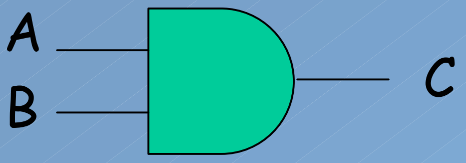
VHDL
Logic Design Overview

EL 310

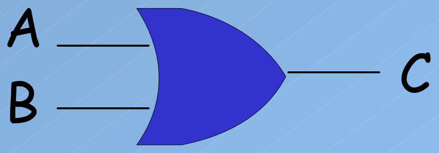
Erkay Savaş
Sabancı University

Combinational Logic

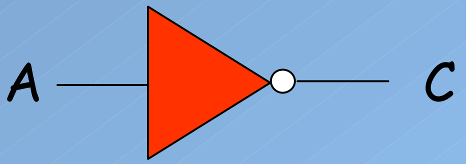
- Basic Gates



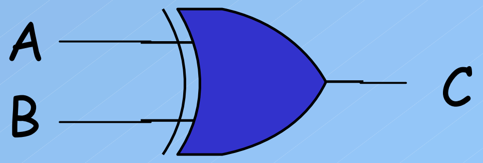
AND: $C = AB$



OR: $C = A+B$



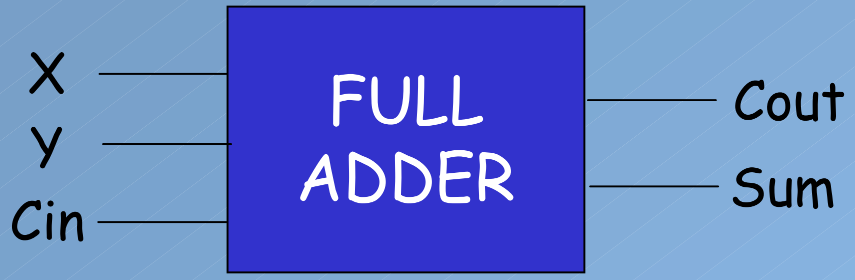
NOT: $C = A'$



XOR: $C = A \oplus B$

Truth Tables and Algebraic Expressions

- Full Adder



X	Y	Cin	Cout	Sum
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

$$Sum = X'Y'Cin + X'Y\,Cin' + X\,Y'Cin' + X\,Y\,Cin$$

$$Cout = X'Y\,Cin + X\,Y'Cin + X\,Y\,Cin' + X\,Y\,Cin$$

Minterm and Maxterm Expansions

- Minterm expansion

Sum = X′ Y′ Cin + X′ Y Cin′ + X Y′ Cin′ + X Y Cin = $\sum$ **m(1, 2, 4, 7)**

Cout = X′ Y Cin + X Y′ Cin +X Y Cin′ + X Y Cin = $\sum$ **m(3, 5, 6, 7)**

- Maxterm expansion

Sum = (X+Y+Cin)(X+Y′+Cin′)(X′+Y+Cin′)(X′+Y′+Cin) **=** $\prod$ **M(0,3, 5, 6)**

Cout = (X+Y+Cin)(X+Y+Cin′)(X+Y′+Cin)(X′+Y+Cin) **=** $\prod$ **M(0,1, 2, 4)**

- Minterm expansion ⇔ Maxterm expansion

Algebraic Simplification 1

1. Combining Terms (XY + XY'=X)
   - ABC'D' + ABCD' = ABD' (C + C') = ABD'

   - Cout = X ' Y Cin + X Y' Cin + X Y Cin' + X Y Cin
     = X' Y Cin + X Y' Cin + X Y Cin' + XY Cin + X Y Cin + X Y Cin
     = X' Y Cin + X Y Cin + X Y' Cin + X Y Cin + X Y Cin' + XY Cin
     = Y Cin + X Cin + XY
2. Eliminating Terms
   - A'B + A'BC = A'B
   - Consensus theorem: XY + X'Z + YZ = XY + X'Z

CBD + C'A'B + DA'B = CBD + C'A'B + BDA'B = CBD + CA'B'

Algebraic Simplification 2

- Eliminating Literals
  - $X + X'Y = X + Y$
  - $A'B + A'B'C'D' + ABCD' = A'(B + B'C'D') + ABCD'$
    $= A'(B + C'D') + ABCD' = A'B + A'C'D' + ABCD'$
    $= B\,(A' + ACD') + A'C'D'$
    $= B\,(A' + CD') + A'C'D' = A'B + BCD' + A'C'D'$
- Adding Redundant Terms
  - Adding XX', multiplying by $(X+X')$,
  - Adding YZ to $XY + X'Z$ or adding XY to X
  - $AB + AD + A'C' + DB'C' = AB + DC'B' + AD + A'C' + DC'$
    $= AB + AD + A'C' + DC' = AB + AD + A'C'$

Duality Principle

- Important property of Boolean algebra
  - If $x + 0 = x$ then $x \cdot 1 = x$

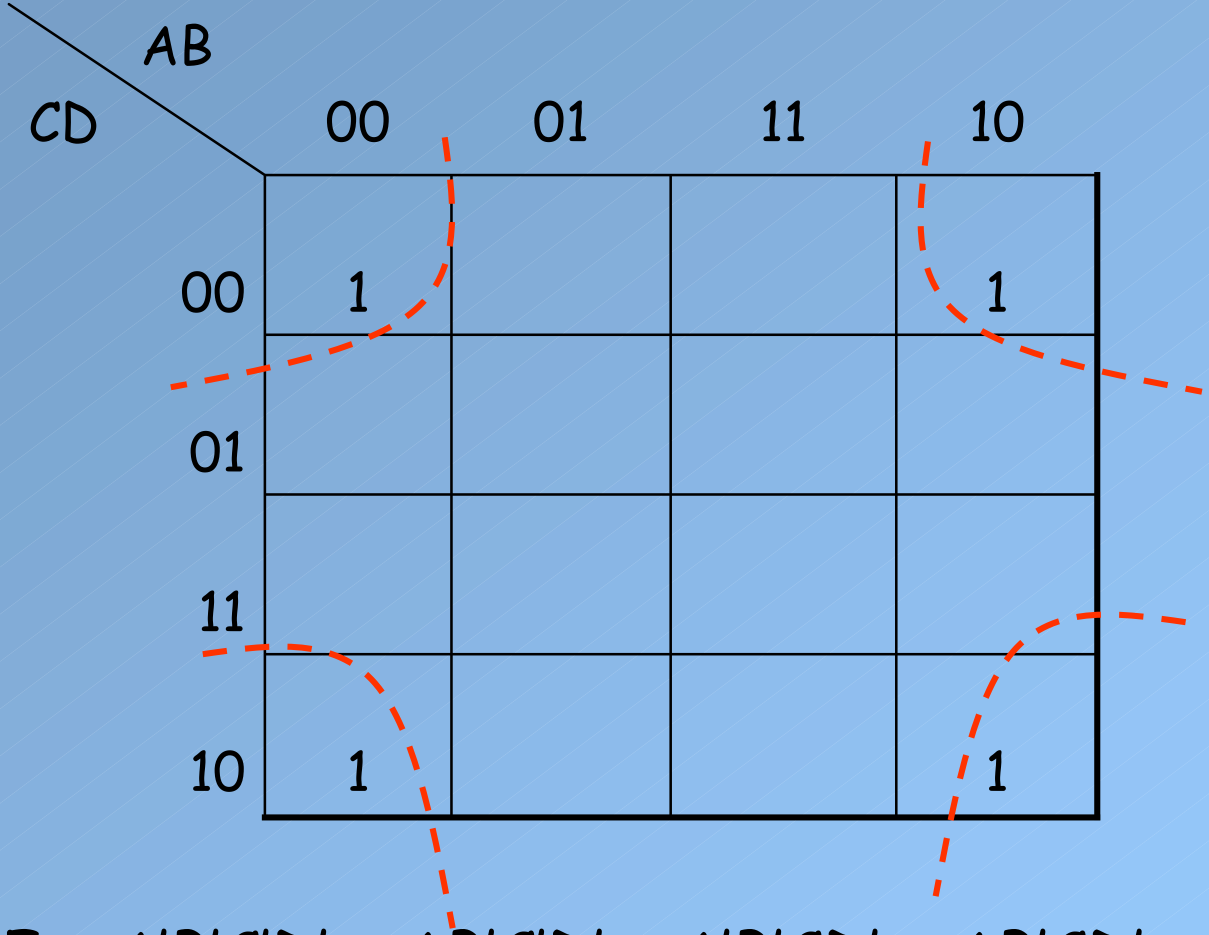
$x + x = x$	$x{\cdot}x = x$
$x + x' = 1$	$x \cdot x' = 0$
$x + 1 = 1$	$x \cdot 0 = 0$
$(x + y + \ldots)' = x' \cdot y' \ldots$	$(x \cdot y \cdot \ldots)' = x' + y' + \ldots$
$x + x \cdot y = x$	$x \cdot (x+y) = x$
$xy + x' z + y z = xy + x' z$	$(x+y)(x'+z)(y+z) = (x+y)(x'+z)$
$(x+y)(x'+z) = x z + x' y$	$xy + x' z = (x+z) (x'+y)$

Exclusive OR (XOR)

- Some rules
  - $X \oplus 0 = X$
  - $X \oplus 1 = X'$
  - $X \oplus X = 0$
  - $X \oplus X' = 1$
  - Commutative, associative, and distributive laws hold.
  - $X \oplus Y = X'Y + XY'$
  - $(X \oplus Y)' = X \oplus Y' = X' \oplus Y = XY + X'Y'$
  - $$\begin{aligned} \text{Sum} &= X' Y' Cin + X' Y Cin' + X Y' Cin' + X Y Cin \\ &= (X' Y' + X Y) Cin + (X' Y + X Y') Cin' \\ &= (X \oplus Y)' Cin + (X \oplus Y) Cin' \\ &= X \oplus Y \oplus Cin \end{aligned}$$

