uOttawa

L'Université canadienne
Canada's university

Digital Logic Circuits

Dr. Voicu Groza

SITE Hall, Room 5017
562 5800 ext. 2159
Groza@EECS.uOttawa.ca

Université d'Ottawa | University of Ottawa

www.uOttawa.ca

Outline

- Boolean Algebra
- Logic Functions Representation
- Logic Gates
- Logic Functions Minimization using
  - Boolean algebra rules
  - Karnaugh – Veitch maps
- Combinational Circuits
- Sequential Circuits
  - Specification
  - Analysis
  - Synthesis / design

Logic function

$z_i = F_i(x_0, x_1, x_2, \ldots) \mid z_i \in \{0,1\},\ i = 0,1,2, \ldots m\text{-}1,\ x_k \in \{0,1\},\ k = 0,1,2, \ldots n\text{-}1$

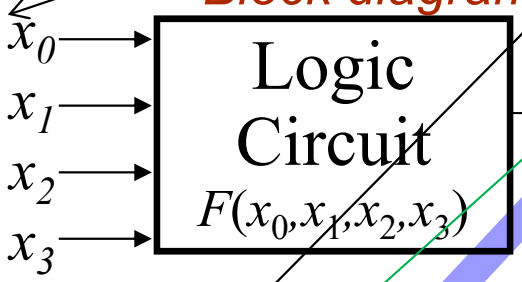
where F_i = *logic functions* ; x_k = *input variables*; z_i = *output variables*

where Logic 0 = FALSE and Logic 1 = TRUE

Logic function F can be **represented** by

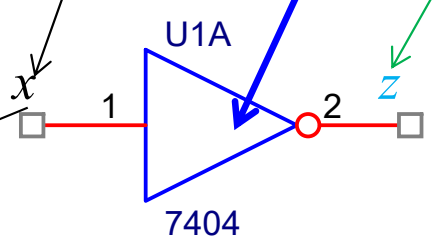
1. truth tables,
2. logic (Boolean) expressions,
3. logic diagrams (schematics) of the logic circuits that implement these functions.
4. K-maps.

Block diagram

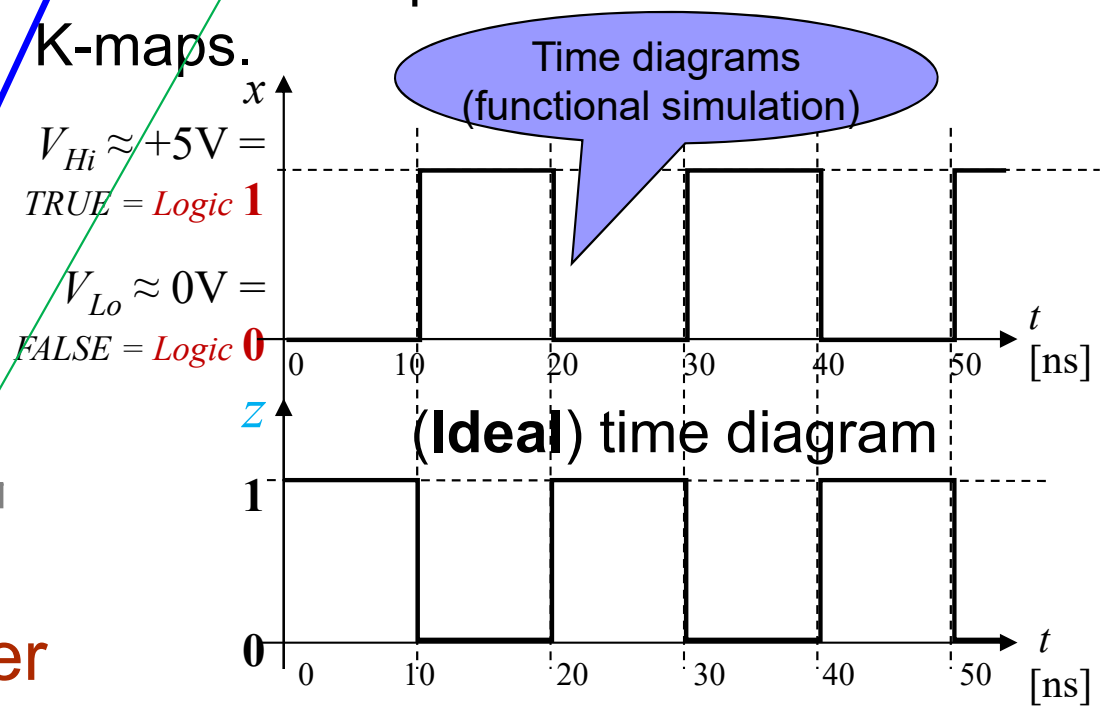


x	z
0	1
1	0

$z = \overline{x}$

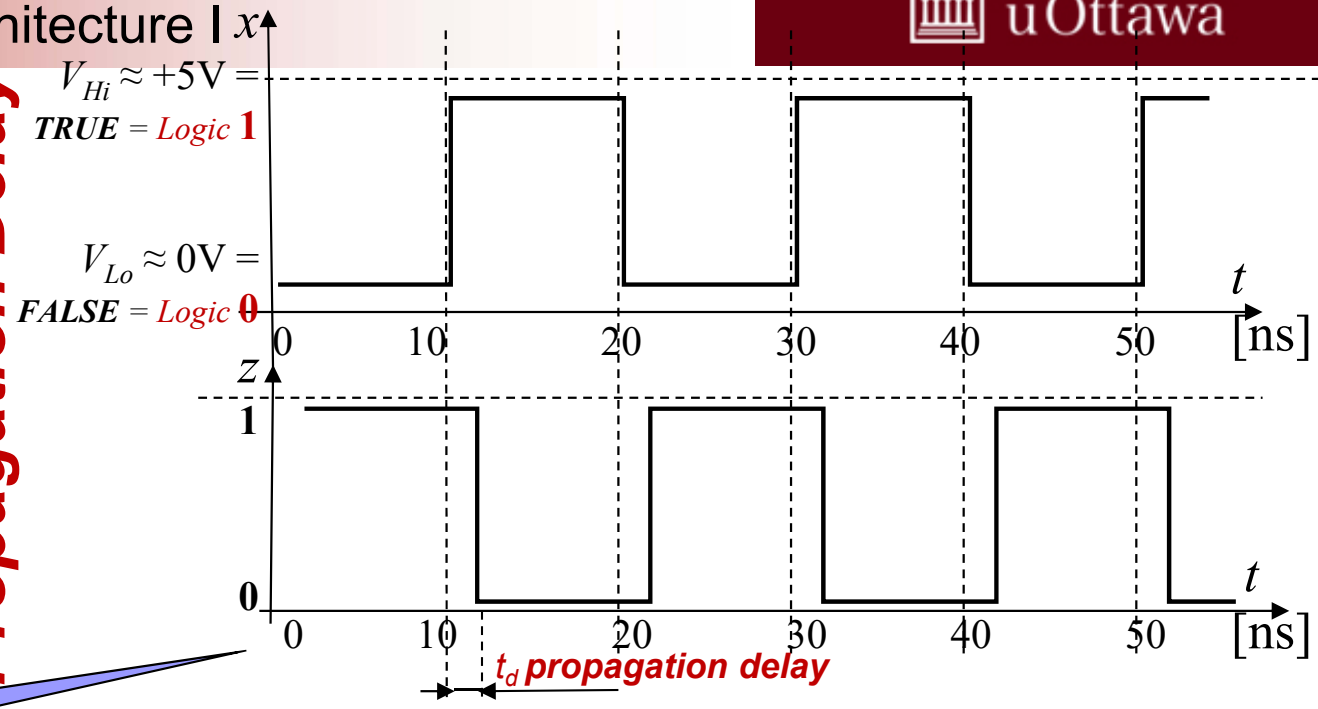


Logic Inverter



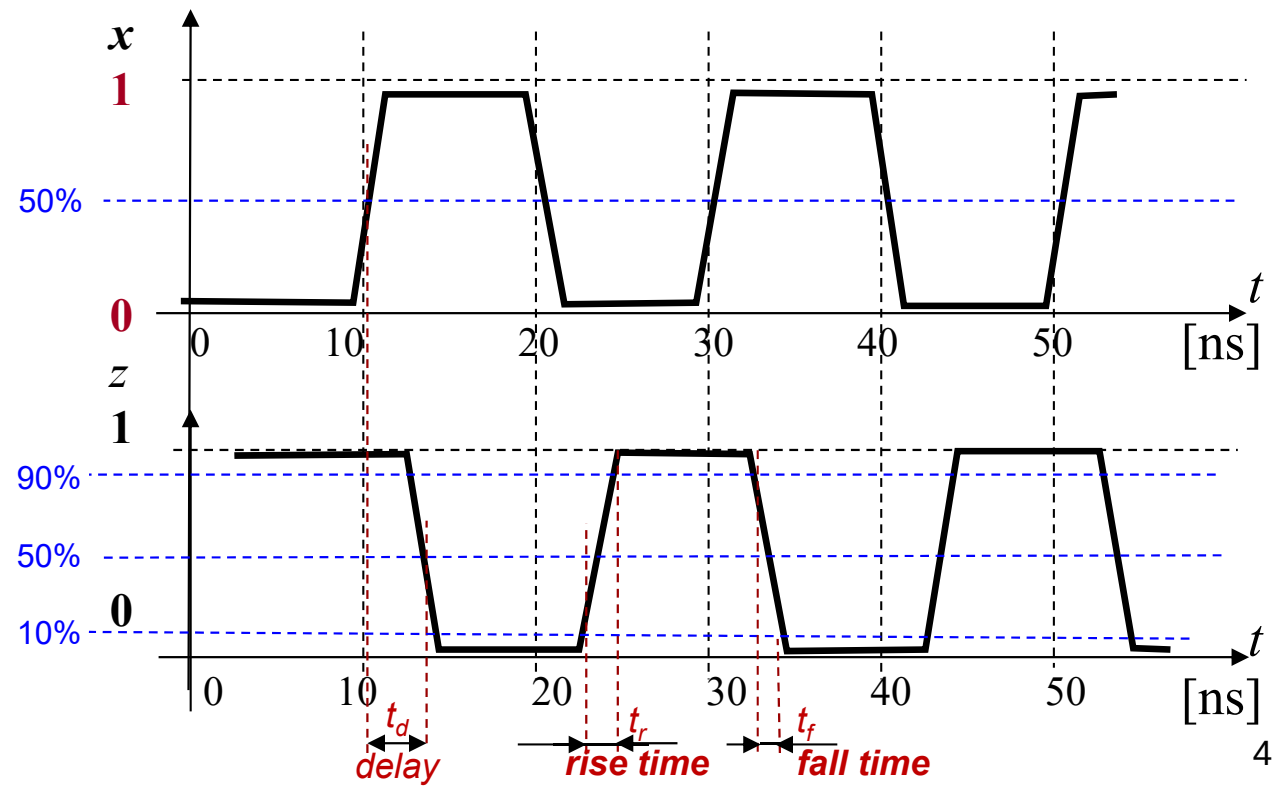
Approximations of Real Time-Diagrams

Propagation Delay

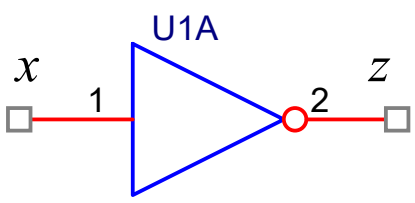


Time diagrams (timing simulation)

Rise / Fall Time



Logic Inverter



$z = \overline{x}$

x	z
0	1
1	0

BOOLEAN ALGEBRA over $\{0, 1\}$

$\{B, =, +, \bullet, \overline{}, 0, 1\}$, where:

B = set with at least 2 distinct elements;

$0, 1$ = 2 constants $\in B$

Binary operations:

$+$ logical OR

$\bullet$ logical AND

Unary operation: $\overline{}$ logical NOT

$=$ equivalence relationship, with usual properties:

– reflexivity: $(\forall\, x \in B) \rightarrow (x = x)$

– symmetry: $(\forall\, x, y \in B) \rightarrow (x = y \;\rightarrow\; y = x)$

– transitivity: $(\forall\, x, y, z \in B) \rightarrow (x = y \text{ and } y = z \rightarrow x = z)$

Parentheses are allowed

BOOLEAN ALGEBRA over Boolean Vectors

$B^n = \{(a_1, a_2, \ldots, a_n) \mid a_i \in \{0,1\}\}$

Let $a=(a_1, a_2, \ldots, a_n)$ and $b = (b_1, b_2, \ldots, b_n) \in B^n$

define

$a \vee b = (a_1 \vee b_1, a_2 \vee b_2, \ldots, a_n \vee b_n)$

$a \cdot b = (a_1 \cdot b_1, a_2 \cdot b_2, \ldots, a_n \cdot b_n)$

$\overline{a}=(\overline{a}_1, \overline{a}_2, \ldots, \overline{a}_n)$

then $\langle B^n, \vee, \cdot, \overline{}, \mathbf{0}, \mathbf{1} \rangle$ is a Boolean algebra, where,
$\mathbf{0} = (0,0, \ldots, 0)$ and $\mathbf{1} = (1,1, \ldots, 1)$

