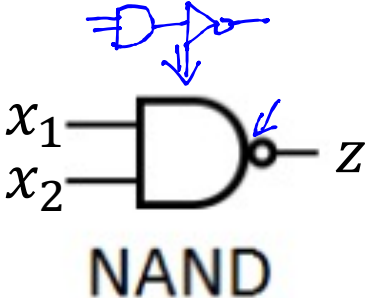
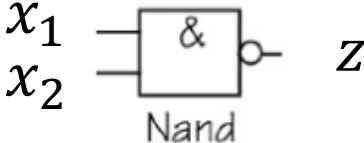
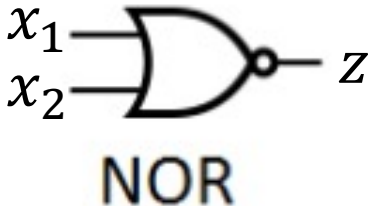
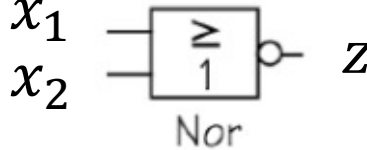

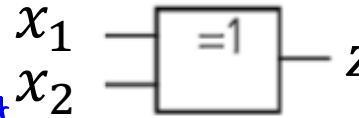

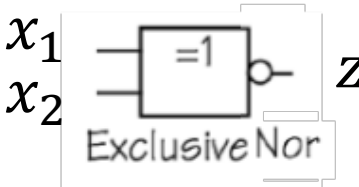


Additional Logic Gates (Derived from Basic Gates)

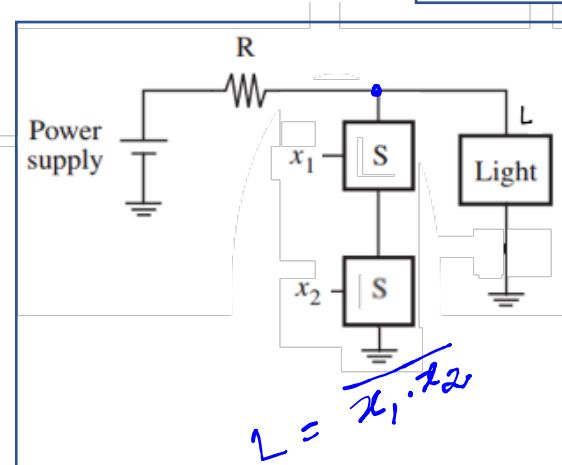
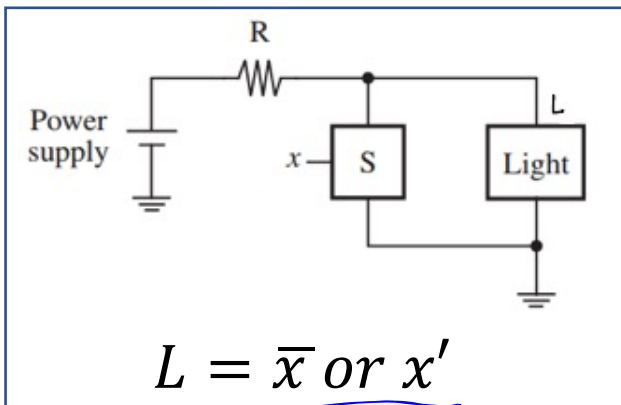
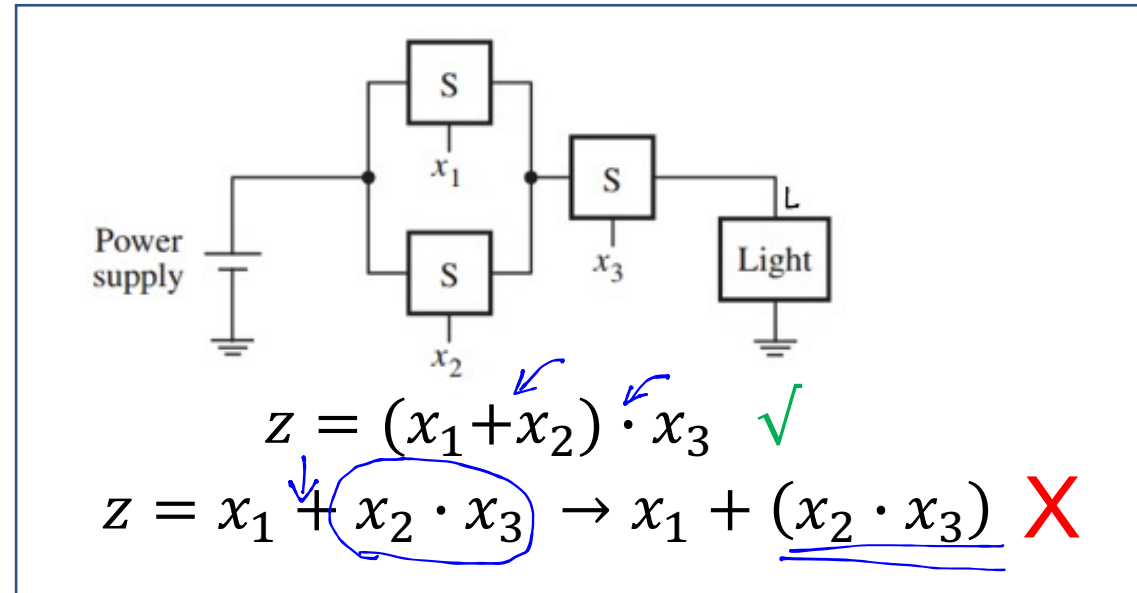
Name	Symbol	IEEE	Function	Truth Table															
NAND			$Z = \overline{x_1 \cdot x_2}$	<table><tr><th>x_1</th><th>x_2</th><th>Z</th></tr><tr><td>0</td><td>0</td><td>1</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table>	x_1	x_2	Z	0	0	1	0	1	1	1	0	1	1	1	0
x_1	x_2	Z																	
0	0	1																	
0	1	1																	
1	0	1																	
1	1	0																	
NOR			$Z = \overline{x_1 + x_2}$	<table><tr><th>x_1</th><th>x_2</th><th>Z</th></tr><tr><td>0</td><td>0</td><td>1</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table>	x_1	x_2	Z	0	0	1	0	1	0	1	0	0	1	1	0
x_1	x_2	Z																	
0	0	1																	
0	1	0																	
1	0	0																	
1	1	0																	