Karnaugh Maps (1)

- F = A'B'C'D' + AB'C'D' + A'B'CD' + AB'CD'

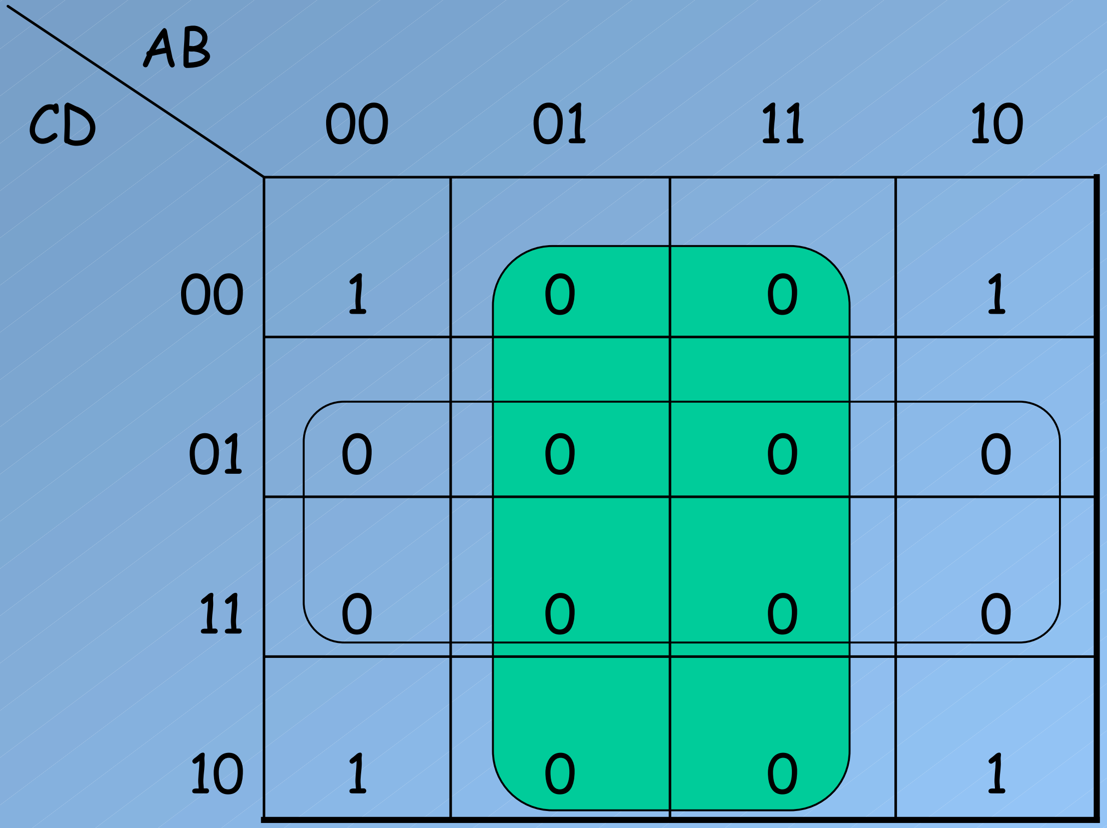


CD \ AB	00	01	11	10
00	1			1
01				
11				
10	1			1

- F = A'B'C'D' + AB'C'D' + A'B'CD' + AB'CD' = B'D'

Karnaugh Maps (2)

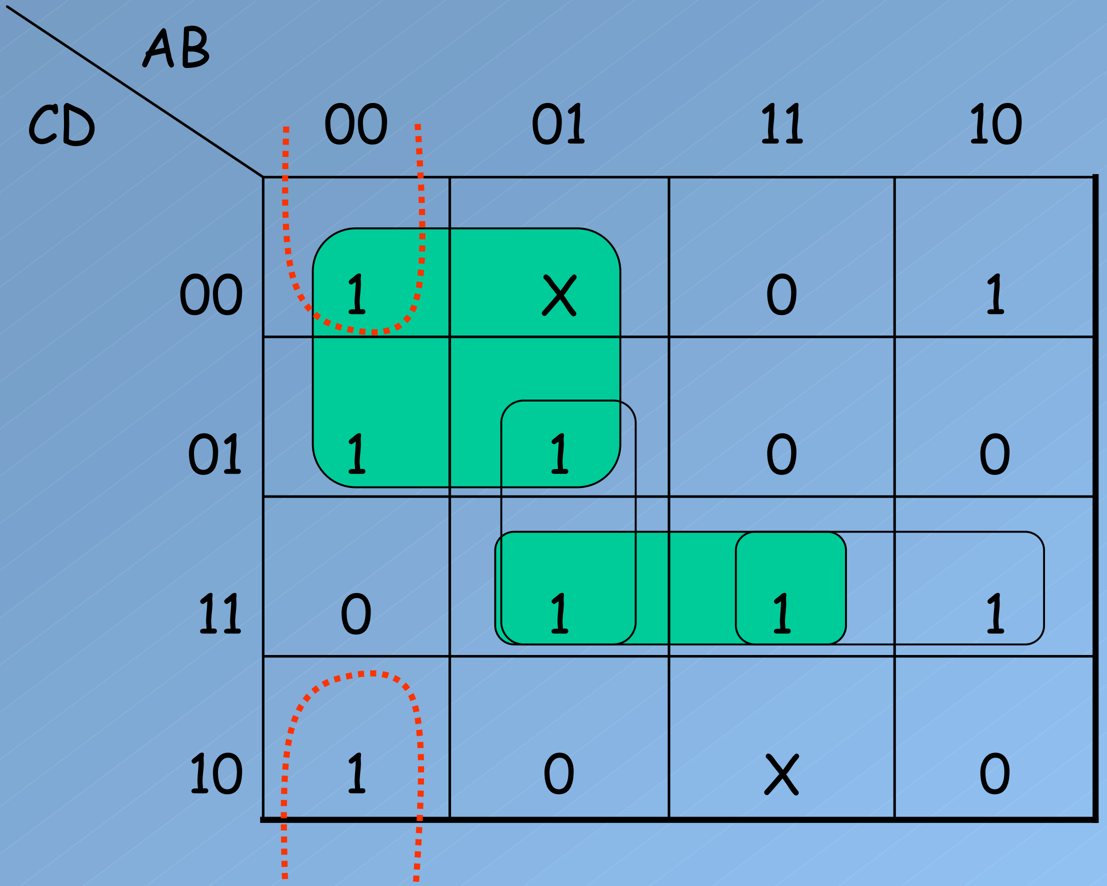
- F = A'B'C'D' + AB'C'D' + A'B'CD' + AB'CD'



CD \ AB	00	01	11	10
00	1	0	0	1
01	0	0	0	0
11	0	0	0	0
10	1	0	0	1

- F = (B+D)' = B'D' (DeMorgan's Law)

Karnaugh Maps (3)

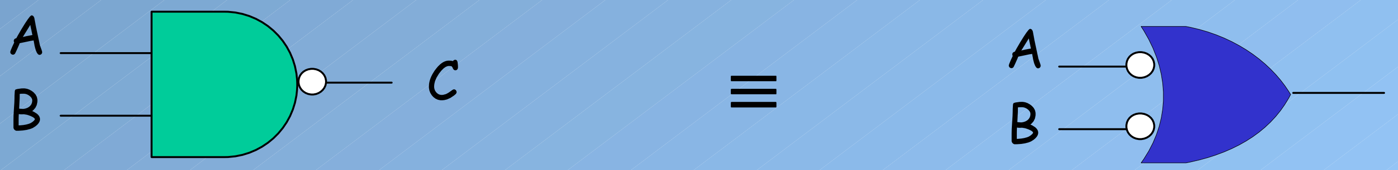


CD \ AB	00	01	11	10
00	1	X	0	1
01	1	1	0	0
11	0	1	1	1
10	1	0	X	0

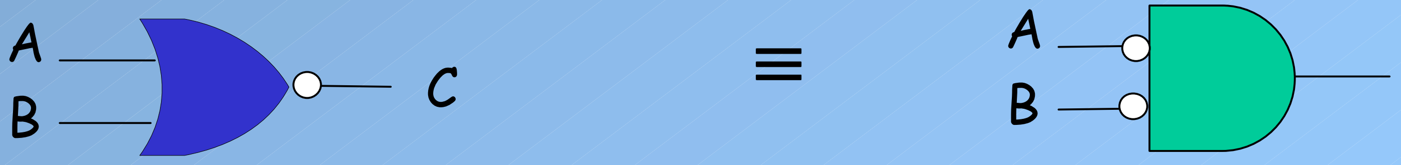
- F = A'C' + A'B'D'+ ACD + BCD   or
- F = A'C' + A'B'D'+ ACD + A'BD

Designing with NAN D and NOR gates - 1

- In many technologies, implementation of NAND or NOR gates is easier than that of AND or OR gates
  - Any logic function can be realized using only NAND gates or only NOR gates