Basic Boolean Identities = axioms and theorems

BOOLEAN ALGEBRA AXIOMS

$B=\{0,1\}$	OR		AND	
Closeness:				
$\forall\, a, b \in B$	$a + b \in B$		$a \cdot b \in B$	
Identical element:				
$\forall\, a \in B$	$a + 0 = 0 + a = a$	[1]	$a \cdot 1 = 1 \cdot a = a$	[4]
Commutative:				
$\forall\, a, b \in B$	$a + b = b + a$	[9]	$a \cdot b = b \cdot a$	[10]
Distributive:				
$\forall\, a, b, c \in B$	$a+(b\cdot c) = (a+b) \cdot (a+c)$	[14]	$a \cdot (b + c) = a\cdot b + a\cdot c$	[13]
Complement:				
$\forall\, a \in B, \exists\, \bar{a}$	$a + \bar{a} = 1$	[7]	and $a \cdot \bar{a} = 0$	[8]
Associative:				
$\forall\, a, b, c \in B$	$(a + b) + c = a + (b + c)$	[11]	$(a \cdot b) \cdot c = a\cdot (b \cdot c)$	[12]

NOTE: the equation numbers in square brackets, e.g. [1], coresspond to the Table 1-1 of Mano's textbook

BOOLEAN ALGEBRA Fundamental Theorems

Idempotency	*Theorem 1a*	$a + a = a$	*Theorem 1b* $a \cdot a = a$ [6]
Null	*Theorem 2a*	$a + 1 = a+(a+\bar{a})=(a+a)+\bar{a}=a+\bar{a}=1$ [7] [T1a]	*Theorem 2b* $a \cdot 0 = 0$ [2]
Absorption	*Theorem 6a*	$a + a\cdot b = a(b+\bar{b})+ab = a$ [14]	*Theorem 6b* $a \cdot (a + b) = a$
	Theorem 7a	$a + \bar{a}\cdot b = (a+\bar{a})(a+b) = a+b$	*Theorem 7b* $a \cdot (a + b) = a \cdot b$
Uniting	*Theorem 8a*	$a\cdot b + a\cdot\bar{b} = a\cdot(b+\bar{b}) = a\cdot 1 = a$	*Theorem 8b* $(+ b)(a + b) = a$

Consensus: 13. $(a \bullet b) + (b \bullet c) + (\bar{a} \bullet c) = a \bullet b + \bar{a} \bullet c$

13D. $(a + b) \bullet (b + c) \bullet (\bar{a} + c) = (a + b) \bullet (\bar{a} + c)$

Factoring: 12. $(a + b) \bullet (\bar{a} + c) = a \bullet c + \bar{a} \bullet b$

12D. $a \bullet b + \bar{a} \bullet c = (a + c) \bullet (\bar{a} + b)$

Theorem 3 [17] $\bar{\bar{a}} = a$

<u>De Morgan's Theorems</u>:

Theorem 5a [15] $\overline{a+b} = \bar{a} \bullet \bar{b}$

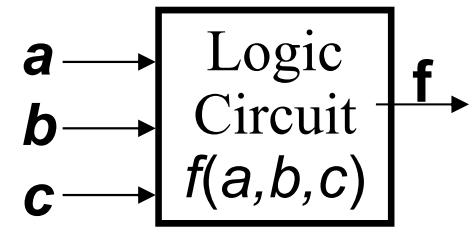
Theorem 5b [16] $\overline{a \bullet b} = \bar{a} + \bar{b}$

Theorem 5a' $\overline{a_1 + a_2 + \ldots + a_n} = \bar{a_1} \bullet \bar{a_2} \bullet \ldots \bullet \bar{a_n}$

Theorem 5b' $\overline{a_1 \bullet a_2 \bullet .. \bullet a_k} = \bar{a_1} + \bar{a_2} + \ldots + \bar{a_k}$

<u>NOTE</u>: equation numbers in square brackets, e.g. [5], correspond to Table 1-1 of Mano's textbook

Logic Functions Representation (1)



Truth table is the unique signature of a Boolean function

- Many alternative gate realizations may have the same truth table
- Truth table contains **logic** variables (inputs - ***a, b, c***, and output **f**) and their values ("0" for FALSE, and "1" for TRUE: *they are not numbers, but logic constants*!).
- IF we map ***a*** to 2^2, ***b*** to 2^1, ***c*** to 2^0, we can interpret ***abc*** as binary numbers and then we can organize all logic combinations in ascending order of these numbers.

	a	*b*	*c*	f
	2^2	2^1	2^0	
(0)	0	0	0	0
(1)	0	0	1	0
(2)	0	1	0	0
(3)	0	1	1	1
(4)	1	0	0	0
(5)	1	0	1	0
(6)	1	1	0	0
(7)	1	1	1	1

Proving Theorems (Perfect Induction)

	AND rules (identities)
(2)	$0 \cdot X = 0$
(4)	$1 \cdot X = X$
(6)	$X \cdot X = X$
(8)	$X \cdot \overline{X} = 0$
(10)	$X \cdot Y = Y \cdot X$
(12)	$X \cdot (Y \cdot Z) = (X \cdot Y) \cdot Z$
(13)	$X \cdot (Y + Z) = X \cdot Y + X \cdot Z$
(16)	$\overline{X \cdot Y} = \overline{X} + \overline{Y}$

" Proof ": $X \cdot (Y+Z) = X \cdot Y + X \cdot Z$

X	Y	Z	Y+Z	X·(Y+Z)	X·Y	X·Z	X·Y+X·Z
0	0	0	0	**0**	0	0	**0**
0	0	1	1	**0**	0	0	**0**
0	1	0	1	**0**	0	0	**0**
0	1	1	1	**0**	0	0	**0**
1	0	0	0	**0**	0	0	**0**
1	0	1	1	**1**	0	1	**1**
1	1	0	1	**1**	1	0	**1**
1	1	1	1	**1**	1	1	**1**

NOTE: equation numbers, e.g. (2), are from Table 1-1 of Mano's textbook

Proving Theorems (Perfect Induction)... continued

	OR rules (identities)
(1)	$0 + X = X$
(3)	$1 + X = 1$
(5)	$X + X = X$
(7)	$X + \overline{X} = 1$
(17)	$\overline{(\overline{X})} = X$
(9)	$X + Y = Y + X$
(11)	$X + (Y + Z) = (X + Y) + Z$
(14)	$X + Y \cdot Z = (X+Y) \cdot (X+Z)$
(15)	$\overline{X} + \overline{Y} = \overline{X \cdot Y}$

" Proof ":

X	Y	Z	$X + Y \cdot Z$	$(X+Y) \cdot (X+Z)$
0	0	0	0	0
0	0	1	0	0
0	1	0	0	0
0	1	1	1	1
1	0	0	1	1
1	0	1	1	1
1	1	0	1	1
1	1	1	1	1

NOTE: equation numbers, e.g. (1), are from Table 1-1 of Mano's textbook

DeMorgan's Theorems

$$\overline{a \vee b} = \overline{a} \wedge \overline{b}$$

$$\overline{a \wedge b} = \overline{a} \vee \overline{b}$$

$$\overline{a + b} = \overline{a} \cdot \overline{b}$$

$$\overline{a \cdot b} = \overline{a} + \overline{b}$$

a	b	$\overline{a} \cdot \overline{b}$	$\overline{a + b}$	$\overline{a} + \overline{b}$	$\overline{a \cdot b}$
0	0	1	1	1	1
0	1	0	0	1	1
1	0	0	0	1	1
1	1	0	0	0	0

- These equations can be generalized

$$\overline{x_1 \vee x_2 \vee \ldots \vee x_n} = \overline{x_1} \cdot \overline{x_2} \cdot \ldots \cdot \overline{x_n}$$

$$\overline{x_1 \cdot x_2 \cdot \ldots \cdot x_n} = \overline{x_1} \vee \overline{x_2} \vee \ldots \vee \overline{x_n}$$

All Logic Functions of Two Variables

- 16 possible functions of 2 input variables:
  - 2**(2**n) = 2^{2^n} functions of n inputs

x	y	F_0	F_1	F_2	F_3	F_4	F_5	F_6	F_7	F_8	F_9	F_{10}	F_{11}	F_{12}	F_{13}	F_{14}	F_{15}
0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
0	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
1	0	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1

Map 1st row to 2^3

Map last Row to 2^0

0

X and Y

X

Y

X xor Y

X or Y

X nor Y
not (X or Y)

X = Y

not Y

not X

X nand Y
not (X and Y)

1

Logic Functions Representation (2)

Logic (Boolean) Expressions

- **Normal forms**
  - Disjunctive **normal** form = Sum of Products (**SoP**): each *product term* (containing all or some of the variables and/or their negations) specifies when the function is 1 (TRUE)
  - Conjunctive **normal** form = Product of Sums (**PoS**): each *sum factor* (containing all or some of the variables and their negations) specifies when function is 0 (FALSE).
- **Canonical forms** - standard normal forms for a Boolean expression, direct mapping the truth table and gives a unique algebraic signature; each term contains all the variables.

1. Sum-of-Products (**SoP**) Canonical Forms – also known as *canonical disjunctive normal form* or as *minterm expansion*
2. Product of Sums (**PoS**) Canonical Forms / *canonical conjunctive normal form* / *maxterm expansion*

Minterms = one product term functions

- AND-ed product of literals – input combination for which output is true ("1")
- Each variable appears exactly once, in true or inverted form (but not both!)
- If we map A to 2^2, B to 2^1 and C to 2^0, the minterm's index represents the value of the coressponding logical combination where it is 1.

ABC	$m_0 = \overline{A}\,\overline{B}\,\overline{C}$	$m_1 = \overline{A}\,\overline{B}C$	$m_2 = \overline{A}B\overline{C}$	$m_3 = \overline{A}BC$	$m_4 = A\overline{B}\,\overline{C}$	$m_5 = A\overline{B}C$	$m_6 = AB\overline{C}$	$m_7 = ABC$
000	1	0	0	0	0	0	0	0
001	0	1	0	0	0	0	0	0
010	0	0	1	0	0	0	0	0
011	0	0	0	1	0	0	0	0
100	0	0	0	0	1	0	0	0
101	0	0	0	0	0	1	0	0
110	0	0	0	0	0	0	1	0
111	0	0	0	0	0	0	0	1

Sum-of-Products (SoP) Canonical Form

- also known as disjunctive canonical form or minterm expansion

is the most complex expression of SoP or disjunctive normal form

- product term (or minterm)
  - AND-ed product of literals – input combination for which output is true
  - Each variable appears exactly once, in true or inverted form (but, of course, not both!)

F in canonical disjunctive form: $F(A,B,C) = \Sigma m(1,3,5,6,7) =$

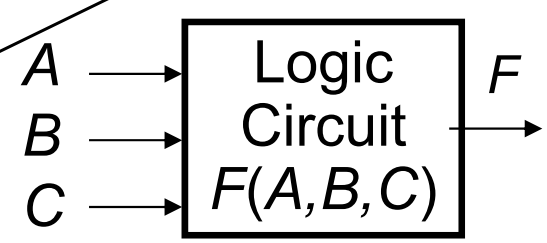
$$m_1 + m_3 + m_5 + m_6 + m_7$$

$$F = 001 \quad 011 \quad 101 \quad 110 \quad 111$$

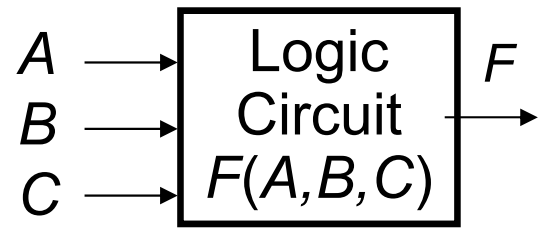
$$F = A'B'C + A'BC + AB'C + ABC' + ABC$$

A	B	C	minterms
0	0	0	A'B'C' m_0
0	0	1	A'B'C m_1
0	1	0	A'BC' m_2
0	1	1	A'BC m_3
1	0	0	AB'C' m_4
1	0	1	AB'C m_5
1	1	0	ABC' m_6
1	1	1	ABC m_7

	A	B	C	F
	2^2	2^1	2^0	
(0)	0	0	0	0
(1)	0	0	1	1
(2)	0	1	0	0
(3)	0	1	1	1
(4)	1	0	0	0
(5)	1	0	1	1
(6)	1	1	0	1
(7)	1	1	1	1



Logic functions representation using *Canonical PoS form*



Maxterm = OR-ed sum of literals in which each variable appears exactly once in either true or complemented form, but not both!

