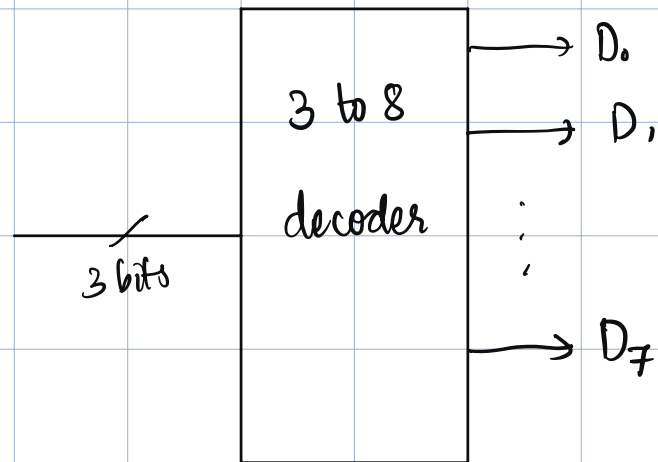
Ex:-



→
bus

	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 $\rightarrow$ o/p and input

have same information

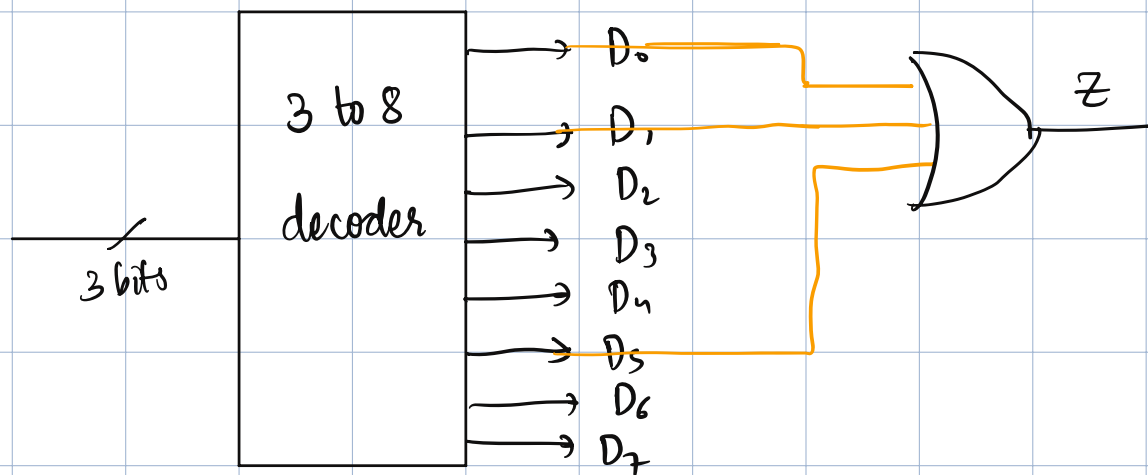
— information content does not change, but it might be useful to expand 3 bits to 8 bits

Decoder $\rightarrow$ OR to get a function
 $\searrow$ outputs minterms

Why is it called decoder?

What are
encoders?