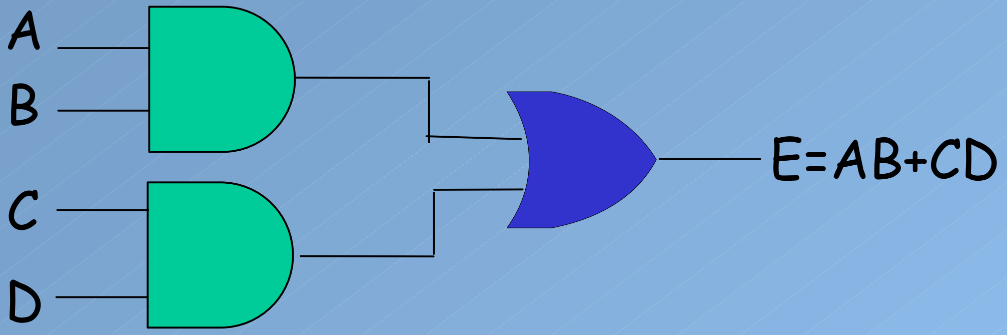
NAND: $C = (AB)' = A'+B'$

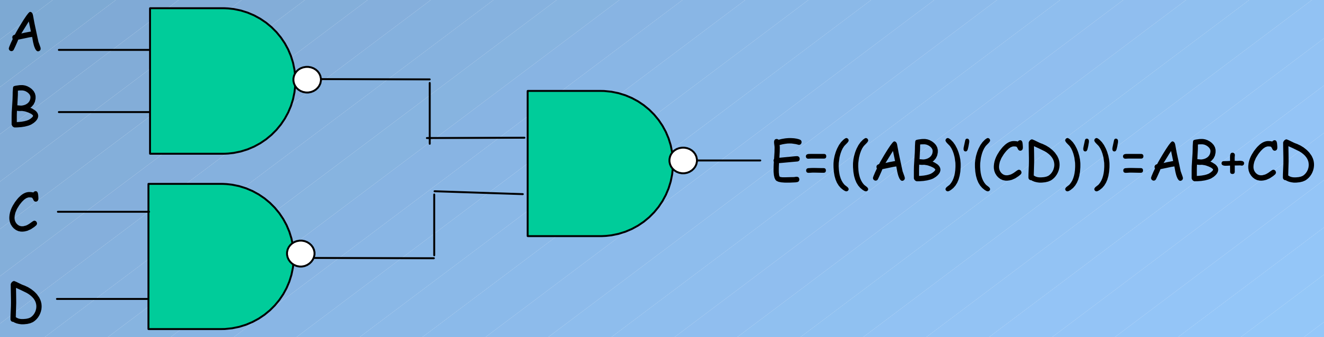


NOR: $C = (A+B)' = A'B'$

Designing with NAN D and NOR gates - 2

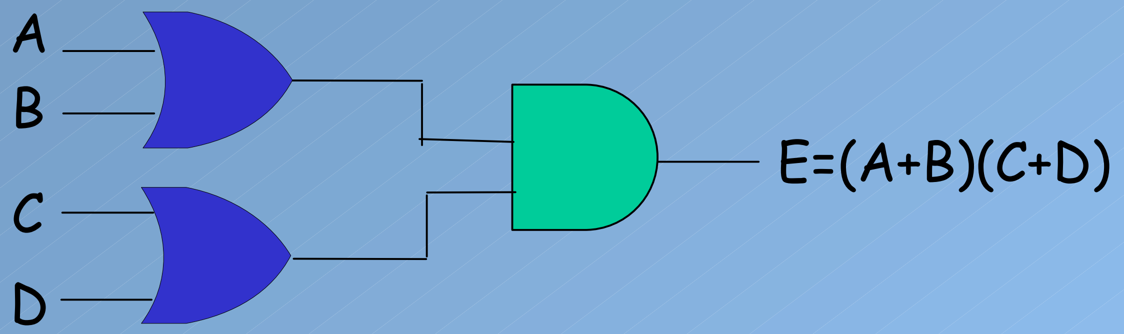
- Conversion of AND-OR network to NAND gates

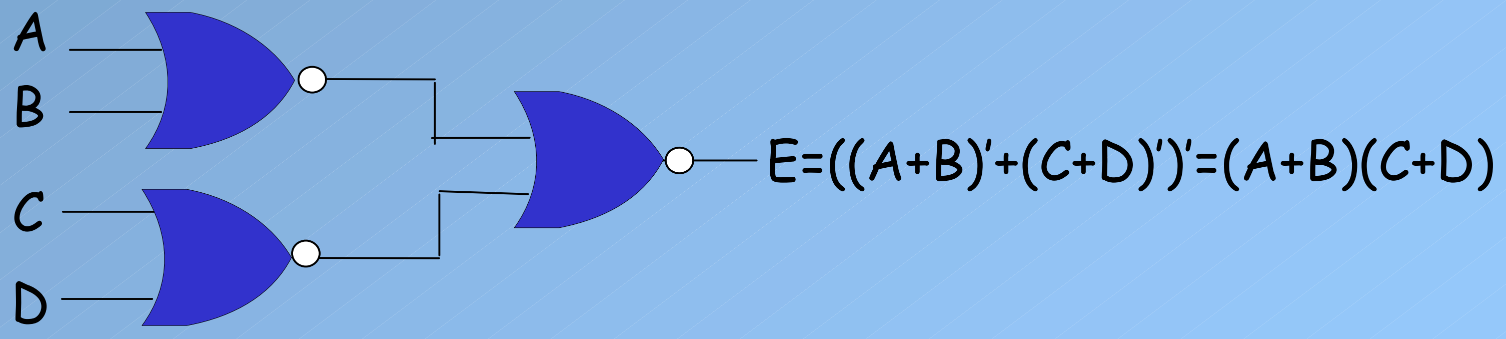




Designing with NAN D and NOR gates - 3

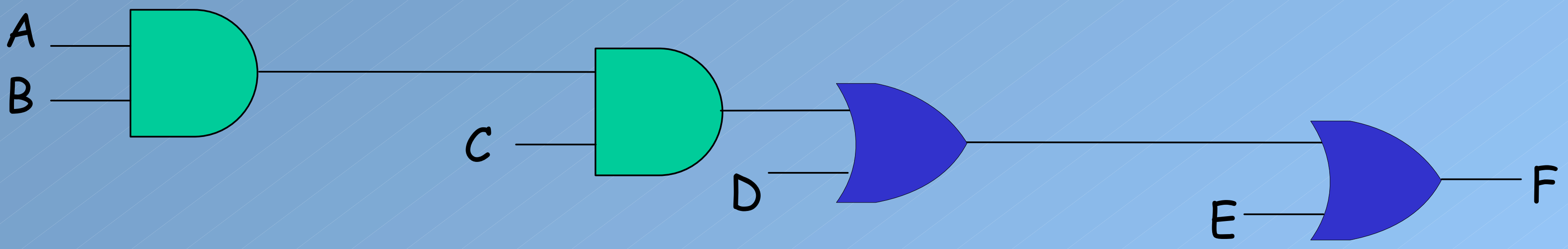
- Conversion of AND-OR Network to NOR gates



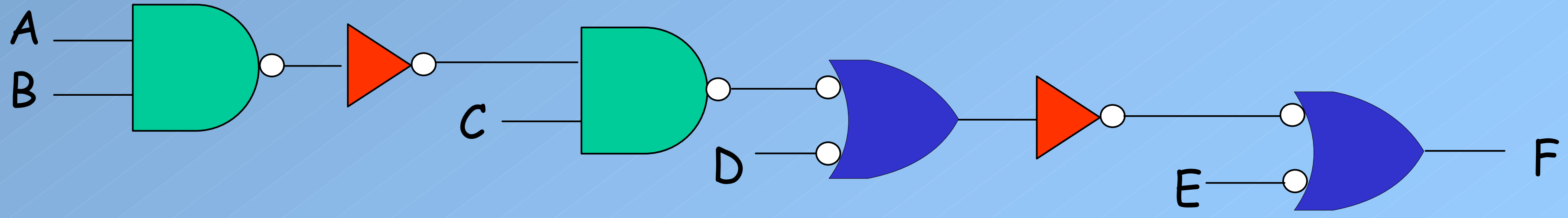


Designing with NAN D and NOR gates - 4

- Conversion of AND-OR network to NAND gates



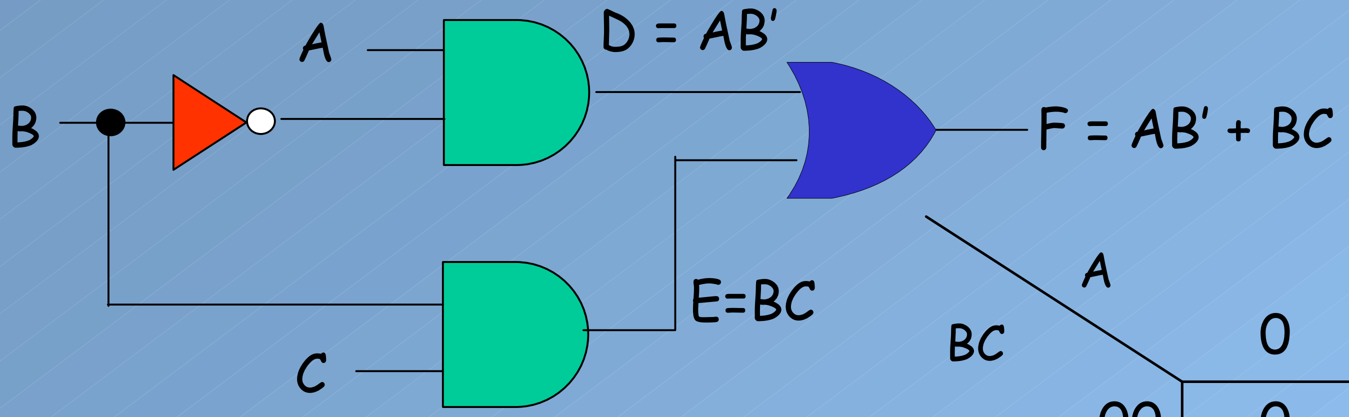
≡



Hazards in Combinational Networks - 1

- Unwanted switching transients
  - Occur when different paths from input to output have different propagation delays
  - 1-*hazard* : network output may momentarily go to 0 when it should remain a constant 1
  - 0-*hazard* : network output may momentarily go to 1 when it should remain a constant 0
  - *Dynamic-hazard* : when the output is supposed to change (1-0 or 0-1), the output may change three times (e.g. 1-0-1-0)