Additional Logic Gates

Name	Symbol	IEEE	Function	Truth Table															
<div>exclusivity x₁ & x₂ x₁ ≠ x₂</div> <div>(E)XOR</div> <div>Exclusive OR</div>	<div>if equal → 0 not equal → 1</div> <div></div> <div>XOR</div> <div>extensively used</div>	<div></div> <div>Exclusive Or</div>	<div>circle OR ⊕</div> <div>$Z = x_1 \oplus x_2$</div> <div>Basic X derived</div> <div>$\bar{x}_1 \cdot x_2 + x_1 \cdot \bar{x}_2$</div> <div>NOT AND OR</div>	<table><tr><th>x₁</th><th>x₂</th><th>Z</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table>	x ₁	x ₂	Z	0	0	0	0	1	1	1	0	1	1	1	0
x ₁	x ₂	Z																	
0	0	0																	
0	1	1																	
1	0	1																	
1	1	0																	
<div>(E)XNOR</div> <div>Exclusive NOR</div>	<div>Inclusive fn</div> <div></div> <div>XNOR</div>	<div></div> <div>Exclusive Nor</div>	$Z = \overline{x_1 \oplus x_2}$	<table><tr><th>x₁</th><th>x₂</th><th>Z</th></tr><tr><td>0</td><td>0</td><td>1</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table>	x ₁	x ₂	Z	0	0	1	0	1	0	1	0	0	1	1	1
x ₁	x ₂	Z																	
0	0	1																	
0	1	0																	
1	0	0																	
1	1	1																	

Precedence of Operations (Similar to BODMAS in arithmetic)

- Order: (), NOT, AND, OR

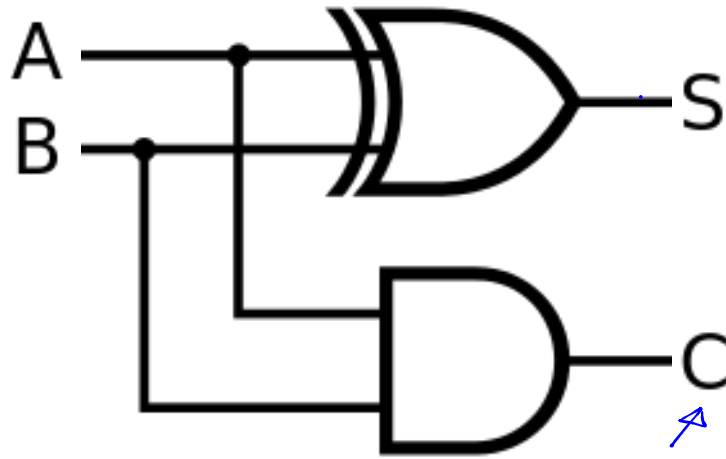


$$L = (x_1 \cdot x_2)' = \overline{(x_1 \cdot x_2)} \quad \checkmark$$

$$L = x_1 \cdot x_2' \quad \text{✗}$$

$$x_1 \cdot x_2' \neq (x_1 \cdot x_2)'$$

- Logical Analysis (Truth table, Logical function, Timing diagram)