***P-o-S* = Conjunctive Normal Form can be obtained by:**

A. Finding the truth table rows where F is **0**

- 0 in input column implies true literal
- 1 in input column implies complemented literal

$F(A,B,C) = \pi M(0,2,4) = (A+B+C)\ (A+B'+C)\ (A'+B+C)$

ABC	M_0	M_1	M_2	M_3	M_4	M_5	M_6	M_7	
000	**0**	1	1	1	1	1	1	1	$A+B+C=M_0$
001	1	**0**	1	1	1	1	1	1	$A+B+\overline{C}=M_1$
010	1	1	**0**	1	1	1	1	1	$A+\overline{B}+C=M_2$
011	1	1	1	**0**	1	1	1	1	$A+\overline{B}+\overline{C}=M_3$
100	1	1	1	1	**0**	1	1	1	$\overline{A}+B+C=M_4$
101	1	1	1	1	1	**0**	1	1	$\overline{A}+B+\overline{C}=M_5$
110	1	1	1	1	1	1	**0**	1	$\overline{A}+\overline{B}+C=M_6$
111	1	1	1	1	1	1	1	**0**	$\overline{A}+\overline{B}+\overline{C}=M_7$

ABC	F	F'
000	**0**	1
001	1	0
010	**0**	1
011	1	0
100	**0**	1
101	1	0
110	1	0
111	1	0

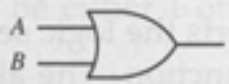
or by

B. Negating the disjunctive form (SoP) of the negated function (F'):

1. Start from the Sum-of-Products of
   F' = A'B'C' + A'BC' + AB'C'
2. Then apply de Morgan's:
   F = (F')' = (A'B'C' + A'BC' + AB'C')'
   = (A+B+C) (A+B'+C) (A'+B+C)

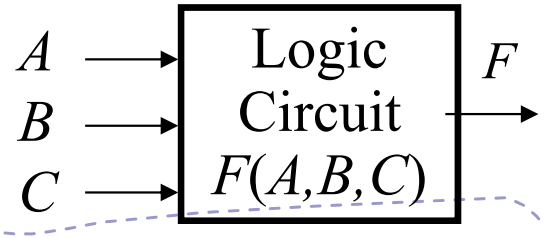
Logic Functions Representation (3)

Digital Logic Gates

Name	Graphic symbol	Algebraic function	Truth table
AND		$x = A \cdot B$ or $x = AB$	A B | x 0 0 | 0 0 1 | 0 1 0 | 0 1 1 | 1
OR		$x = A + B$	A B | x 0 0 | 0 0 1 | 1 1 0 | 1 1 1 | 1
Inverter		$x = A'$	A | x 0 | 1 1 | 0
Buffer		$x = A$	A | x 0 | 0 1 | 1

Name	Graphic symbol	Algebraic function	Truth table
NAND		$x = (AB)'$	A B | x 0 0 | 1 0 1 | 1 1 0 | 1 1 1 | 0
NOR		$x = (A + B)'$	A B | x 0 0 | 1 0 1 | 0 1 0 | 0 1 1 | 0
Exclusive-OR (XOR)		$x = A \oplus B$ or $x = A'B + AB'$	A B | x 0 0 | 0 0 1 | 1 1 0 | 1 1 1 | 0

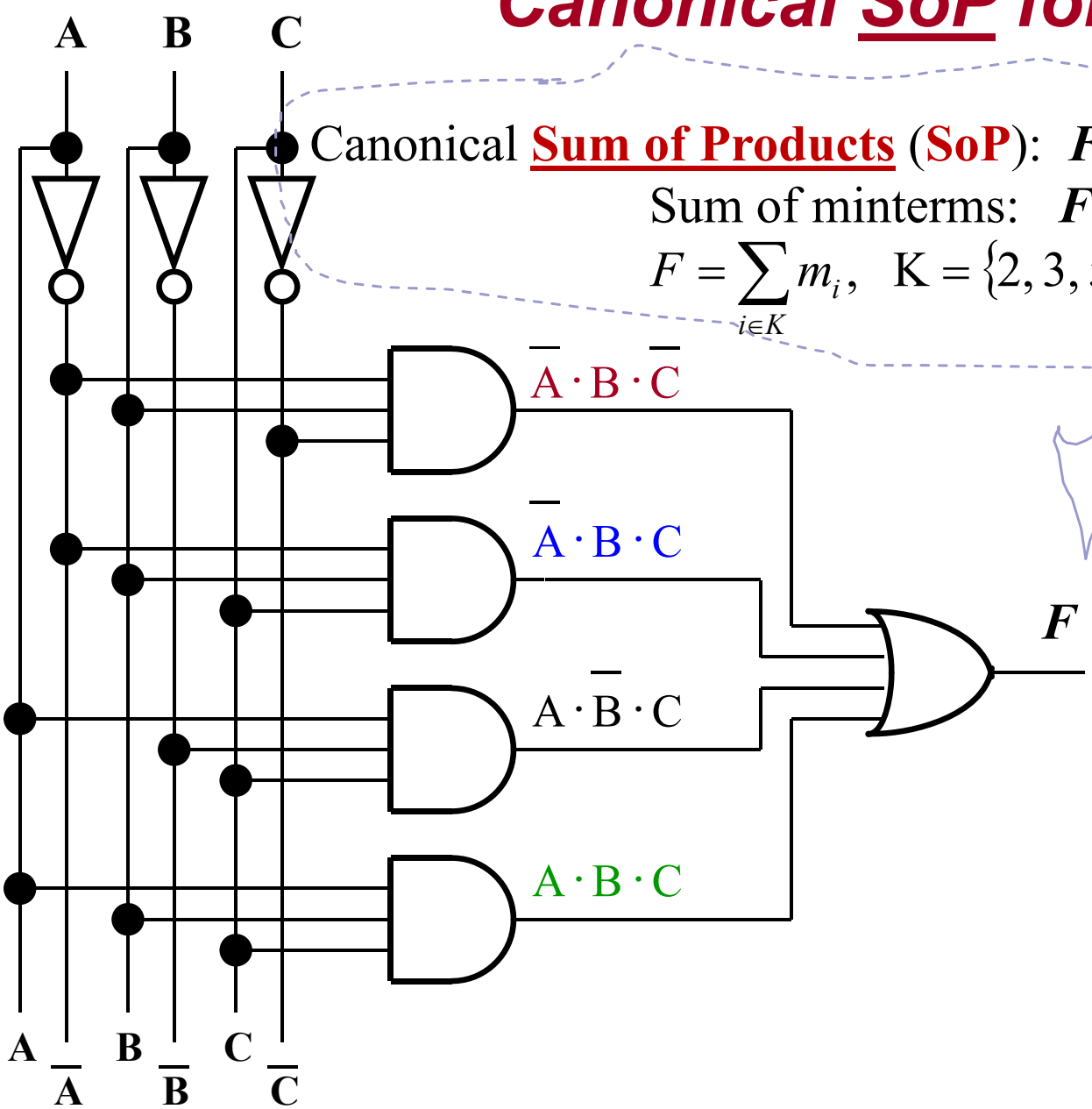
Logic functions representation using *Canonical SoP form*



Canonical **Sum of Products** (**SoP**): $F = \bar{A}B\bar{C} + \bar{A}BC + A\bar{B}C + ABC$

Sum of minterms: $F = m_2 + m_3 + m_5 + m_7$

$F = \sum_{i \in K} m_i, \quad K = \{2, 3, 5, 7\}$; $F = \Sigma\,(2, 3, 5, 7)$



min-term	A	B	C	F
(m_0)	0	0	0	0
(m_1)	0	0	1	0
(m_2)	0	1	0	**1**
(m_3)	0	1	1	**1**
(m_4)	1	0	0	0
(m_5)	1	0	1	**1**
(m_6)	1	1	0	0
(m_7)	1	1	1	**1**

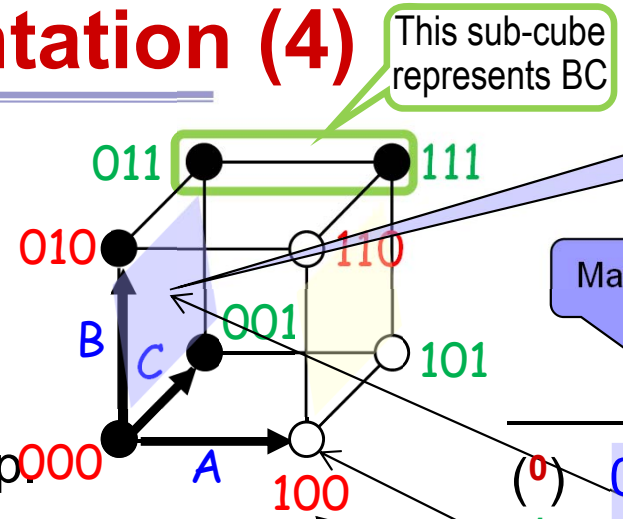
Map to 2^2

Map to 2^0

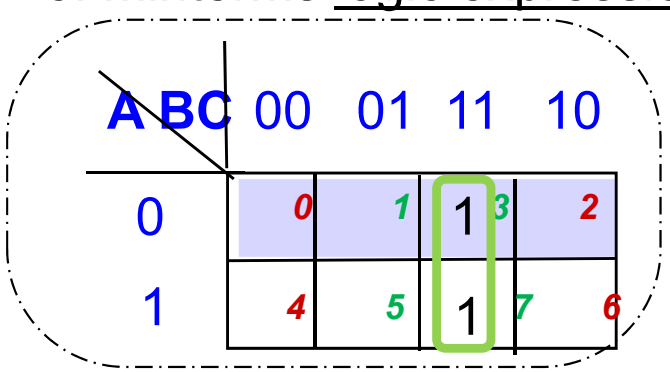
Logic functions Representation (4)

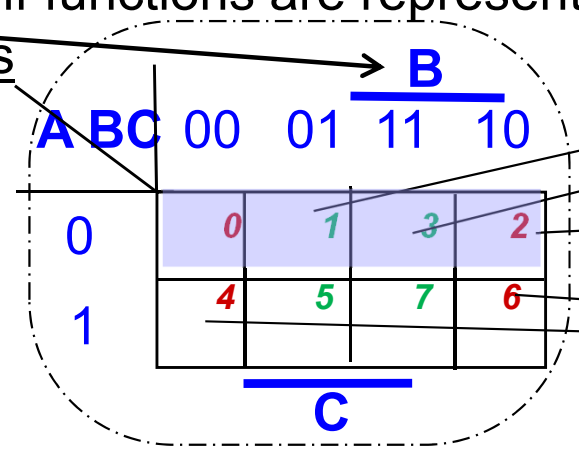
Karnaugh Maps

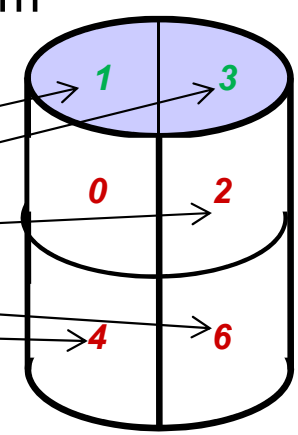
- **Karnaugh map** => graphical representation of a truth table of a logic function.
- Each line in the truth table (minterm) corresponds to a square in the Karnaugh map.
- The Karnaugh map squares are labeled so that horizontally or vertically adjacent squares differ only in one variable. (*Each square in the top row is considered to be adjacent to a corresponding square in the bottom row. Each square in the left most column is considered to be adjacent to a corresponding square in the right most column.*)
- This format is recommended if functions are represented by sum of minterms logic expressions



	Map to 2^2		Map to 2^0	
	A	B	C	F
(0)	0	0	0	1
(1)	0	0	1	1
(2)	0	1	0	1
(3)	0	1	1	1
(4)	1	0	0	0
(5)	1	0	1	0
(6)	1	1	0	0
(7)	1	1	1	1







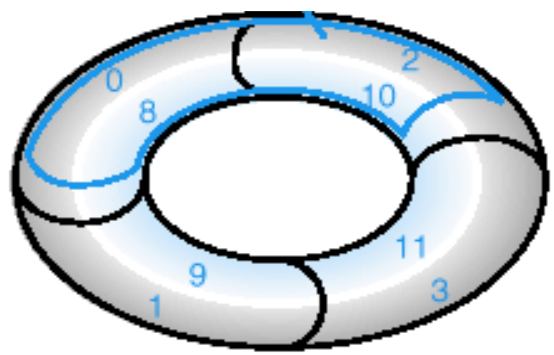
4 variable logic functions representation on Karnaugh maps

Map to 2^3 2^2 2^1 2^0

	A	B	C	D	F
(0)	0	0	0	0	…
(1)	0	0	0	1	…
(2)	0	0	1	0	…
(3)	0	0	1	1	…
(4)	0	1	0	0	…
(5)	0	1	0	1	…
(6)	0	1	1	0	…
(7)	0	1	1	1	…
(8)	1	0	0	0	…
(9)	1	0	0	1	…
(10)	1	0	1	0	…
(11)	1	0	1	1	…
(12)	1	1	0	0	…
(13)	1	1	0	1	…
(14)	1	1	1	0	…
(15)	1	1	1	1	…

Numerical equivalent of logic combinations ABCD read as binary values with A = msb and D = lsb