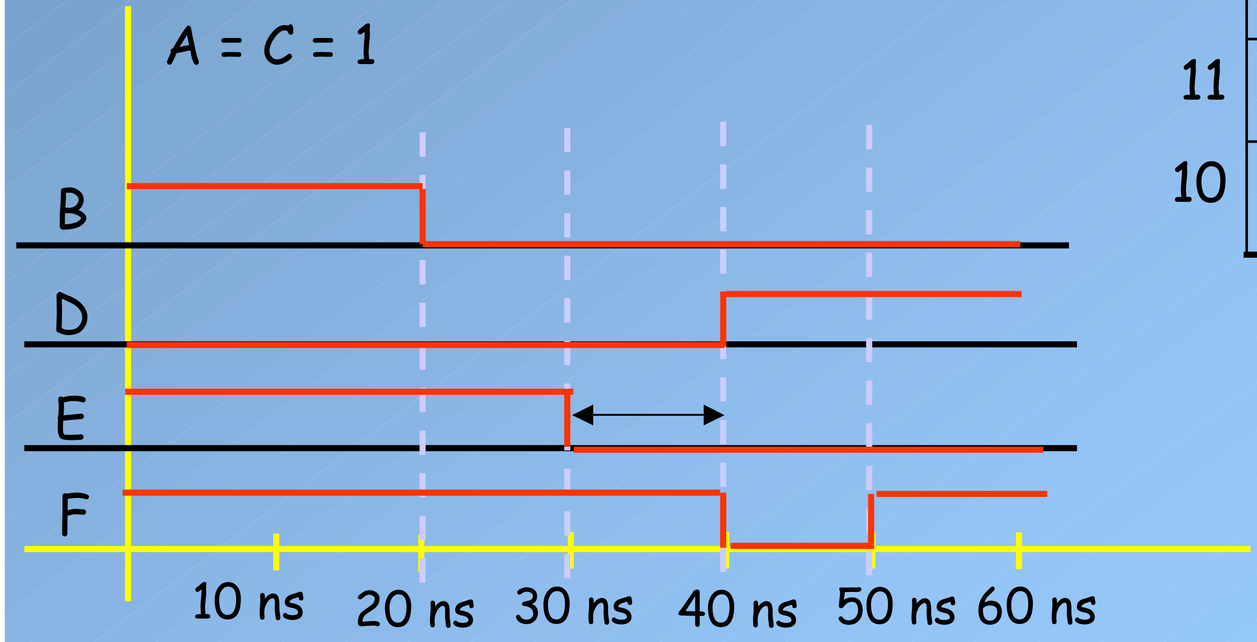
Hazards in Combinational Networks -2



BC \ A	0	1
00	0	1
01	0	1
11	1	1
10	0	0

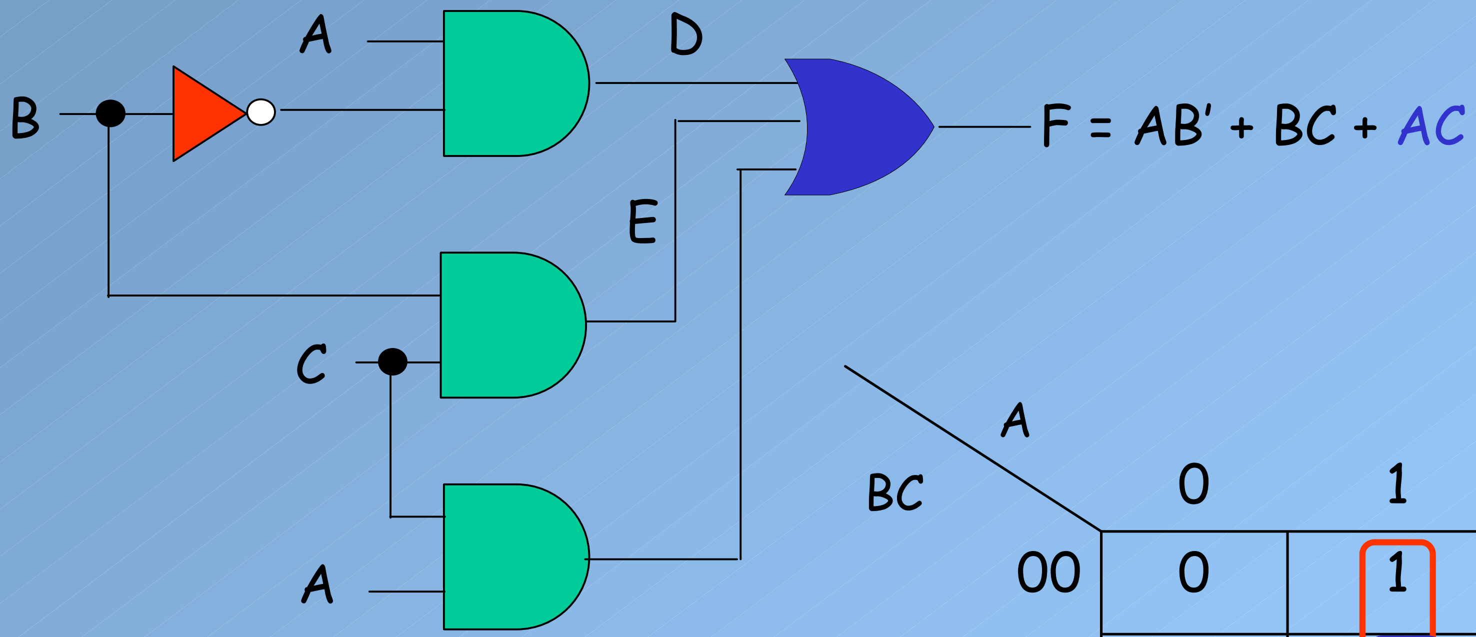
1-hazard

Each gate has 10 ns propagation delay!