A	B	S	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Truth Table

delay

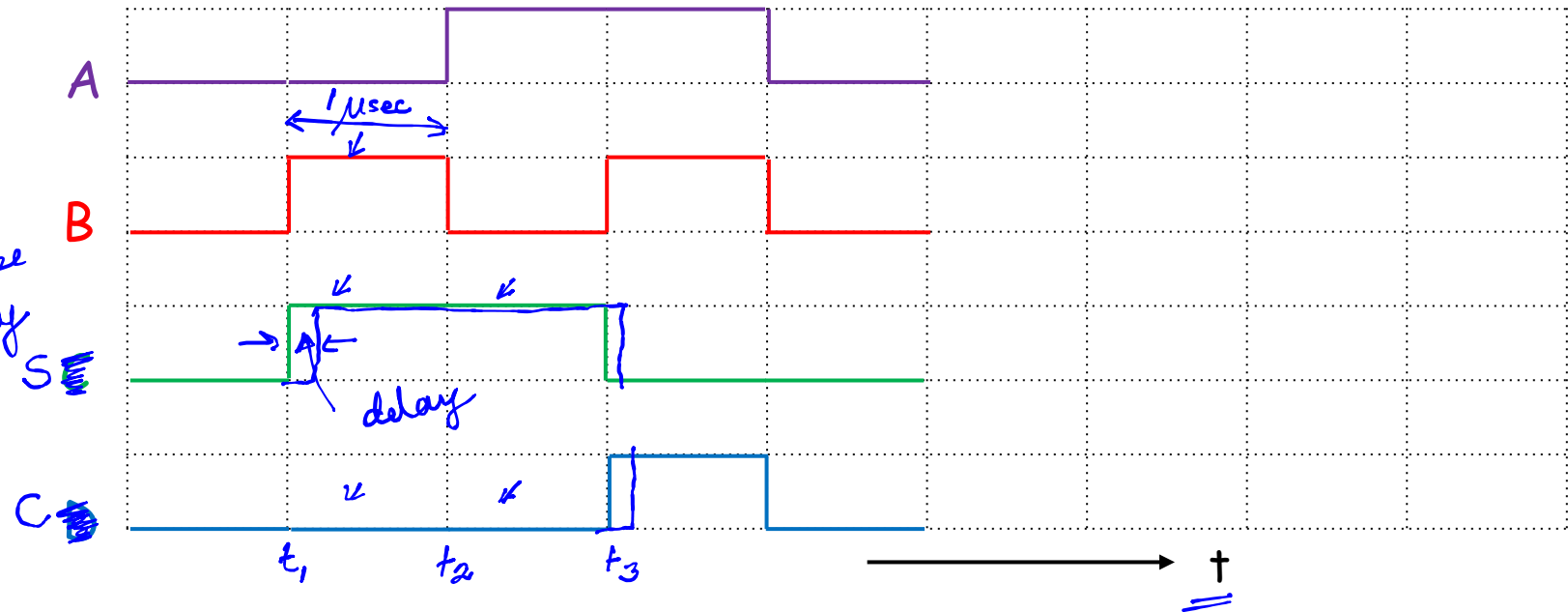
Timing diagrams

$10^6 \text{ Hz} \rightarrow 1 \text{ MHz}$

$S(A, B) = A \oplus B$

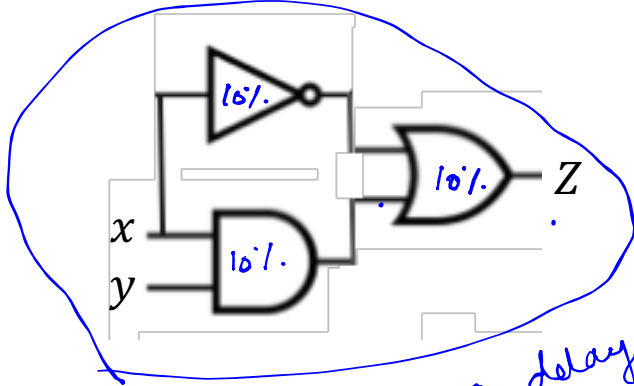
$C = A \cdot B$

1 p sec delay free response no delay



- Logical Analysis (Truth table, Logical function, Timing diagram)