The minterms are listed by an equivalent decimal number for easy reference



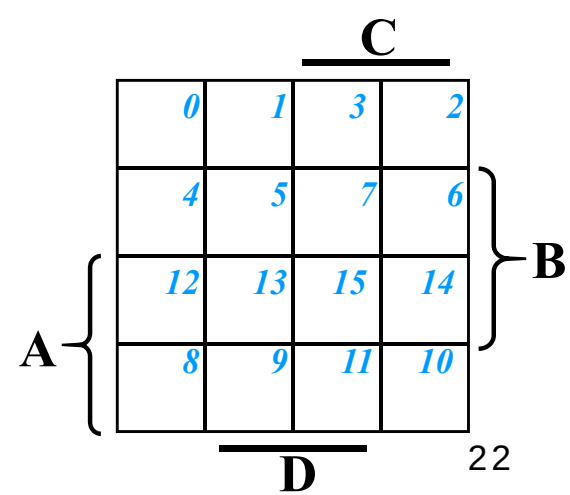
Map to 2^3 2^2 2^1 2^0

AB \ CD	00	01	11	10
00	0	1	3	2
01	4	5	7	6
11	12	13	15	14
10	8	9	11	10

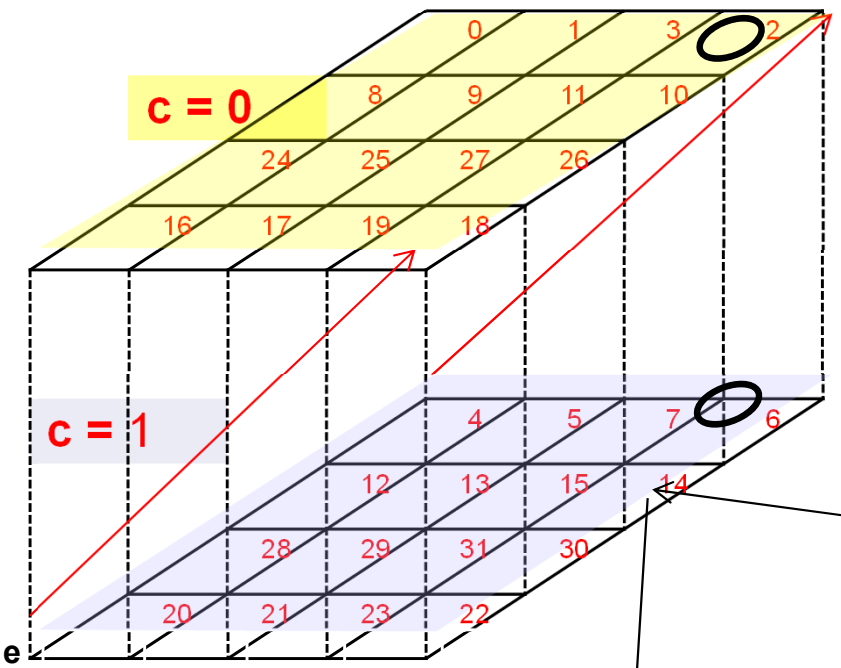
C, B, A, D

Recommended for functions represented by sum of minterms logic expressions

AB \ CD	00	01	11	10
00	0	1	3	2
01	4	5	7	6
11	12	13	15	14
10	8	9	11	10



5 variable logic functions representation on K-maps

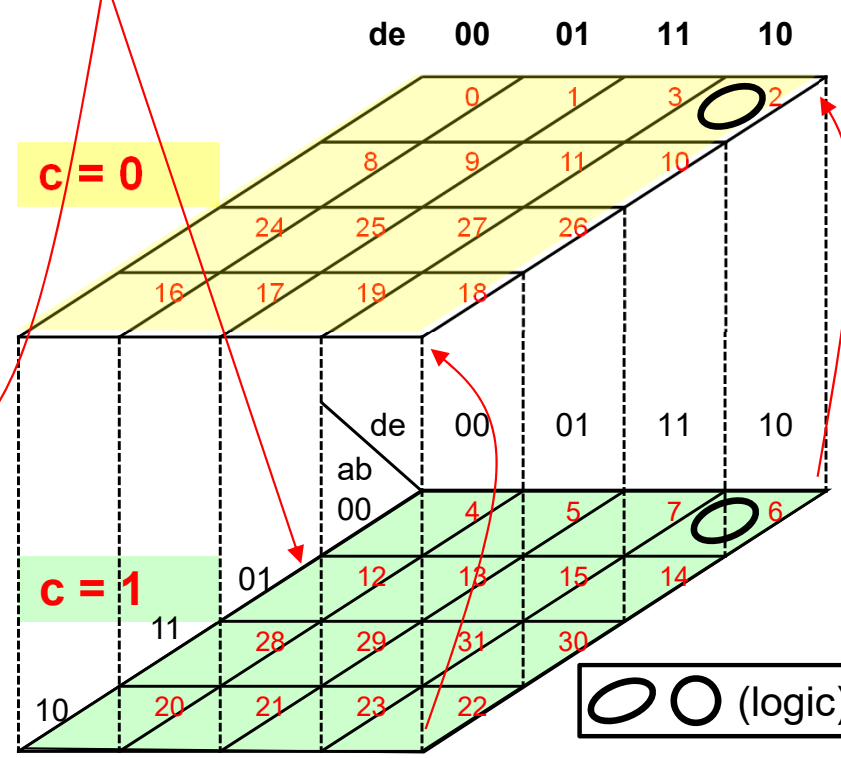


The 2 K-maps are obtained from the 2 3D parallel adjacent squares by

1. translation of the blue square or
2. rotation of the green square.

a b \ cd e	000	001	011	010	100	101	111	110
00	0	1	3	2	4	5	7	6
01	8	9	11	10	12	13	15	14
11	24	25	27	26	28	29	31	30
10	16	17	19	18	20	21	23	22

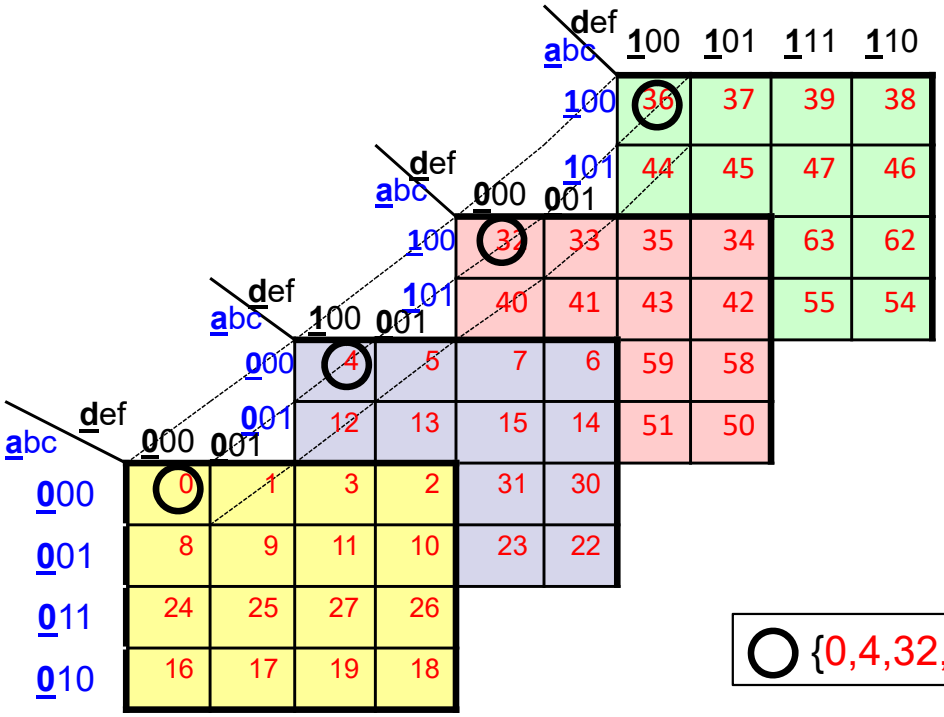
a b \ cd e	000	001	011	010	100	101	111	110
00	0	1	3	2	6	7	5	4
01	8	9	11	10	14	15	13	12
11	24	25	27	26	30	31	29	28
10	16	17	19	18	22	23	21	20



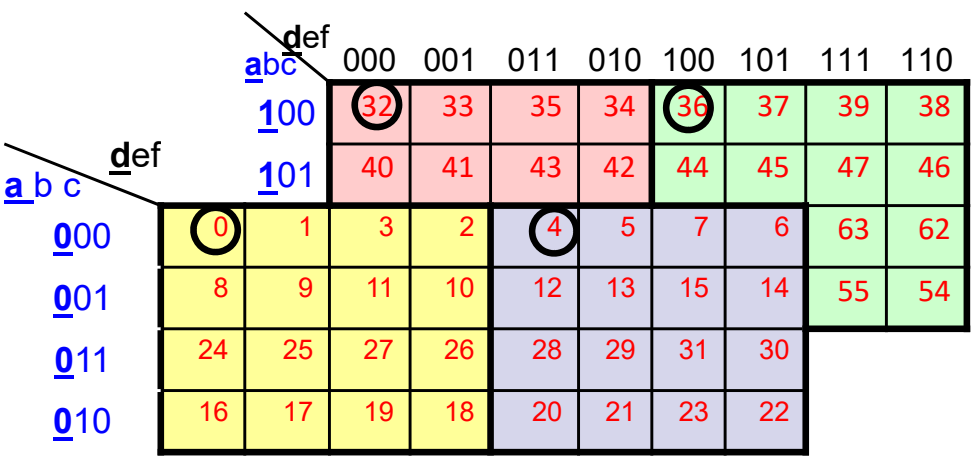
	a	b	c	d	e	Y
(0)	0	0	0	0	0	
(1)	0	0	0	0	1	
(2)	0	0	0	1	0	
(3)	0	0	0	1	1	
(4)	0	0	1	0	0	
(5)	0	0	1	0	1	
(6)	0	0	1	1	0	
(7)	0	0	1	1	1	
(8)	0	1	0	0	0	
(9)	0	1	0	0	1	
(10)	0	1	0	1	0	
(11)	0	1	0	1	1	
(12)	0	1	1	0	0	
(13)	0	1	1	0	1	
(14)	0	1	1	1	0	
(15)	0	1	1	1	1	
(16)	1	0	0	0	0	
(17)	1	0	0	0	1	
(18)	1	0	0	1	0	
(19)	1	0	0	1	1	
(20)	1	0	1	0	0	
(21)	1	0	1	0	1	
(22)	1	0	1	1	0	
(23)	1	0	1	1	1	
(24)	1	1	0	0	0	
(25)	1	1	0	0	1	
(26)	1	1	0	1	0	
(27)	1	1	0	1	1	
(28)	1	1	1	0	0	
(29)	1	1	1	0	1	
(30)	1	1	1	1	0	
(31)	1	1	1	1	1	

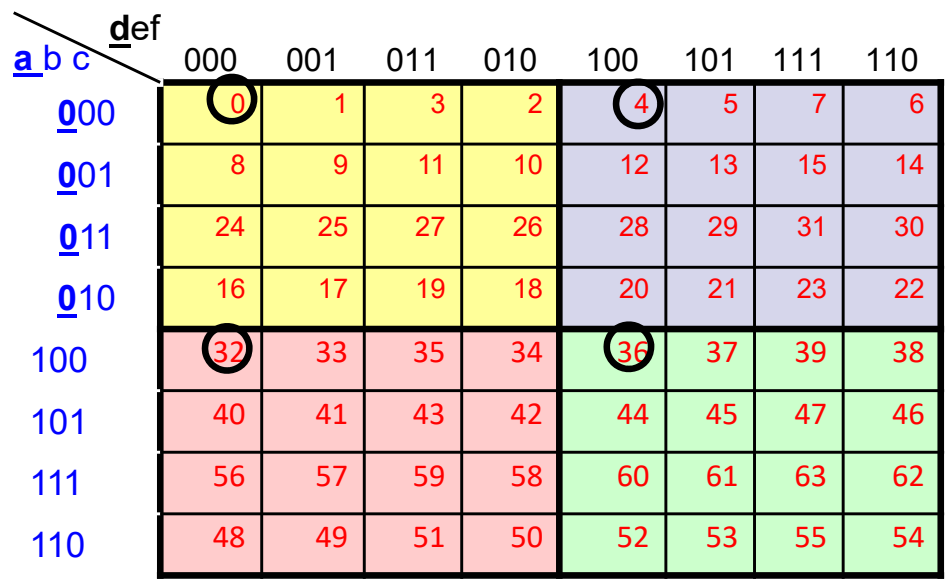
(logic) adjacent cells

6 variable logic functions representation on K-maps



○ {0,4,32,36} = adjacent cells



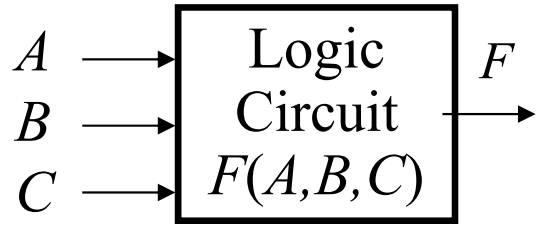


Logic Minimization

- Logic minimization aims to reduce
  - the number of gates (resulting from reduction of the number of terms)
  - the number of inputs per gate (resulting from fewer variables per term)
- Cost function: (proportional with) the number of gates inputs
- Simplifying logic function => Logic circuit minimization
- The minimization will reduce cost, efficiency and power consumption.
- Manual procedures:
  1. Boolean Algebraic manipulation
  2. Karnaugh maps (K-maps)
- Automatic procedures for:
  a) Finding all Prime Implicants (PI)
  b) Finding the minimum cover from the PI's list