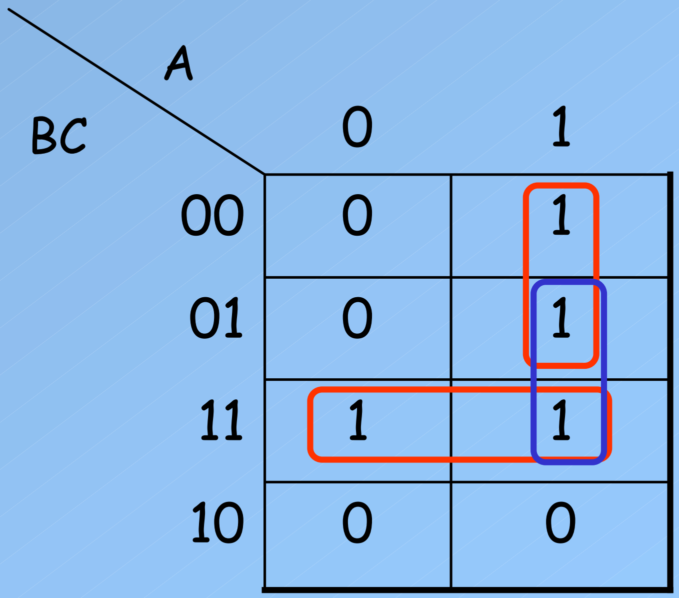


Hazards in Combinational Networks -3

- Remedy



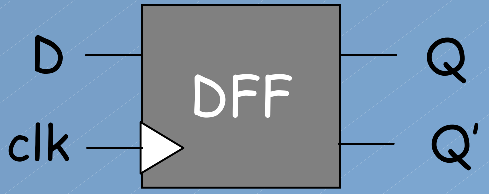
$F = AB' + BC + AC$



BC \ A	0	1
00	0	1
01	0	1
11	1	1
10	0	0

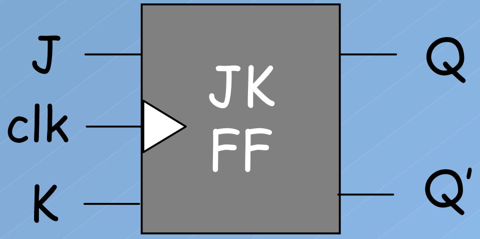
Flip-Flops

D Flip-Flop



D	Q	Q^+
0	0	0
0	1	0
1	0	1
1	1	1

Clocked JK Flip-Flop

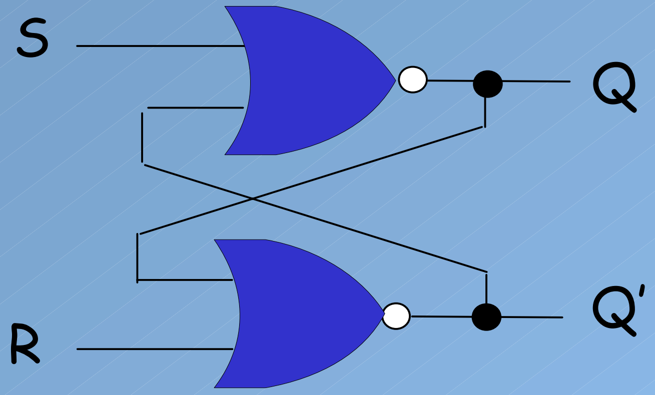


Q = JQ' + K'Q

J	K	Q	Q^+
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

Latches - 1

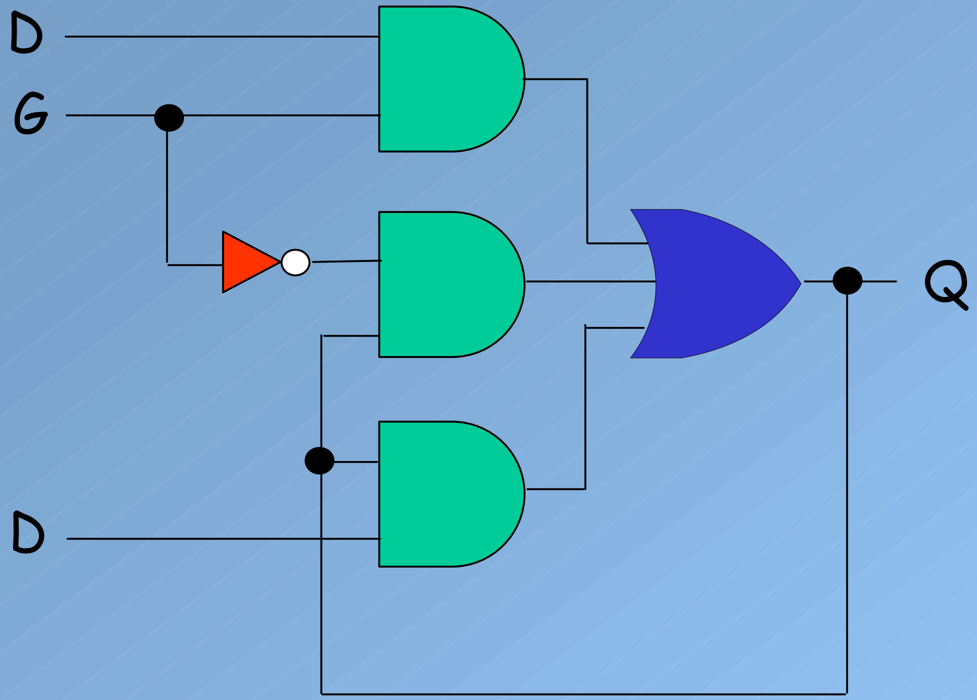
S-R Latch



S	R	Q	Q^+
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	-
1	1	1	-

Latches - 2

Gated D Latch

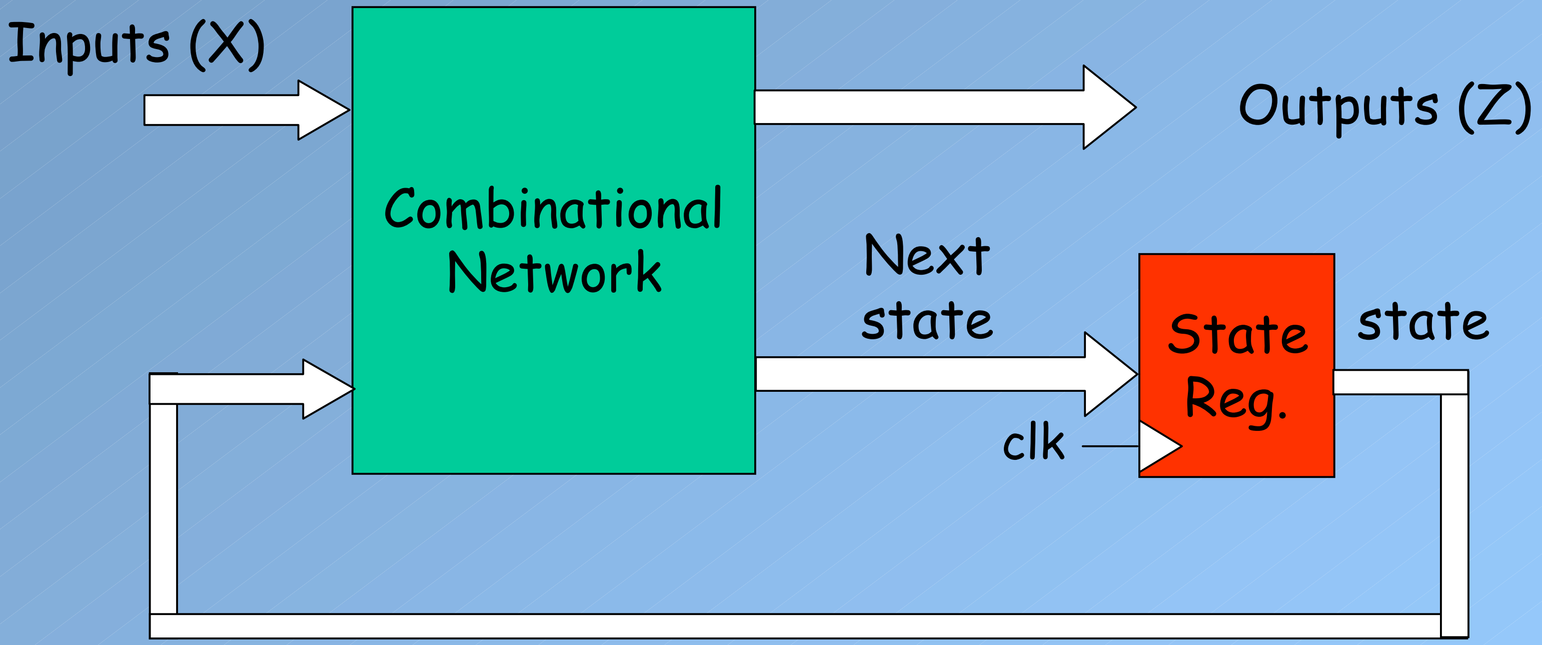


G	D	Q	Q^+
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

$$Q^+ = DG + G'Q + DQ$$

Mealy Sequential Network Design

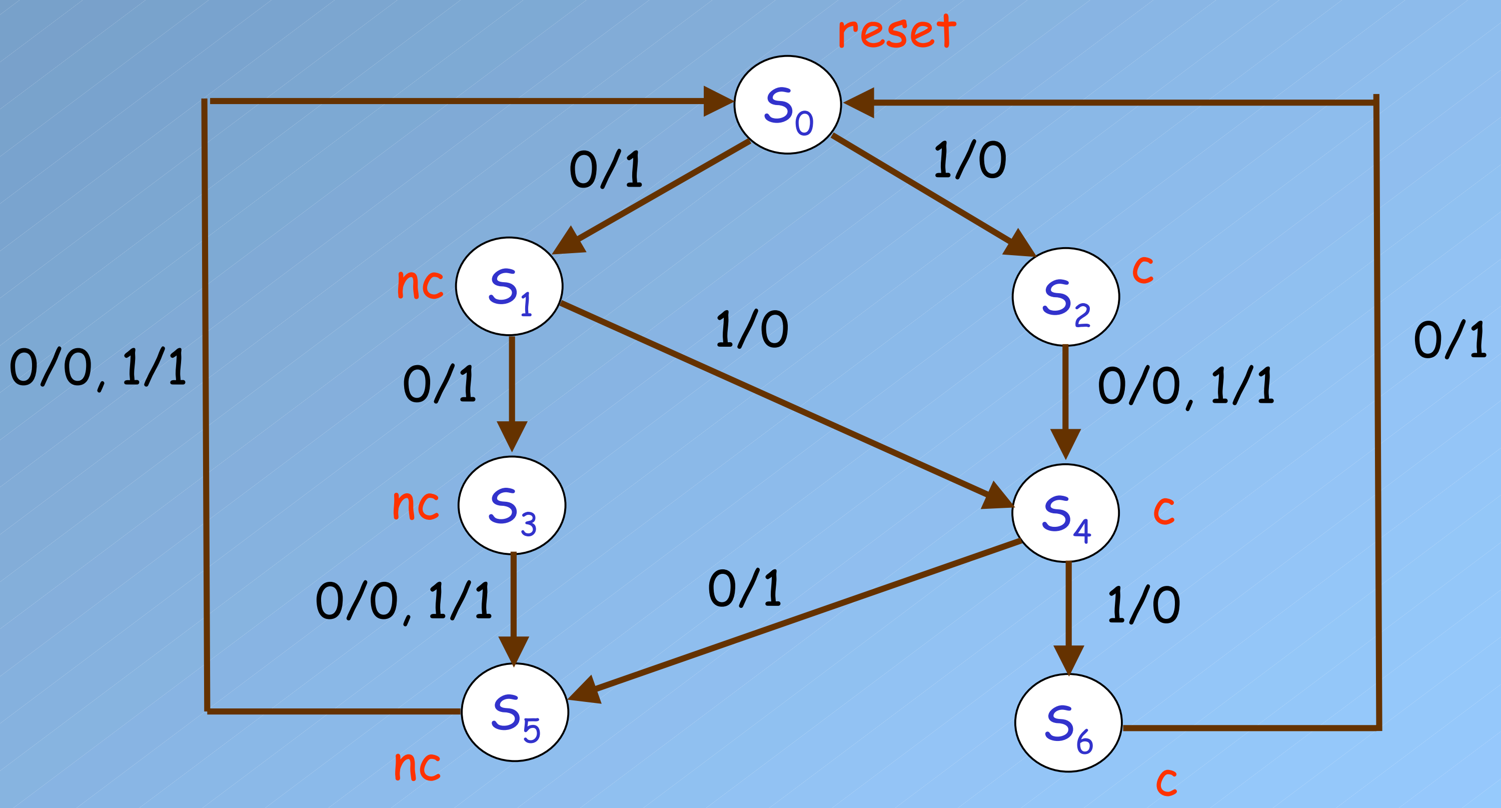
General Model of Mealy Sequential Design



Outputs depend on both present state and present input

Mealy Machine – Example (1)

- Problem: code conversion
  - Convert a BCD digit to another code by adding 3.
  - For example, $(1001)_2 + (0011)_2 = (1100)_2$
  - Serial input/output, one bit at a time starting LSB.