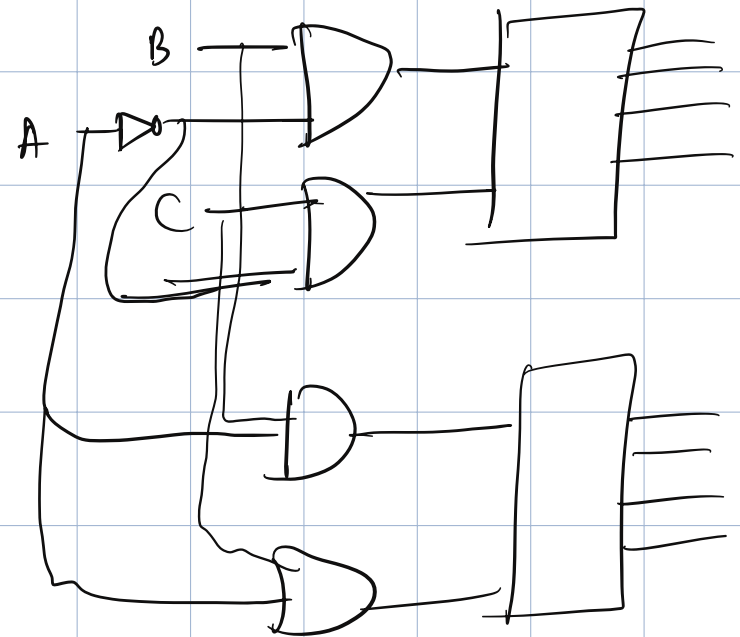
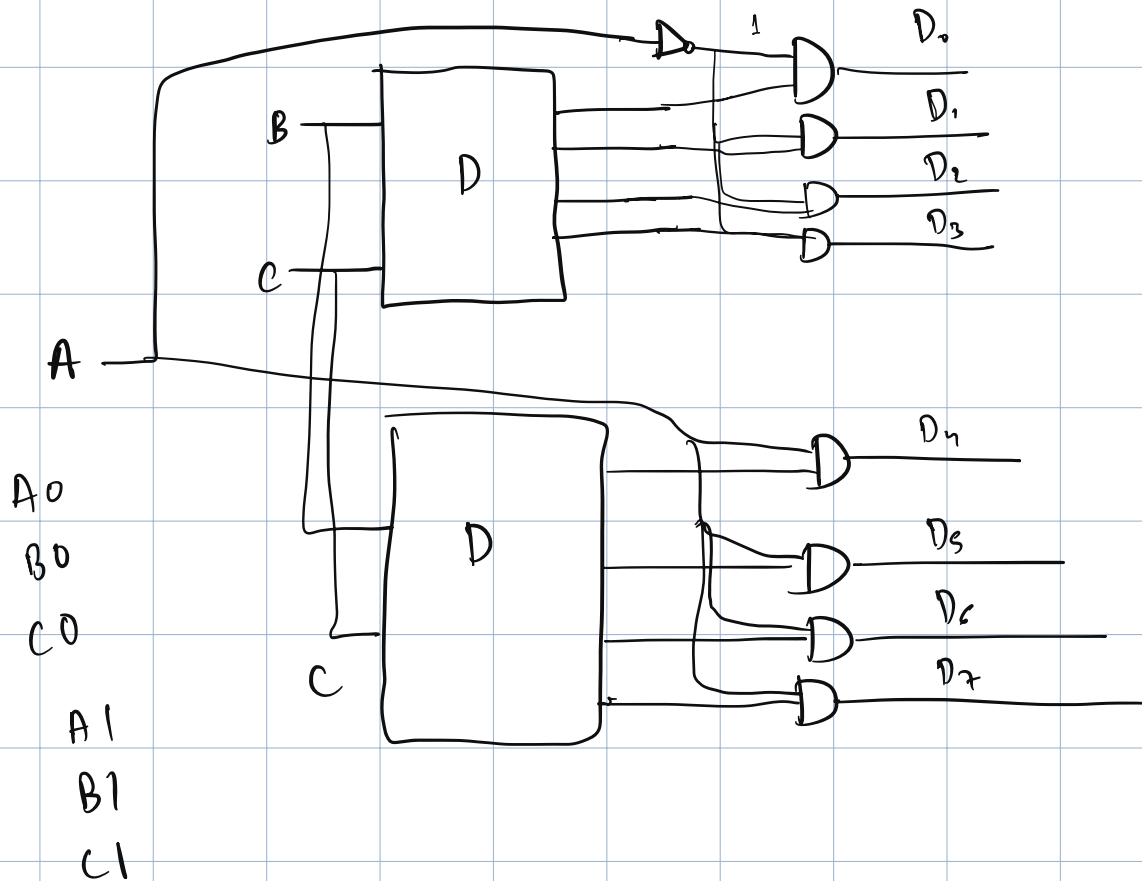
reverse output



works for general case ← } extra logic, may not be always useful

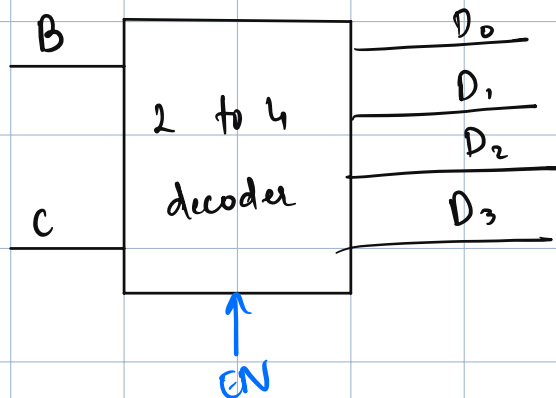
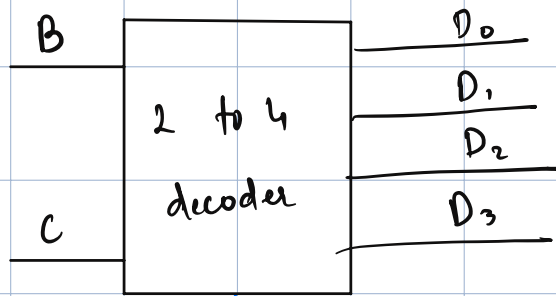
A	B	C	Z
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	0