Simplifying logic functions using

1) Boolean algebra rules

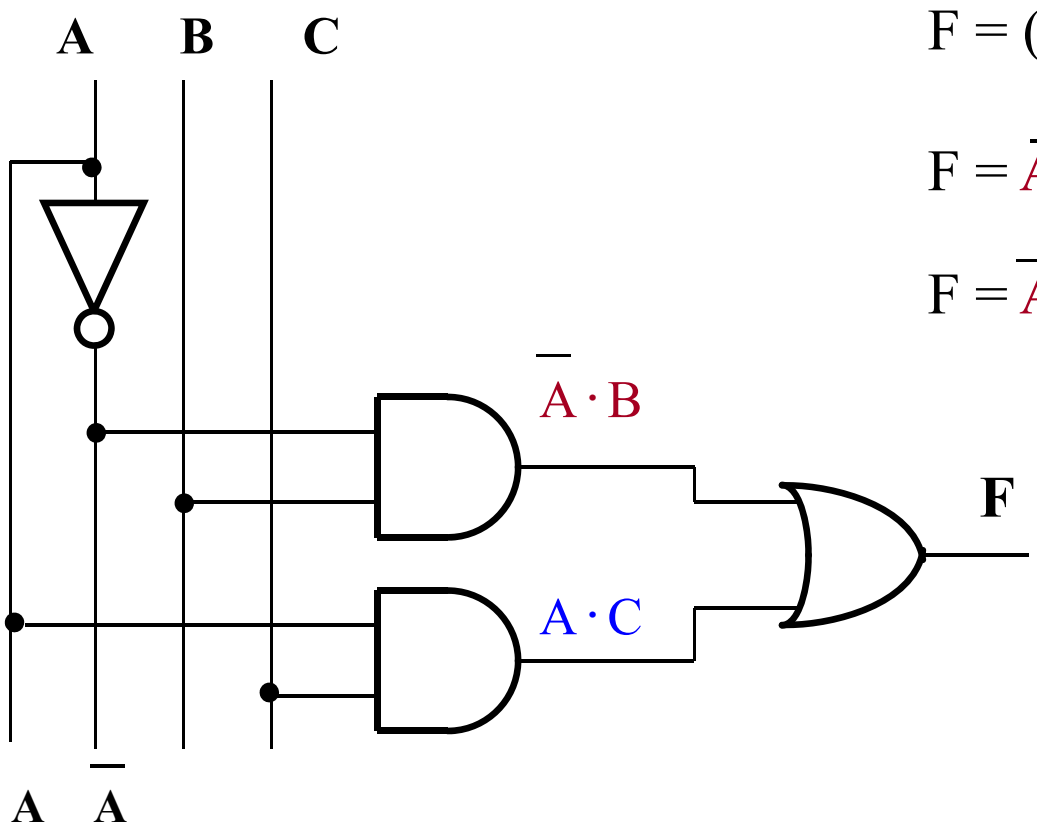


$$F = \bar{A}B\bar{C} + \bar{A}BC + A\bar{B}C + ABC$$

$$F = (\bar{A}B\bar{C} + \bar{A}BC) + (A\bar{B}C + ABC)$$

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

$$F = \bar{A}B(\underbrace{\bar{C} + C}_{1}) + AC(\underbrace{\bar{B} + B}_{1})$$



$$\mathbf{F = \bar{A}B + AC}$$

The Uniting Theorem

- Key tool for simplification: A (B' + B) = A (*Theorem 8a/slide 8*) where A can be a product term of any other input variables
- Essence of simplification:
  - Find two element subsets of the ON-set where only one variable changes its value – this single varying variable can be eliminated and a single product term used to represent both elements

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

F = A'B'+AB' = (A'+A)B' = B'

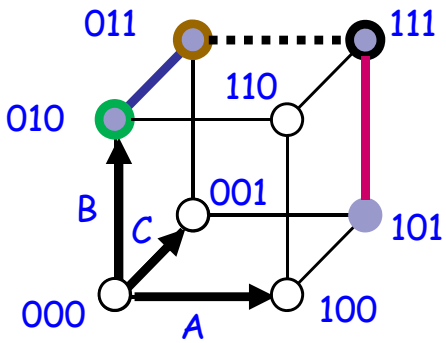
B has the same value (0) in both on-set rows
=> B remains in the expression of the product term

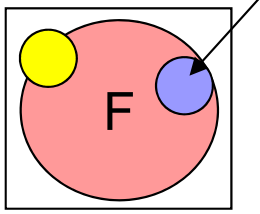
A has different values in the two rows
=> A is eliminated

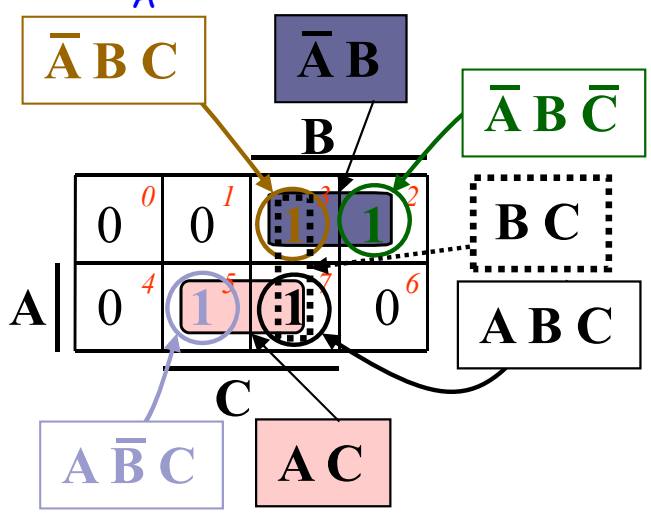
  - To facilitate comparison of combinations where only one variable changes its value, the function is represented on K-maps

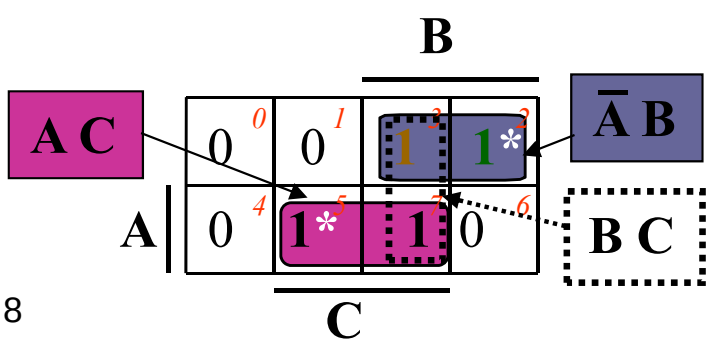
Implicants & Graphic Representation

$F = \sum(2,3,5,7)$









Given a logic function $z = f(x_0, x_1, x_2, \ldots)$

Implicant = product term that, if equal to 1, implies $f = 1$

- whenever the implicant is $1 \Rightarrow f = 1$
- f can be 1 other times, as well: it may be implied by other implicants, other than the one at hand

– from the K-map point of view, an implicant is a rectangle of 1, 2, 4, 8, … (any power of 2) 1's

Prime Implicant (PI)

= the largest possible *k-cube for which f* = 1

= cannot be totally covered by another implicant

(e.g., AC, **BC**, A'B)

Each "1" which is covered only by a single prime implicant is marked with a star (*)

Essential Prime Implicant (EPI) = a prime implicant that contains at least one "**1***"

A **minimum cover** of a logic function has to contain all its essential prime implicants

- *Theorem: The minimal SoP of a function is a sum of prime implicants*

Logic Function minimization using 2) K-maps

- To obtain the SoP of F_{min} (the minimized F), one has to find the minimum set of PI's which covers F as follows:
  1. Represent the given function F on a K-map
  2. Find the PI set of the function F
  3. Mark with * all K-map cells that contain a 1 which is covered by only a single PI; each PI that contains a * is an EPI. Include all EPI's in the minimal set which covers the function.
  4. Remove from the K-map the EPI's and the 1's they cover, then apply step 3 onto the remaining 1's and PI's, to find the secondary EPI's.
  5. Repeat step 4 until no higher order EPI's are found. Then find a minimum set from the rest of the non-EPI prime implicants that cover the remaining 1's on the map, then add them to the minimal set which covers the function.

Simplifying logic functions using Karnaugh maps … looping

- The logic function can be simplified by replacing canonical terms with a minimum cover of prime implicants, obtained by properly combining squares (**looping**) of the Karnaugh maps which contain **1**s.
- Looping a pair of adjacent **1**s eliminates the variable that appears in both direct and complemented form.