Mealy Machine – Example (2)

- State table

Present State	Next State		Output	
	input=0	input=1	input=0	input=1
S_0	S_1	S_2	1	0
S_1	S_3	S_4	1	0
S_2	S_4	S_4	0	1
S_3	S_5	S_5	0	1
S_4	S_5	S_6	1	0
S_5	S_0	S_0	0	1
S_6	S_0	-	1	-

- Seven states ➔ three flip-flops
- Challenge: state assignment to FF so that logic is the simplest

Mealy Machine – Example (3)

- State Assignments

Q_2Q_3 \ Q_1	0	1
00	S_0	S_1
01	0	S_2
11	S_5	S_3
10	S_6	S_4

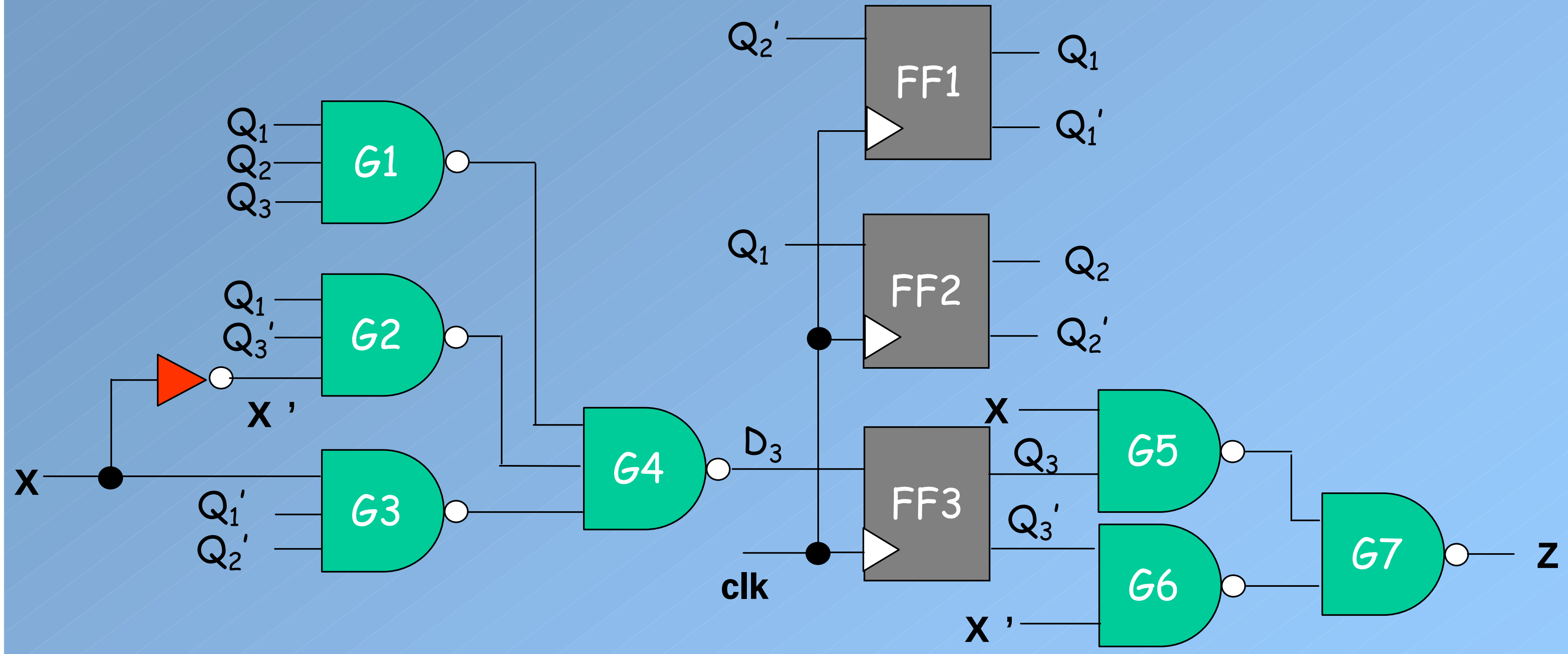
Assignment map

	$Q_1^+Q_2^+Q_3^+$		Z	
$Q_1Q_2Q_3$	X=0	X=1	X=0	X=1
000	100	101	1	0
100	111	110	1	0
101	110	110	0	1
111	011	011	0	1
110	011	010	1	0
011	000	000	0	1
010	000	xxx	1	x
001	xxx	xxx	x	x

Transition Table

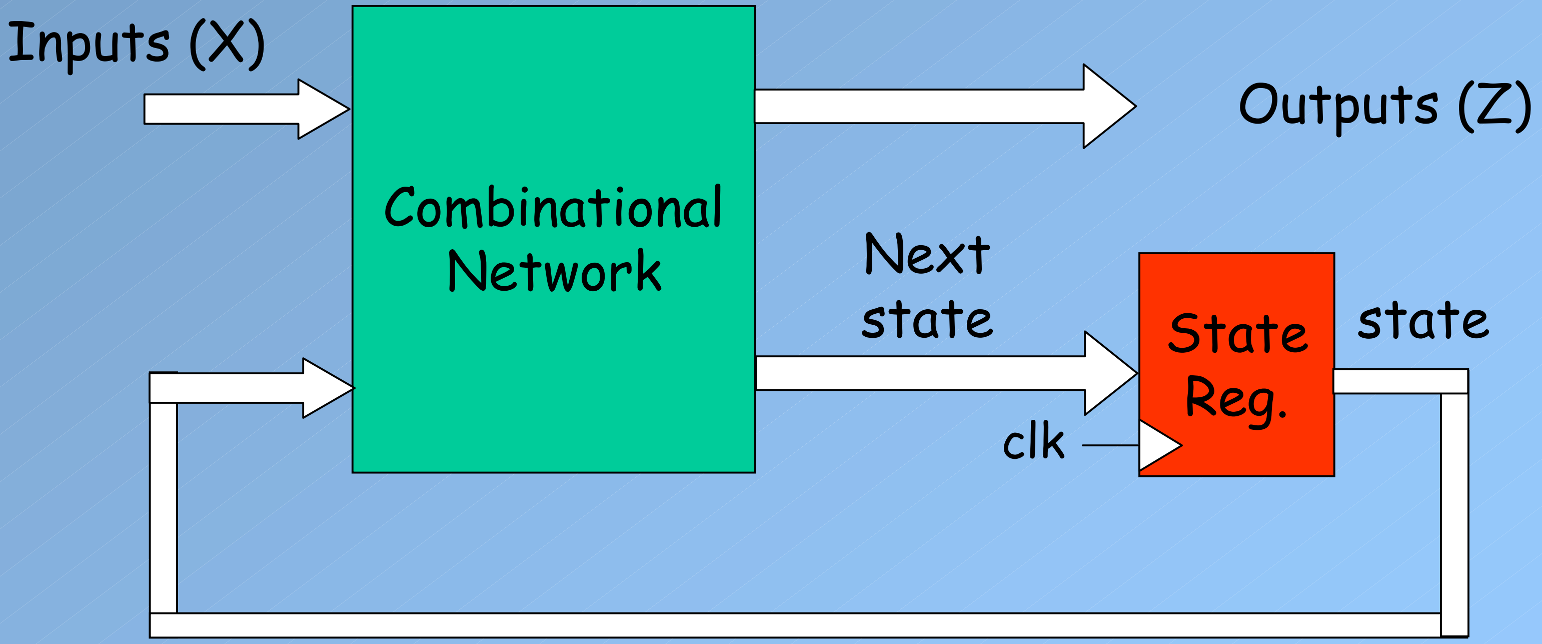
Mealy Machine – Example (4)