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



HDL → edaplayground → Mini Project

2024/09/10



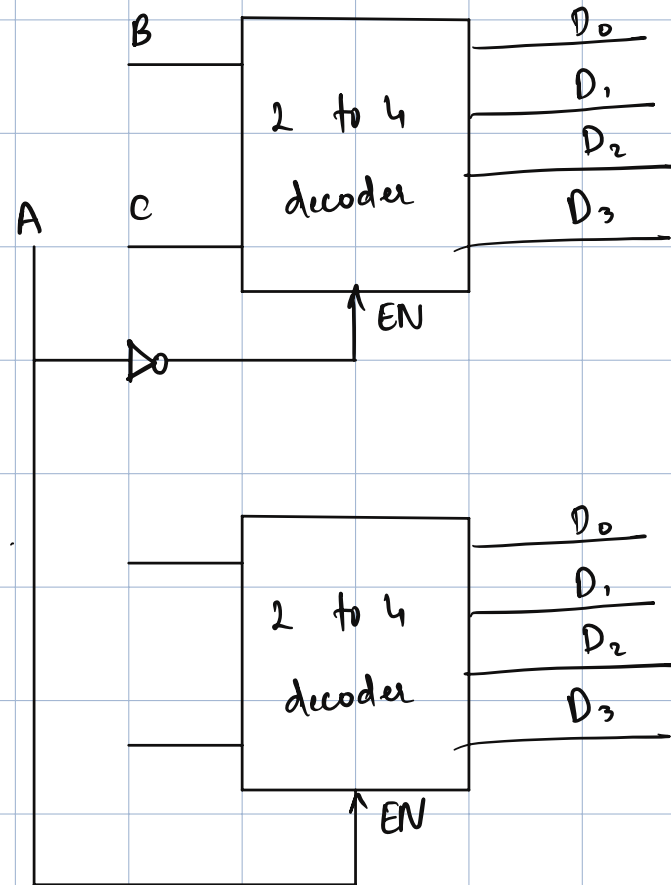
A	B	C	D ₀	D ₁	D ₂	D ₃	D ₄	D ₅	D ₆	D ₇
0	0	0	1							
0	0	1		1						
0	1	0			1					
0	1	1				1				
1	0	0					1			
1	0	1						1		
1	1	0							1	
1	1	1								1

2 to 4 decoder

2 to 4 decoder

decoders have AND gate
internally
↓
connected to
ENable input

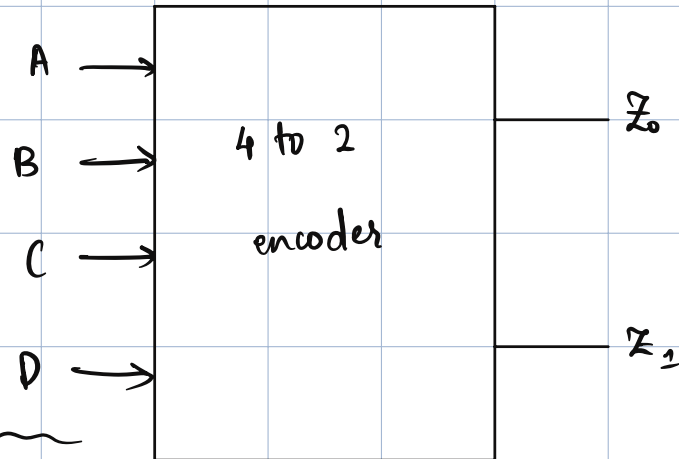
EN	A	B	D ₀	D ₁	D ₂	D ₃
1	0	0	1	0	0	0
1	0	1	0	1	0	0
1	1	0	0	0	1	
1	1	1	0	0	0	1
0	X	X	0	0	0	0 ...
0	X	X				
0	X	X				
0	X	X				



Decoders discussed here → aka binary decoders

Encoders

reverse



works only
when one
of the
input is
high

one hot
input

m_1

m_2

m_3

m_4

	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_3	0	1	0	0	1	0
m_4	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 encodes

→ A is highest in priority

if A is 1, ignore other
inputs, output 11

		Z_0			
		$\bar{C}\bar{D}$	$\bar{C}D$	CD	$C\bar{D}$
$\bar{A}\bar{B}$	$\bar{C}\bar{D}$	0	1	3	2
$\bar{A}B$	$\bar{C}\bar{D}$	4	5	7	6
AB	$\bar{C}\bar{D}$	12	13	15	14
$A\bar{B}$	$\bar{C}\bar{D}$	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