- $Z = \bar{x} + x \cdot y$



assume 0 delay
assume 10% delay

x	y	Z
0	0	1
0	1	1
1	0	0
1	1	1

Fill it up on your own

- Logical Analysis (Truth table, Logical function, Timing diagram)

- $\bar{x} + y$

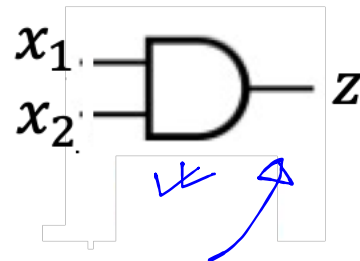
draw the Kkt.

All yours

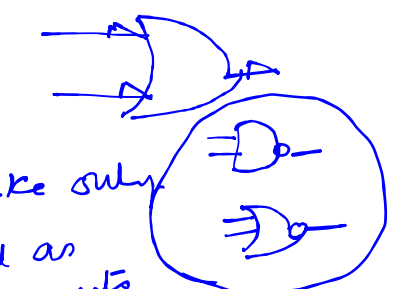
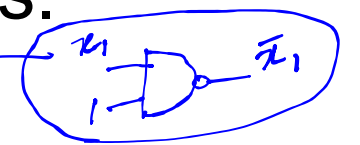
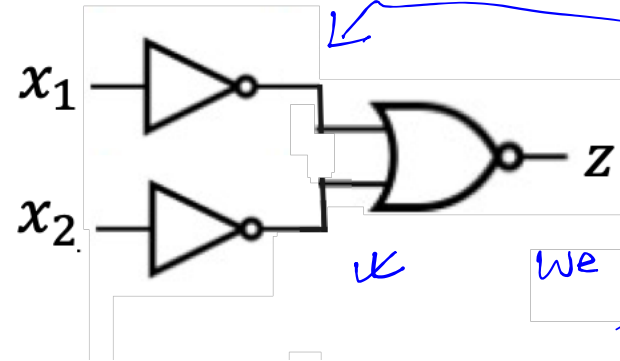
Functionally Equivalent Networks

$2 \rightarrow 4$

- When the truth table for any two or more networks are the same, then they are called as functionally equivalent networks.



\equiv



We make only
used as
universal elements

- Ideally, we should use the network having lowest cost (say, no of gates, delay, area, power etc.)
- For complex network, truth table approach may not be feasible
- Boolean Algebra:** for analyzing logical networks

6 Variables
 2^6

COMBINATIONAL DESIGN EXAMPLE

- Suppose that an automobile has three switches that can operate the turn signals.
 - Two of these are on the stalk behind the steering wheel, one for the left turns and one for the right.
 - The third is the emergency-flasher button that activates both the left and right signals.
- Design the digital logic needed to activate the left- and right-turn/emergency signals.

- We don't have complete information. For example, what kind of switches are these? *Not necessary for me. → information*
- How do they provide the information, in binary, presumably, that they have been actuated?
- But when we are developing something, we often find that the statement of the problem is far from complete.
- What we need are the specifications *problem* that tie down the details so completely that we are not going to make wrong assumptions.
- These specifications also serve as a guide and a contract, because once we have everything tied down, we will get our customer to agree to them.
- Then both the customer and we will know when we have finished the design.
- The specifications form a major part of the contract between us. (Don't forget to include some money considerations too!)

Specifications

We are proposing following as the specifications:

- The three switches are arranged with appropriate logic so that they produce a binary 1 when actuated and a binary 0 otherwise.
- The left- and right-turn switches cannot be actuated at the same time because they are mechanically separate on the stalk.
- Operating the turn signal is to override the emergency flasher.
- The design is only of the activation of the lights; flashing them is not a part of the design.

switch ON $\rightarrow 1$
switch OFF $\rightarrow 0$



right turn $\rightarrow R$ or L left turn
signal is 1
Flasher
OFF





⑪

There! Does that tie things down?

Specifications

- Look back through the information and see if you can find anything missing.
- If you were my customer, you'd have to do this, even if you weren't an expert on digital logic.
- Once you agree to these specs, you are bound to pay me for my work when I show that I have done exactly what they say.
- Mistakes can be expensive!
- The next step is to say this all again in a logic statement.