- Realization of Code Converter



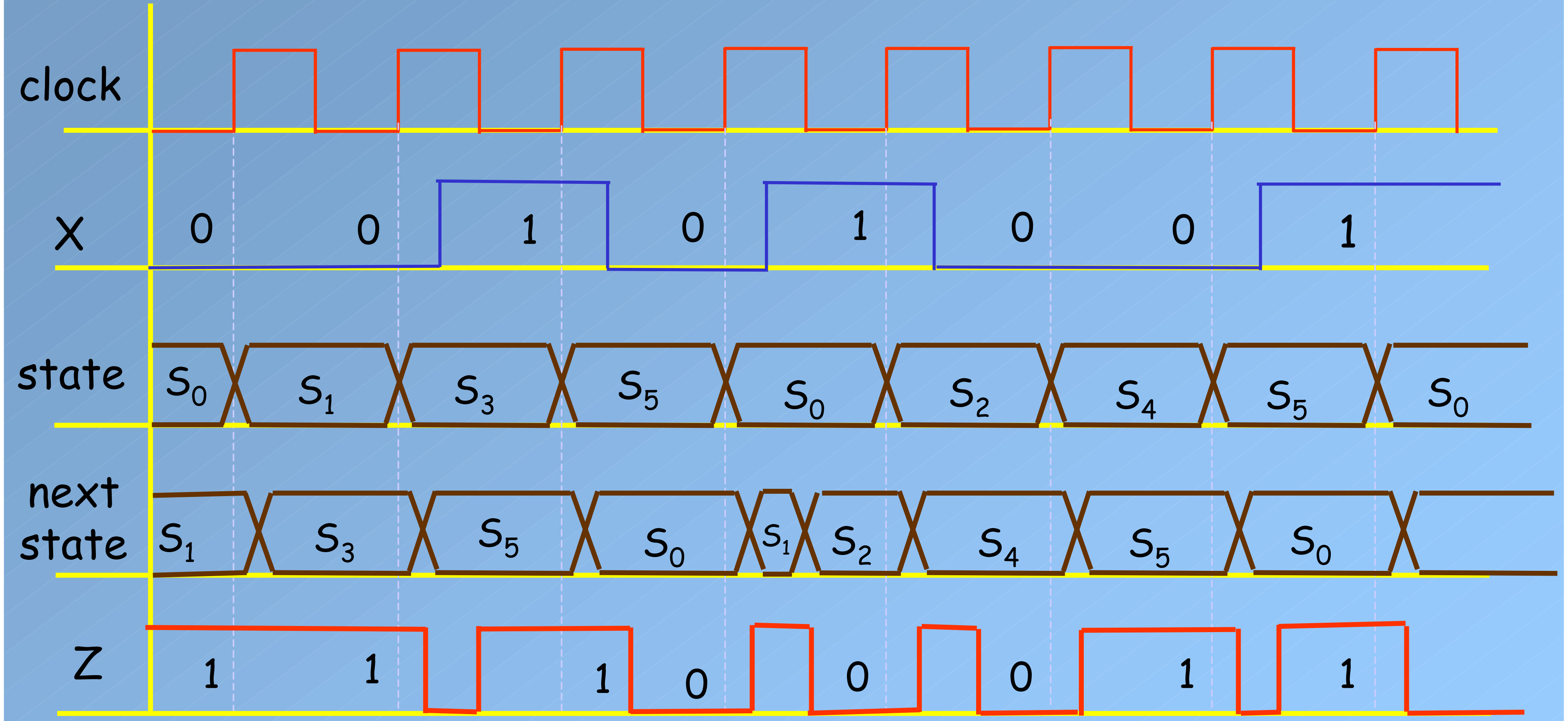
Moore Machine

- The output is only function of the present state, independent of the input.
- Input determines the next state.



Sequential Network Timing

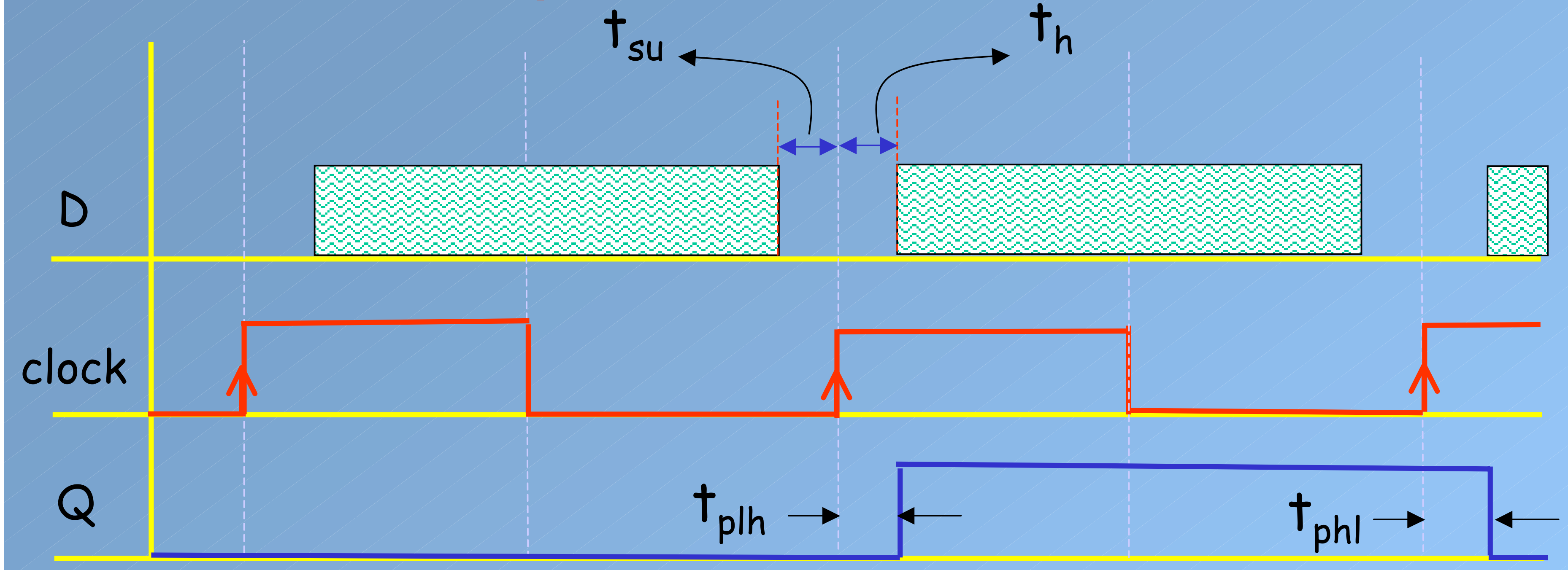
- Timing diagram for Code Converter



Setup and Hold Times 1

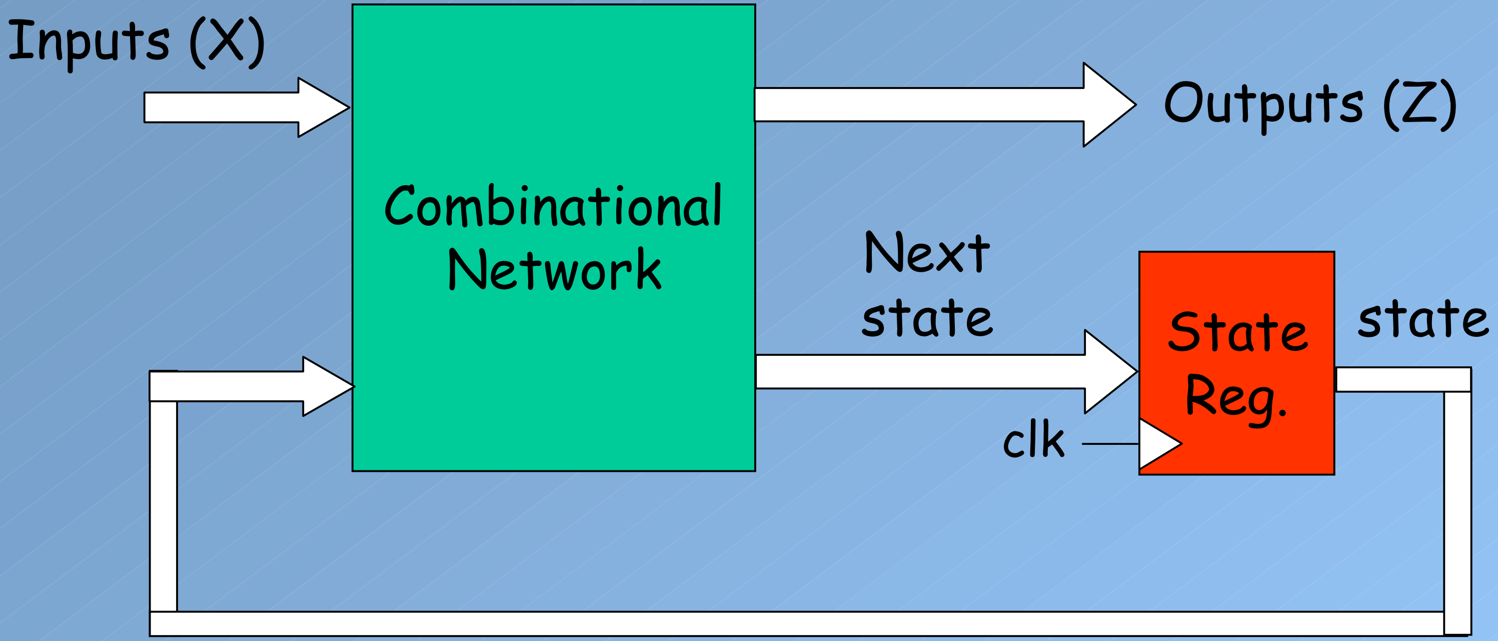
- For an ideal D flip-flop
  - If the D input changes at exactly the same time as the active edge of the clock, flip-flop operate correctly
- In reality
  - The D input must be stable for a certain amount of time before the active edge of the clock; <u>setup time</u>
  - Furthermore, D must be stable for a certain amount of time after the active edge of the clock; <u>hold time</u>

Setup and Hold Times 2



- In shaded interval above D may be changed
- Otherwise, it cannot be determined whether the flip-flop will change state
- Even worse, the flip-flop may malfunction and output a short pulse and even go into oscillation