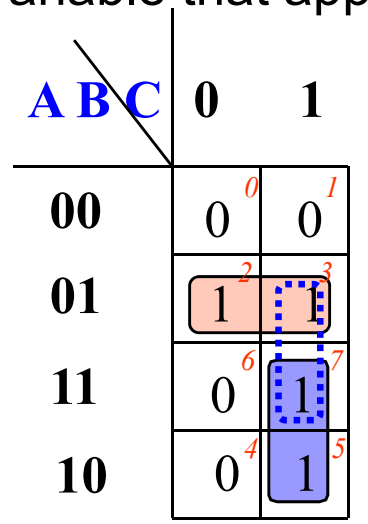
PI: AC, BC, A'B

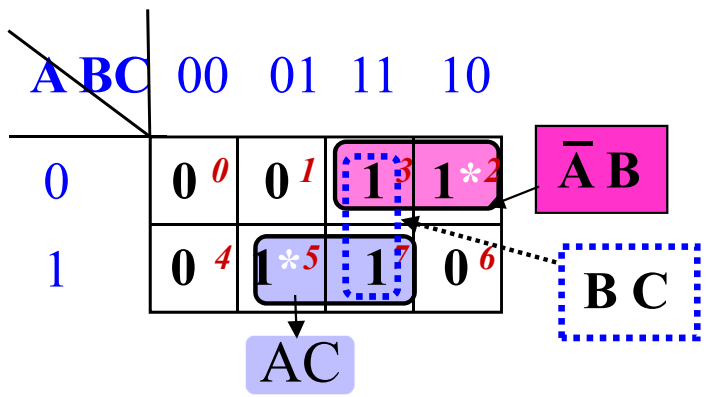
EPI: AC, A'B

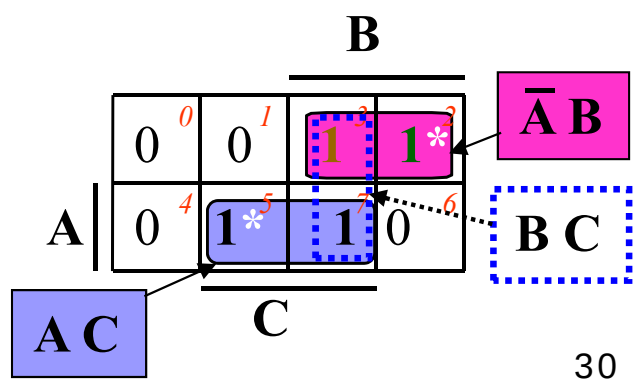
Redundant Implicant: **BC**

	A	B	C	F
(0)	0	0	0	0
(1)	0	0	1	0
(2)	0	1	0	1
(3)	0	1	1	1
(4)	1	0	0	0
(5)	1	0	1	1
(6)	1	1	0	0
(7)	1	1	1	1

$$F = \sum(2,3,5,7) = AC + \bar{A}B$$







Simplifying logic functions using Karnaugh maps

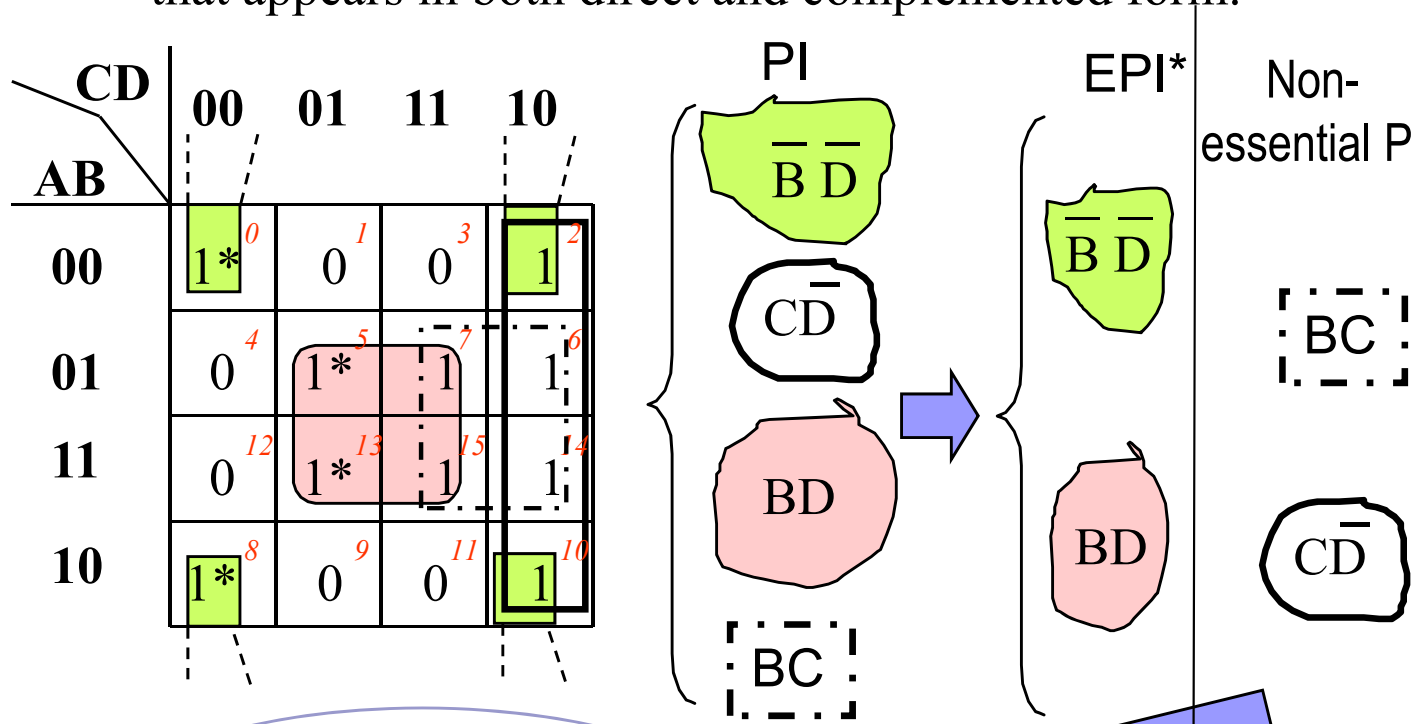
… more looping

$$F = \sum(0,2,5,6,7,8,10,13,14,15) =$$

$$= \bar{A}\bar{B}\bar{C}\bar{D} + \bar{A}\bar{B}C\bar{D} + \bar{A}B\bar{C}\bar{D} + \bar{A}BC\bar{D} + \bar{A}BCD + A\bar{B}\bar{C}\bar{D} + A\bar{B}C\bar{D} + AB\bar{C}\bar{D} + ABC\bar{D} + ABCD$$

	A	B	C	D	F
(0)	0	0	0	0	1
(1)	0	0	0	1	0
(2)	0	0	1	0	1
(3)	0	0	1	1	0
(4)	0	1	0	0	0
(5)	0	1	0	1	1
(6)	0	1	1	0	1
(7)	0	1	1	1	1
(8)	1	0	0	0	1
(9)	1	0	0	1	0
(10)	1	0	1	0	1
(11)	1	0	1	1	0
(12)	1	1	0	0	0
(13)	1	1	0	1	1
(14)	1	1	1	0	1
(15)	1	1	1	1	1

Looping a *quad* of adjacent **1**s eliminates the two variables that appears in both direct and complemented form.



$$F = \bar{B}\bar{D} + BD + \begin{cases} C\bar{D} \\ \text{or} \\ BC \end{cases}$$

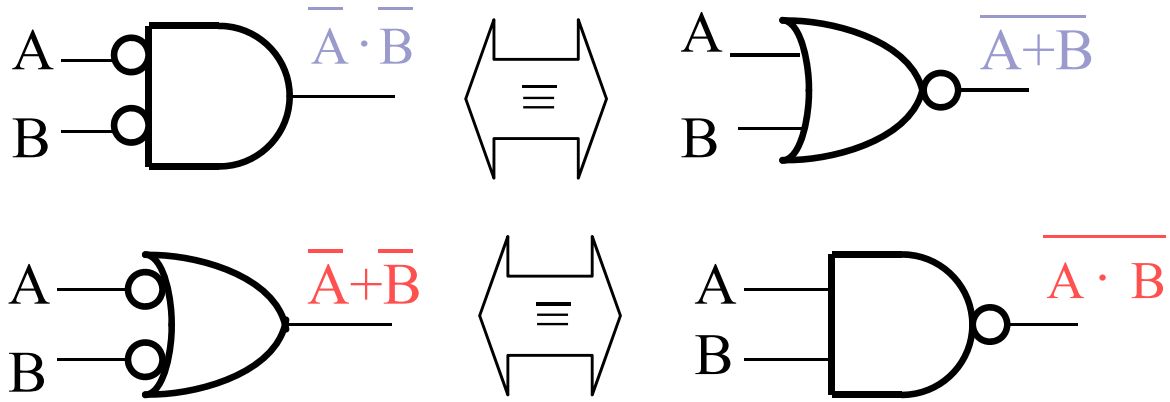
SoP / NAND & PoS / NOR implementation

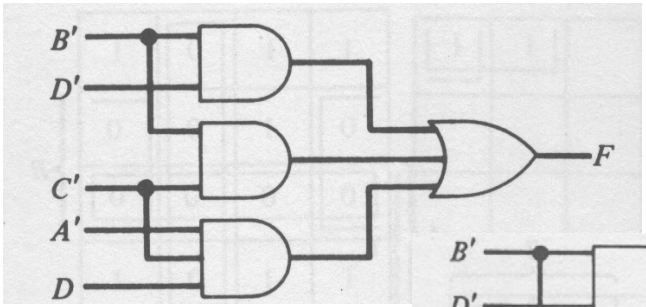
Remember
DeMorgan's Theorem

$$\overline{A} \cdot \overline{B} = \overline{A + B}$$

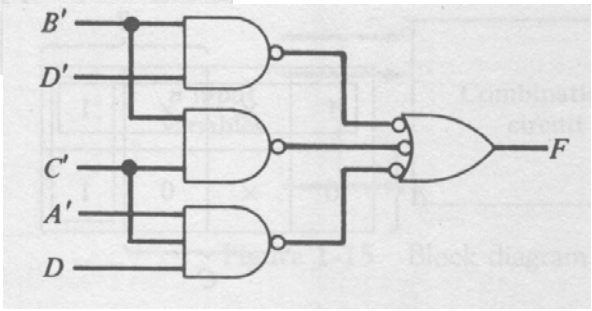
$$\overline{A} + \overline{B} = \overline{A \cdot B}$$

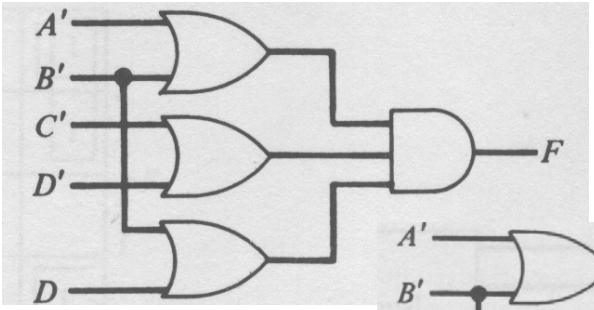
Equivalent Gate Symbols



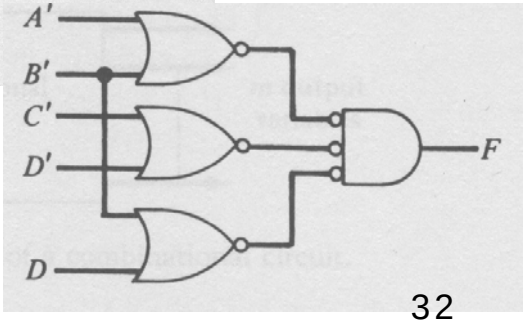


SoP/
NAND





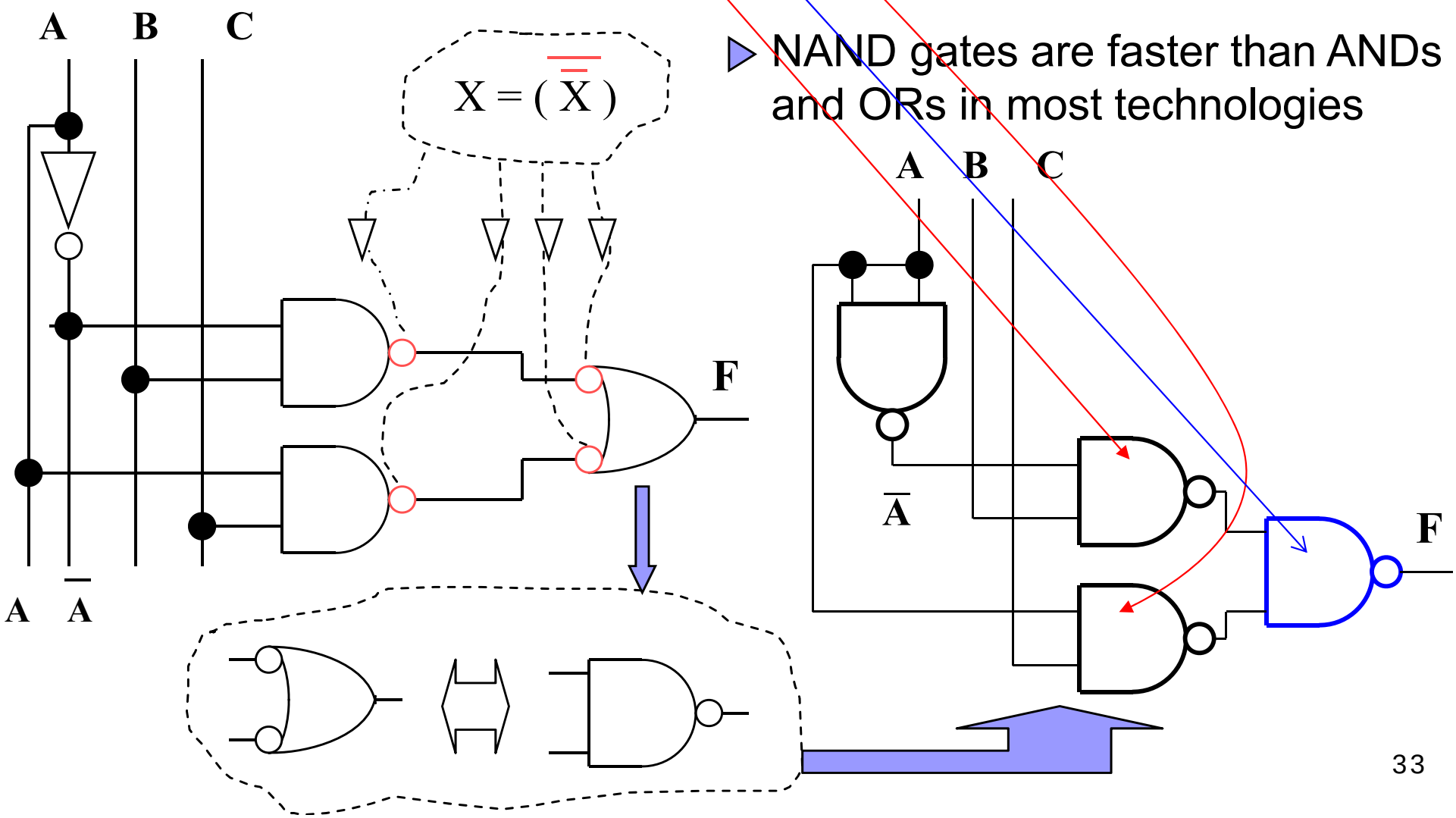
PoS/
NOR



TWO-LEVEL NAND gate implementation of the "Sum-of-Product" logic functions

$$F = \bar{A}\cdot B + A\cdot C = \overline{\overline{\bar{A}\cdot B + A\cdot C}} = \overline{(\overline{\bar{A}\cdot B})\cdot(\overline{A\cdot C})}$$

- NAND gates are faster than ANDs and ORs in most technologies



Incompletely Specified Logic Functions

- **Undefined situations**
- **Can't happen conditions**
  - Example: binary coded decimal increment by 1
    - BCD digits encode decimal digits 0 – 9 in bit patterns 0000 – 1001

	A	B	C	D	W	X	Y	Z
(0)	0	0	0	0	**0**	**0**	**0**	**1**
(1)	0	0	0	1	**0**	**0**	**1**	**0**
(2)	0	0	1	0	**0**	**0**	**1**	**1**
(3)	0	0	1	1	**0**	**1**	**0**	**0**
(4)	0	1	0	0	**0**	**1**	**0**	**1**
(5)	0	1	0	1	**0**	**1**	**1**	**0**
(6)	0	1	1	0	**0**	**1**	**1**	**1**
(7)	0	1	1	1	**1**	**0**	**0**	**0**
(8)	1	0	0	0	**1**	**0**	**0**	**1**
(9)	1	0	0	1	**0**	0	0	0
(10)	1	0	1	0	**x**	**x**	**x**	**x**
(11)	1	0	1	1	**x**	**x**	**x**	**x**
(12)	1	1	0	0	**x**	**x**	**x**	**x**
(13)	1	1	0	1	**x**	**x**	**x**	**x**
(14)	1	1	1	0	**x**	**x**	**x**	**x**
(15)	1	1	1	1	**x**	**x**	**x**	**x**

off-set of W

on-set of W

don't care (DC) set of W

these inputs patterns should never be encountered in practice
– "don't care" about associated output values, hence can be exploited in minimization