Truth Table

- The truth table is a complete tabulation of all the possible outcomes for every possible arrangement of inputs. 
- To develop the truth table, need names or labels on the input signals from the switches and the output signals to the lamps.
- We call the left-turn switch L ; when $L = 0$ the switch is not actuated, and when $L = 1$ the switch is actuated.
- The right-turn switch will be R and the emergency-flasher switch will be F .
- Output signals will go from my logic to the lamps at the front and rear of the car. 
- I'll call the signal to the left bulbs LT ; when $LT = 0$ the bulbs are not on, and when $LT = 1$ the bulbs are to be on. 
- The right bulbs will be RT . 

- Now for the truth table, shown in Figure below.
- The headings are the input and output signals.
- On the **left** you'll see all eight possible combinations of three binary digits in ascending numerical sequence.
- On the **right** are the possible outcomes of each of the eight input combinations

Truth Table

No.	L	R	F	LT	RT
→ 0	0	0	0	0	0
1	0	0	1	1	1
2	0	1	0	0	1
3	0	1	1	0	1
4	1	0	0	1	0
5	1	0	1	1	0
6	1	1	0	x	x
→ 7	1	1	1	x	x

Truth Table

8 possible combination

do not exist
don't-case

- 000 Both outputs are 0 because no switch is actuated.
- 001 The emergency flashers are being requested, so both bulbs are lit.
- 010 This is a request for a right turn.
- 011 Both a right turn and the emergency flashers are being requested, but the specifications say that the turn signal is to dominate.
- 100 This is a request for a left turn.
- 101 A left turn dominates when both a left turn and the flasher are requested.
- 110 Here the output is shown as x , which means that I don't care what the output actually is. Why? The specs say that both L and R cannot be actuated at the same time, so the output, whatever it might be, will never happen. (The x is called a *don't care*.)
- 111 Here's another impossible situation.

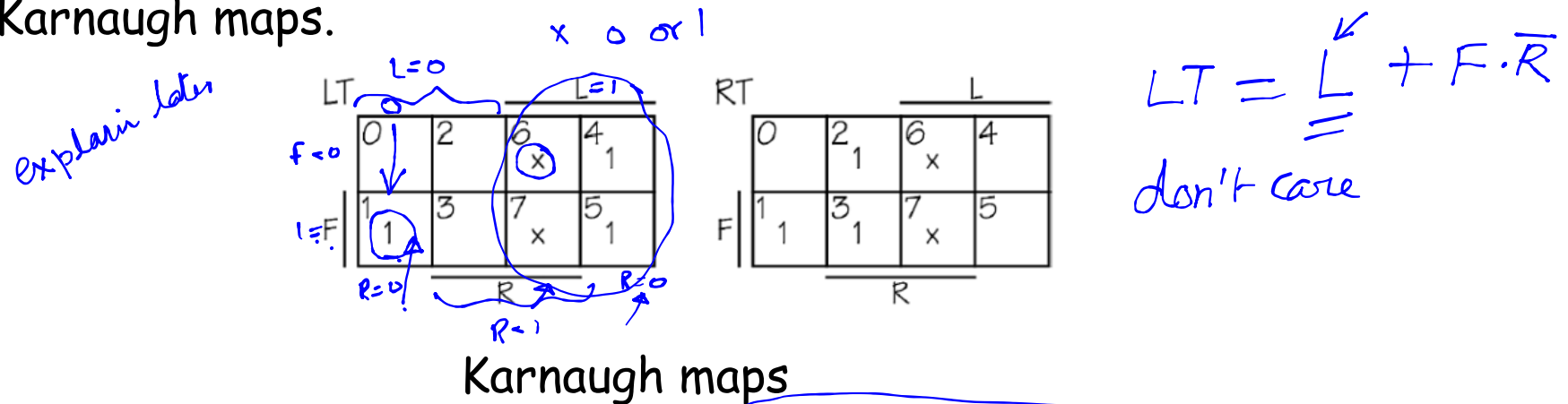
All eight possible switch combinations have been listed and the outputs specified.

I check these against the specifications to make sure I'm still on the right track.

Reduction and Simplification

Now I need to reduce this logic representation to a logic statement that can lead to logic gates.

- To reduce and simplify the logic statements, **the Karnaugh Map**, which is a graphical way of doing the job, is used.
- This is just one way that we could do the job. Figure below shows the Karnaugh maps.



We will do this without explanation --- We will take it up later.