Maximum Clock Frequency

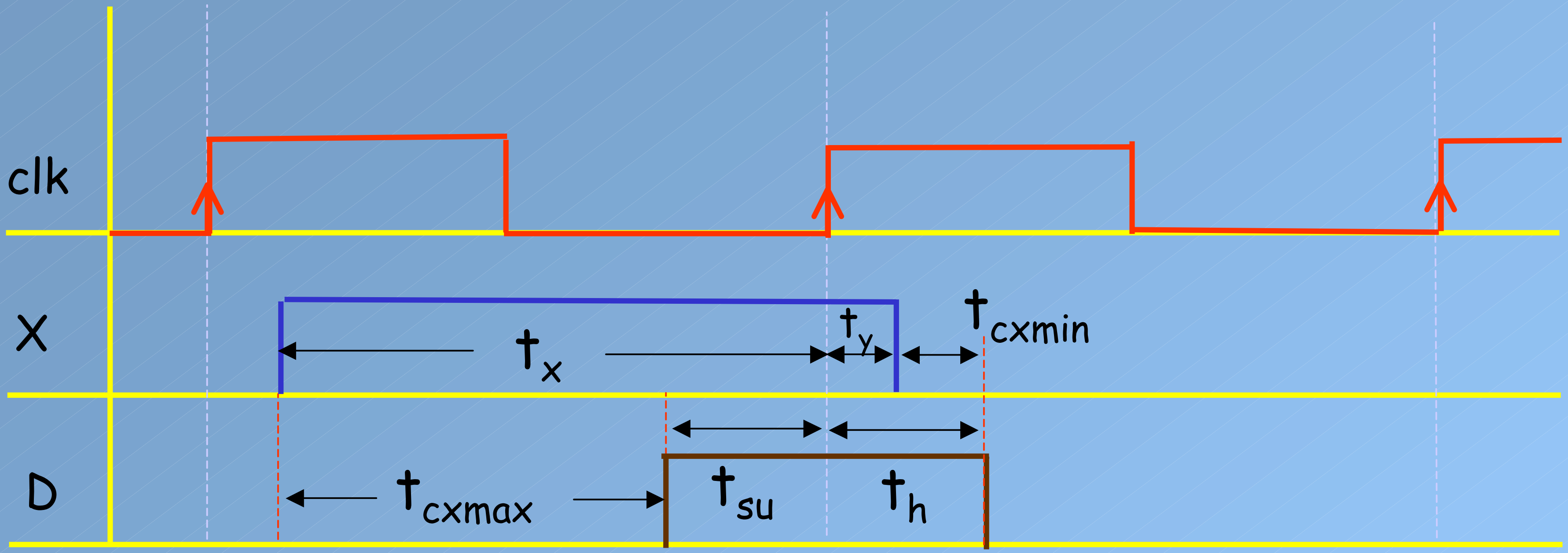


- t_{ck}: clock period; $t_{pmax} = \max(t_{plh}, t_{phl})$
- t_{cmax}: maximum propagation delay through combinational network
- Then, $t_{pmax} + t_{cmax} \leq t_{ck} - t_{su}$.

Setup and Hold-time Violations (1)

- Hold-time violation
  - If the change in Q fed back through the combinational network and caused D to change too soon after the clock edge.
  - $t_{pmin} + t_{cmin} \geq t_h$ ($t_{pmin} > t_h$ for normal flip-flops)
- Usually, setup or hold-time violation occurs if the input X to the network changes too close to the active edge of the clock.
  - We must make sure that an input change propagates to the flip-flop inputs such that setup time is satisfied before the active edge of the clock.

Setup and Hold-time Violations (2)

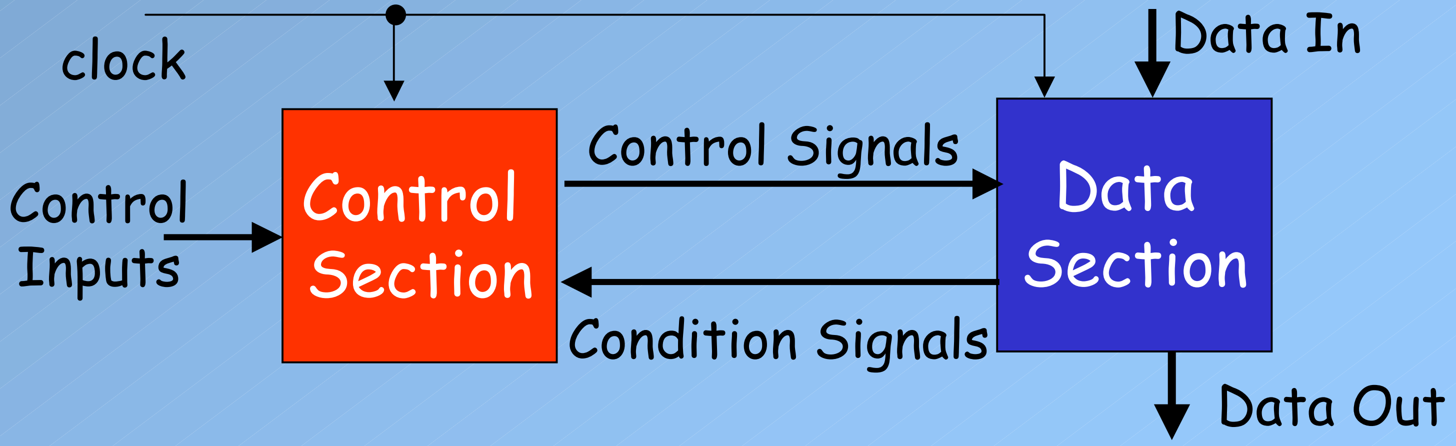


- For setup time: $t_x \geq t_{cxmax} + t_{su}$

- For hold-time: $t_y \geq t_h - t_{cxmin}$ or ($t_y + t_{cxmin} \geq t_h$)

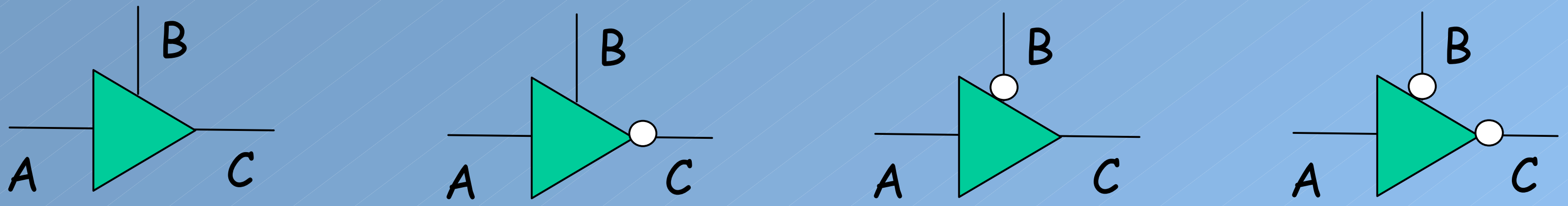
Synchronous Design

- A clock is used
  - To synchronize the operation of all flip-flops, registers, and counters in the system
  - All state changes occur following the active edge of the clock
  - The clock period must be long enough so that all flip-flop and register inputs will have time to stabilize before the next active edge of the clock



Tristate Logic and Buses

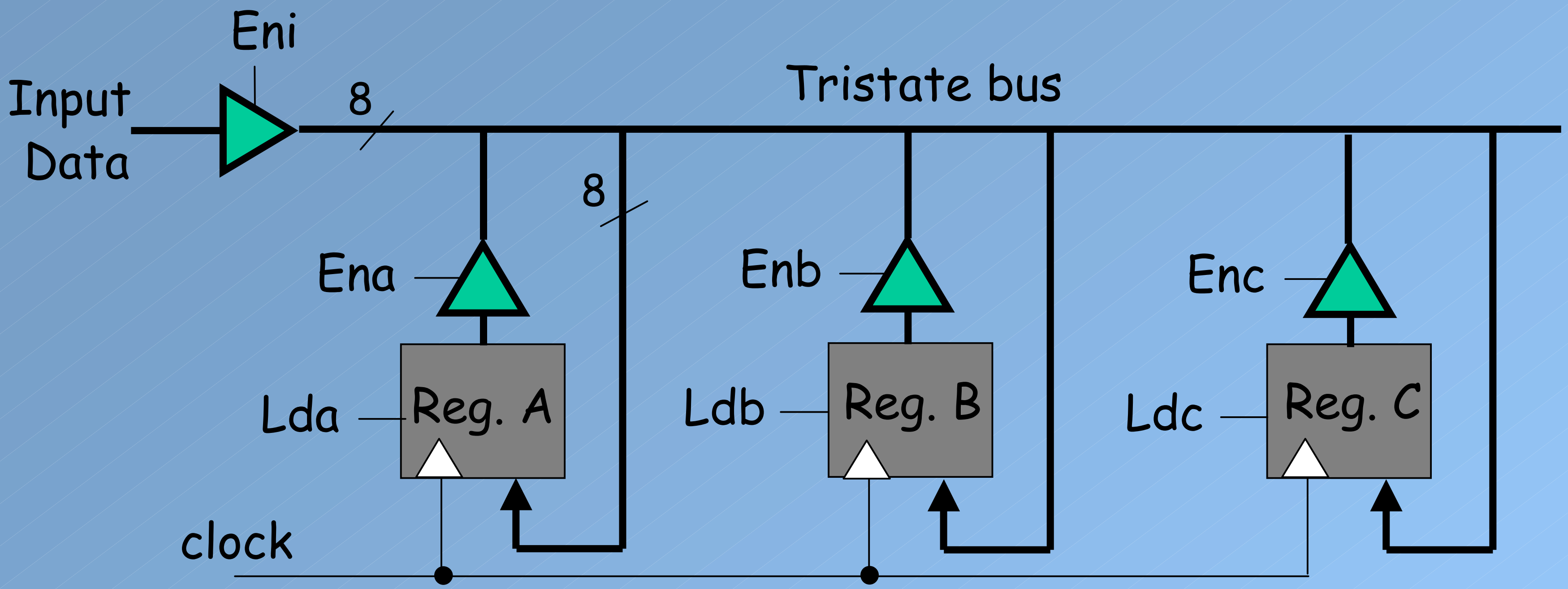
- Tristate buffers



B	A	C	B	A	C	B	A	C	B	A	C
0	0	Hi-Z	0	0	Hi-Z	0	0	0	0	0	1
0	1	Hi-Z	0	1	Hi-Z	0	1	1	0	1	0
1	0	0	1	0	1	1	0	Hi-Z	1	0	Hi-Z
1	1	1	1	1	0	1	1	Hi-Z	1	1	Hi-Z

- Hi-Z : high-impedance which is equivalent to an open circuit.
- We use tristate buffers to connect the outputs of more than one gates or flip-flops.

Tristate Bus



- If Enb = Ldc = 1 (all others 0), then the data in register B will be copied into register C when the active edge of the clock occurs.
- If Eni = Lda = Ldc = 1 then the input data will be loaded in registers A and C when the registers are clocked.