Incompletely Specified Logic Functions

$F = \sum(0,1,3,7,8,12) + dc(5,10,13,14)$ ✦ *dc* -> x = *don't care* terms

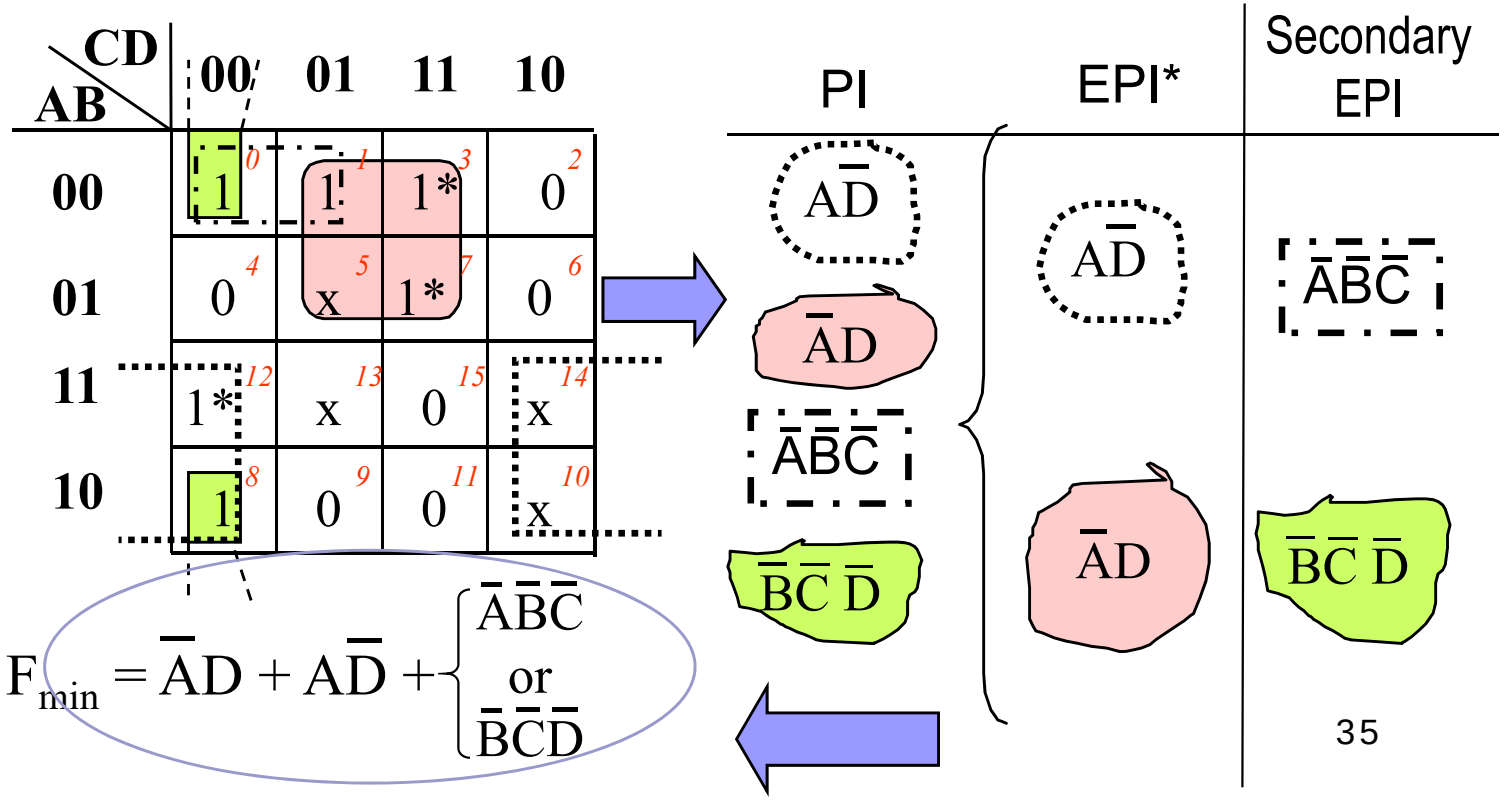
	A	B	C	D	F
(0)	0	0	0	0	1
(1)	0	0	0	1	1
(2)	0	0	1	0	0
(3)	0	0	1	1	1
(4)	0	1	0	0	0
(5)	0	1	0	1	x
(6)	0	1	1	0	0
(7)	0	1	1	1	1
(8)	1	0	0	0	1
(9)	1	0	0	1	0
(10)	1	0	1	0	x
(11)	1	0	1	1	0
(12)	1	1	0	0	1
(13)	1	1	0	1	x
(14)	1	1	1	0	x
(15)	1	1	1	1	0

Don't care terms can be treated as 1's or 0's, depending on which one is more advantageous.

So, if including a *don't care* in a group makes

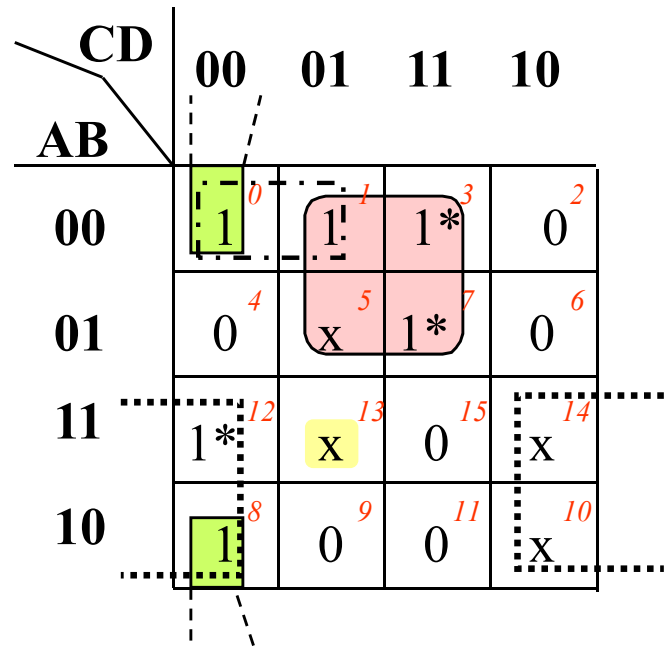
1. the group bigger, or
2. makes it possible to reduce the number of groups,

the *don't care* should be included in the group, but not otherwise!

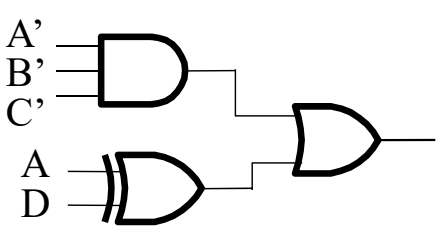


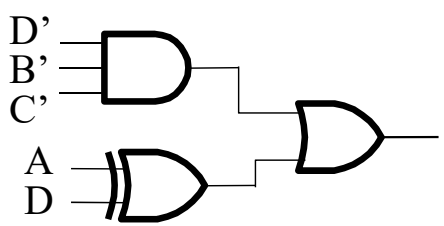
Realizations of Incompletely Specified Logic Functions

	A	B	C	D	F	F_0	F_1	F_2	F_3	F_4	F_5	F_6	F_7	F_8	F_9	F_{10}	F_{11}	F_{12}	F_{13}	F_{14}	F_{15}
(0)	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
(1)	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
(2)	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
(3)	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
(4)	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
(5)	0	1	0	1	x	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
(6)	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
(7)	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
(8)	1	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
(9)	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
(10)	1	0	1	0	x	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
(11)	1	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
(12)	1	1	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
(13)	1	1	0	1	x	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
(14)	1	1	1	0	x	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
(15)	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0



$$F_{13min} = \bar{A}D + A\bar{D} + \begin{cases} \bar{A}\bar{B}\bar{C} \\ \text{or} \\ \bar{B}\bar{C}\bar{D} \end{cases}$$

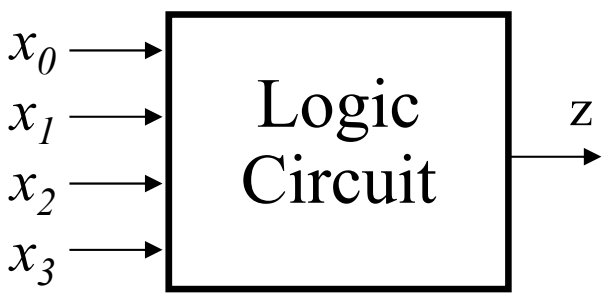




$$F_{13} = A \oplus D + \begin{cases} \bar{A}\bar{B}\bar{C} \\ \text{or} \\ \bar{B}\bar{C}\bar{D} \end{cases}$$

COMBINATIONAL CIRCUITS

- A combinational circuit is a setup of a number of connected logic gates implementing the logic function between n input variables and m output variables.
- In a combinational circuit, the output is time-independent and does only depend on the circuit's input
- In a combinational circuit, the output is “re-computed" as soon as a change in the input occurs, and it is presented to the output with a delay



$$z = F(x_0, x_1, x_2, \ldots) \mid z \in \{0, 1\}, x_k \in \{0, 1\}, k = 0,1,2,\ldots$$

F can be represented by

- logic expression
- truth table
- K-map

Combinational Circuit Design Procedure

1. Define the problem
2. Assign different letter symbols to the input and output variables
3. Derive the truth table defining the relationship between the inputs and the outputs
4. Obtain the simplified boolean expression for each output variable
5. Draw the circuit's logic diagram

Half Adder

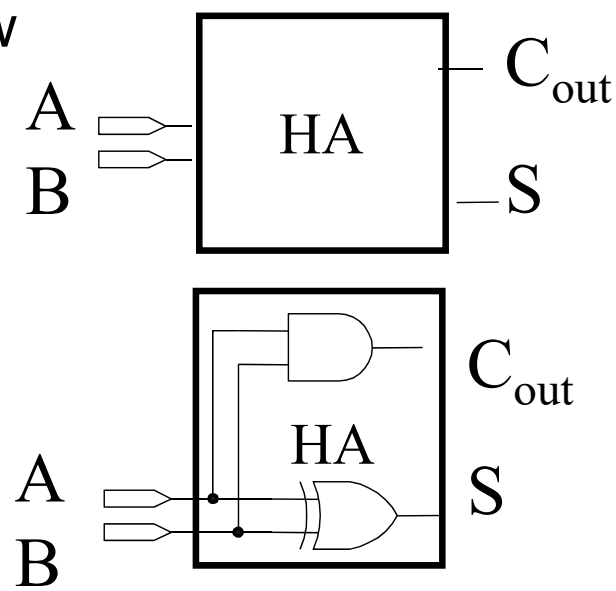
Interpreted as

- Binary number from the specification point of view
- Logic variable from the logic circuit perspective

A	B	C_{out}	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

$$S = A \oplus B$$

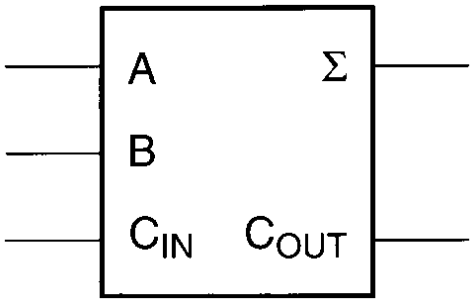
$$C_{out} = (A \cdot B)$$



Shannon identified **<u>the bit</u>** as a fundamental unit of information and, coincidentally, the basic unit of computation.

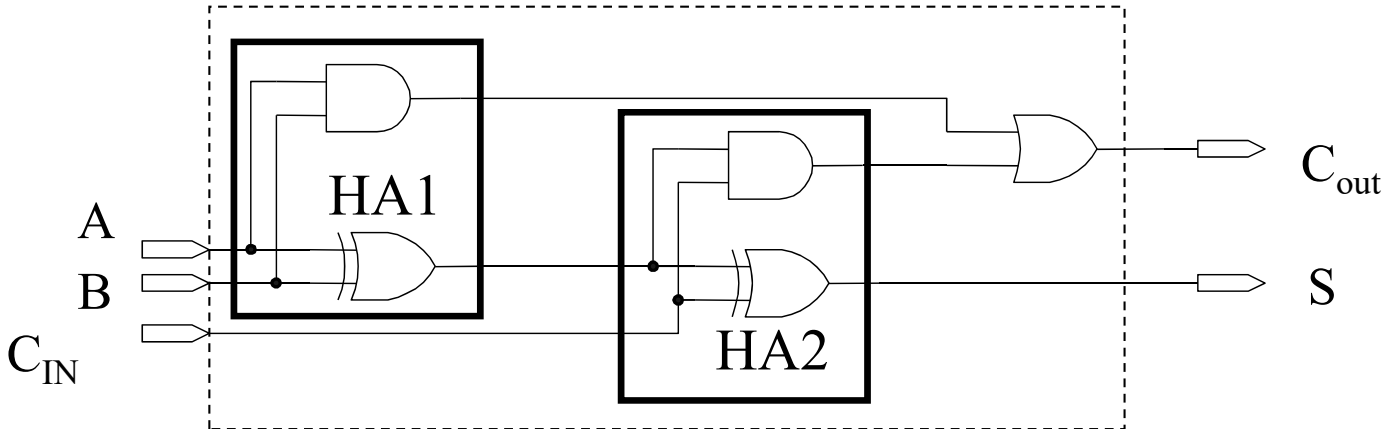
Full Adder

A	B	C_{in}	C_{out}	S
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



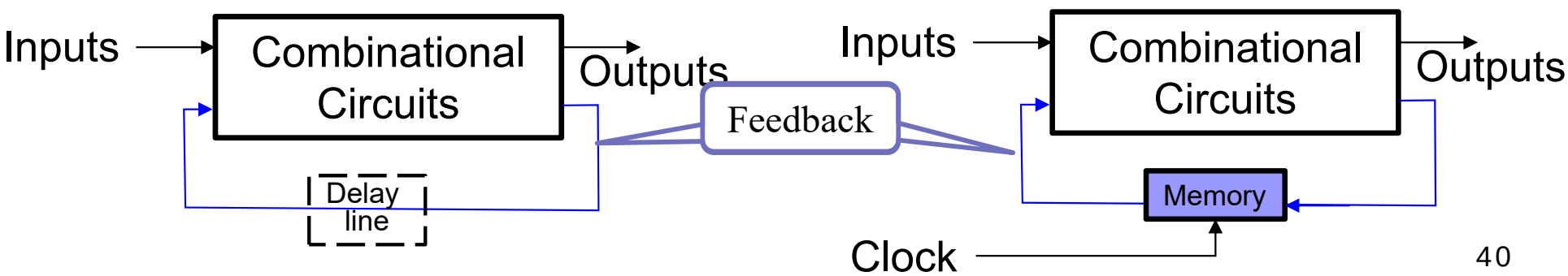
Equations are modified as follows.