$$Z_0 = A + B = A(B + \bar{B}) + B(\bar{A} + A)$$

$$Z_1 = A + C = AB + A\bar{B} + \bar{A}B$$

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

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

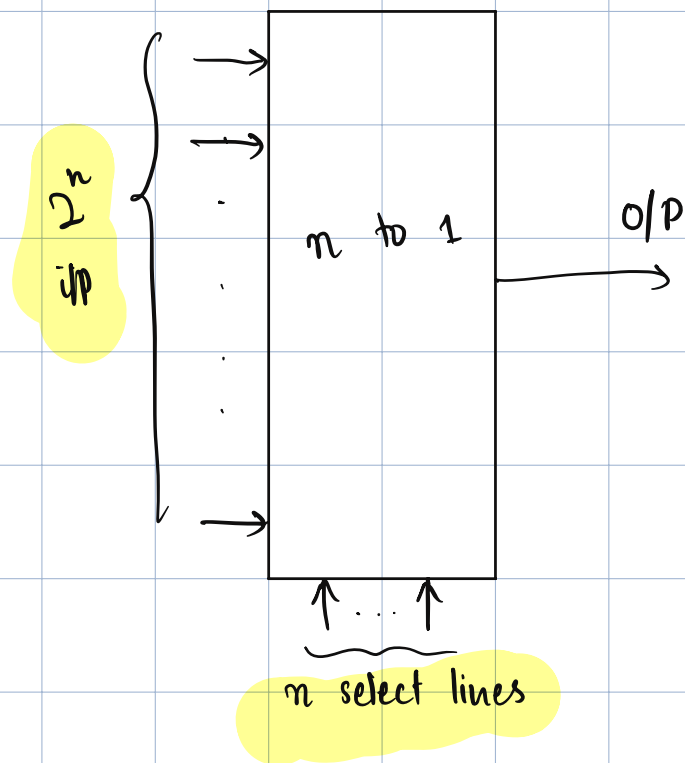
$\begin{matrix} CD \\ AB \end{matrix}$	$\bar{C}\bar{D}$	$\bar{C}D$	CD	$C\bar{D}$	Z_0
$\bar{A}\bar{B}$	0 ₀	0 ₁	0 ₃	0 ₂	choose 1
$\bar{A}B$	1 ₄	X ₅	X ₇	X ₆	
AB	1 ₁₂	X ₁₃	X ₁₅	X ₁₄	
$A\bar{B}$	1 ₈	X ₉	X ₁₁	X ₁₀	

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

$$= A + B$$

$\begin{matrix} CD \\ AB \end{matrix}$	$\bar{C}\bar{D}$	$\bar{C}D$	CD	$C\bar{D}$	Z_1
$\bar{A}\bar{B}$	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_out A;  
    :  
);  
:  
:  
}
```

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;
```

logic
≡ auto

```
    // declare output variables to capture the output of the gates
```

wire AND_out, NOR_out, ...

instantiation
↓
positional
↓
named

```
    all_gates uut {
```

```
        . A (A);
```

variable in design → test bench registers

```
        . B (B);    . AND_out (.AND_out); ...
```

```
    }
```

Test bench + design

↓
Verilog

Tools & simulators

↓
check E P Wave

initial begin // initial block

\$dumpfile (" gates.vcd");

\$dumpvar (testbench);

← \$monitor (

only
when
variable
changes

\$display

// Test cases :

A = 0 , B = 0 , # 10

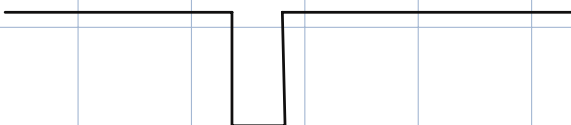
100 100 ns end test
 \$finish

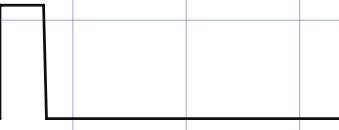
The screenshot shows a web-based Verilog editor with a dark theme. The code defines a module `mux_2to1` with output `Y` and inputs `A`, `B`, and `SEL`. It uses structural modeling to instantiate a NOT gate (`U1`), two AND gates (`U2`, `U3`), and one OR gate (`U4`). The browser tabs at the top include "Bits-Verilog-Sol", "C Programming - Y...", "Clock Domain Cross...", and "Asynchronous FIFO...". The editor's top bar has "Resources", "Community", "Help", and "Playgrounds" menus. The file name "design.sv" is visible in the top left of the editor area.

```
1 module mux_2to1 (output Y, input A, B, SEL);
2
3     wire notSEL;    // wire for the inverted select signal
4     wire andA, andB; // wires for the outputs of the AND gates
5
6     // Instantiate NOT gate for inverting the select line
7     not U1 (notSEL, SEL);
8
9     // Instantiate AND gates for A and B paths
10    and U2 (andA, A, notSEL);
11    and U3 (andB, B, SEL);
12
13    // Instantiate OR gate to combine the outputs of the AND gates
14    or U4 (Y, andA, andB);
15
16 endmodule
17
```

① simpler
② assign → behavioral
② structural
① always control driven

Timing hazards and glitches

static -1 

static 0 

different delays for each gate
dynamic 

Timing diagrams

HDL bits

NPTL
Hardware
no telling us
serial till
22 lectures

Solution

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

* use buffers

assign with delay:

assign #1 nc = -C;

1ns delay