- $C_{out} = ((A \oplus B) \cdot C_{IN}) + (A \cdot B)$
- $S = A \oplus B \oplus C_{IN}$



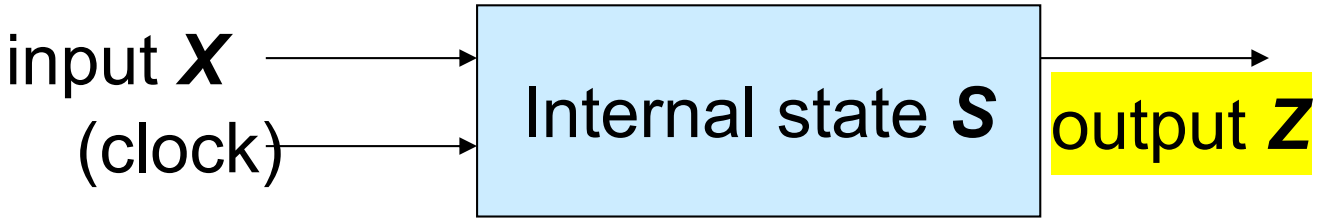
SEQUENTIAL CIRCUITS

- The output of sequential circuits is logic function of the present state of external inputs, but also on the state of these inputs in past. Therefore, they are also referred to as circuits with memory.
- The sequential circuits are formed by additional **feedback** signals, looped backward from the output to the input of the circuit (referred to as signals of internal state of the circuit). By this backward loop, the dependence on previous values of inputs is implemented as dependence on the current internal state of the machine.
- The sequential switching circuits are classified in:
  - **Asynchronous** sequential circuits = any (asynchronously) change of an input may trigger an output change
  - **Synchronous** sequential circuits = outputs may change at time events defined by a periodical control signal (pulse) called **clock**.



Classical automata

Finite State Machines (FSMs)



Next state S^+ computed by function δ

Output Z computed by function λ

- Moore-automata:
  $Z = \lambda(S)$; $S^+ = \delta(X, S)$
- Mealy-automata
  $Z = \lambda(X, S)$; $S^+ = \delta(X, S)$

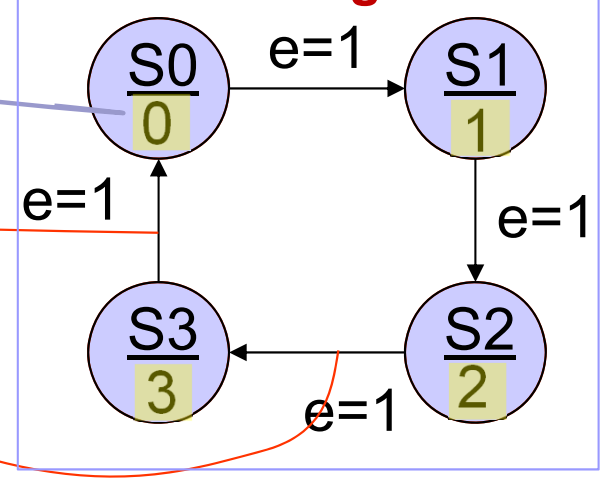
Transition Function

Output Function

A **state diagram** is a directed graph connecting all the possible states of the sequential circuits, where each transition has the following form:

Example
- X = {e}
- Z = {0,1,2,3}
- S = {S0,S1,S2,S3}
- Initial state = S0

State Diagram:



Sequential Circuits as FSM

Input set: $\mathbb{X}=\{X_i \mid i=0,m-1\}$

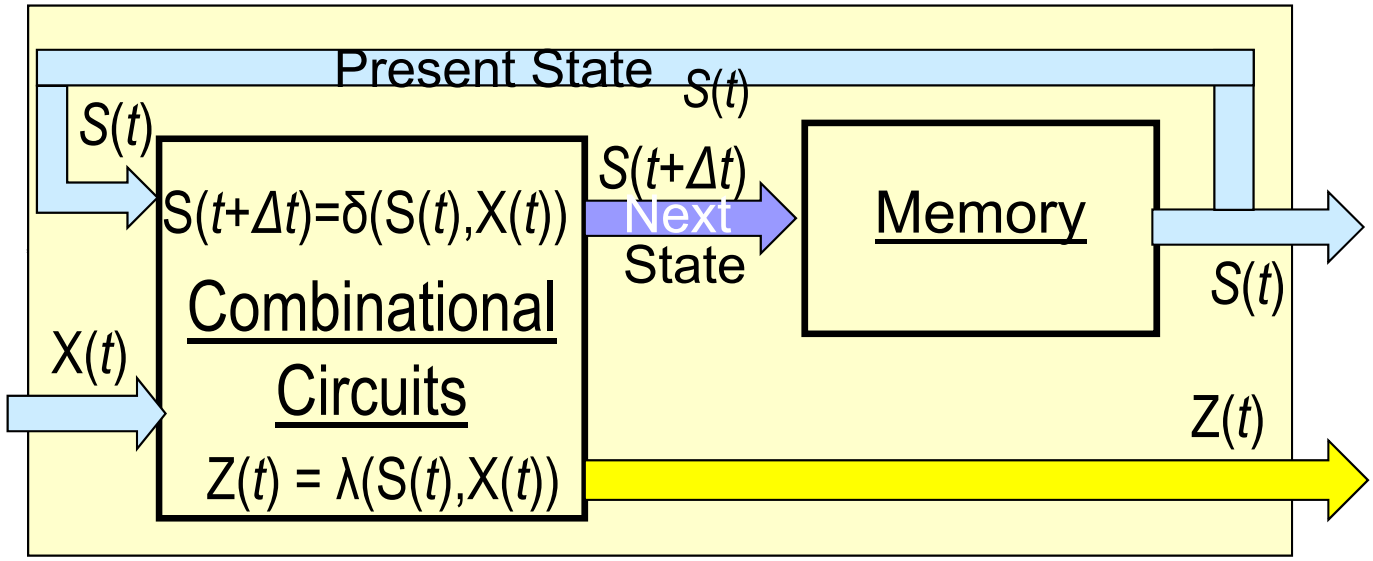
$\mathbb{X}= \{X_0,X_1,X_2,\ldots, X_{m-1}\}$

Output set: $\mathbb{Z}=\{z_j \mid j=0,p-1\}$

$= \{Z_0,Z_1,Z_2,\ldots, Z_{p-1}\}$

States: $\mathbb{S}= \{S_k \mid k=0,q-1\}$

$\mathbb{S}=\{S_0,S_1,S_2,\ldots, S_{q-1}\}$



Transition function δ: $\mathbb{X}\times\mathbb{S}\rightarrow\mathbb{S}$

$$S(t+\Delta t) = \delta(S(t),X(t))$$

Output function λ : $\mathbb{X} \times \mathbb{S} \rightarrow \mathbb{Z}$

$$Z(t) = \lambda\,(S(t),X(t))$$

If $\Delta t=T$ and $t = nT$

(i.e., synchronous)

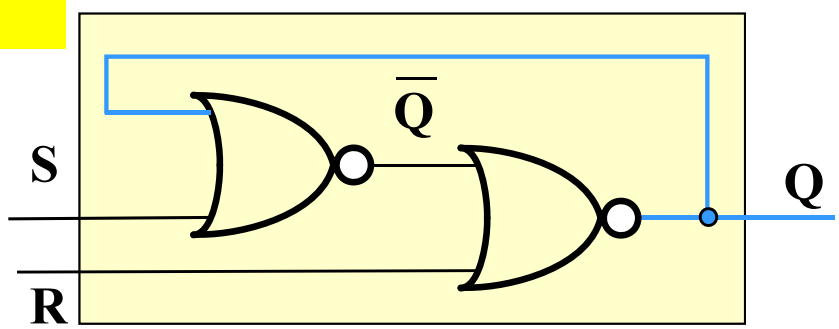
$$S^{n+1} = \delta(S^n,X^n)$$

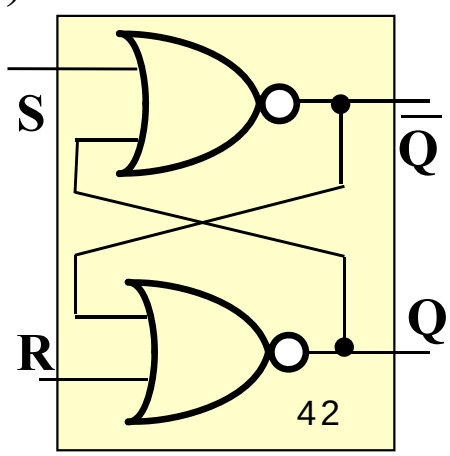
$$Z^n = \lambda(S^n,X^n)$$

Example S-R Latch (asynchronous)

A flip-flop or a latch holds (stores) 1 "bit".

"Bit" ::= "binary digit."



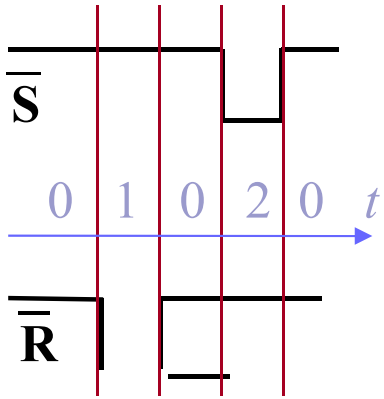


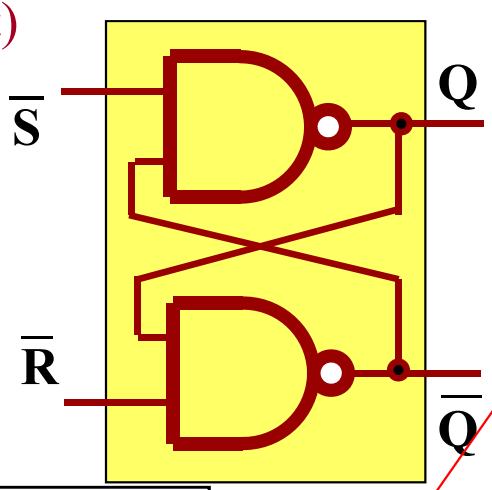
uOttawa

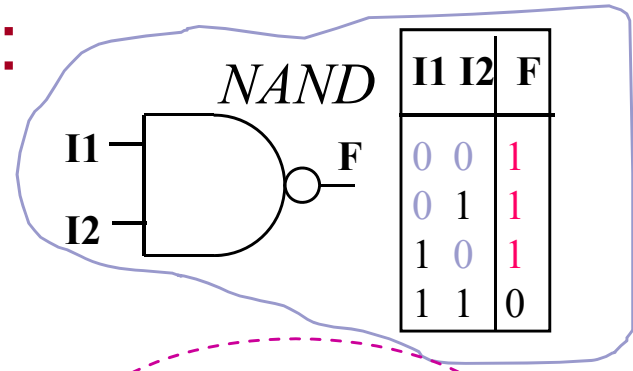
ELEMENTARY **MEMORY** ELEMENTS: LATCHES AND FLIP-FLOPS

✦ *S-R* Latch (Asynchronous Sequential Circuit)

(Set-Reset)

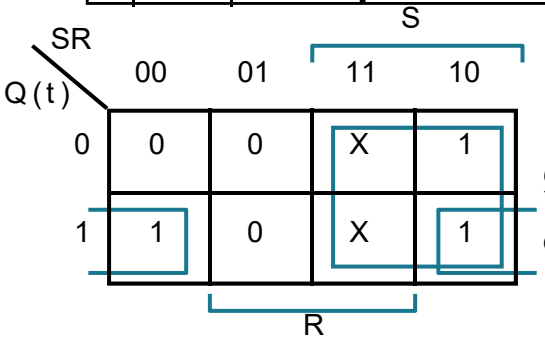






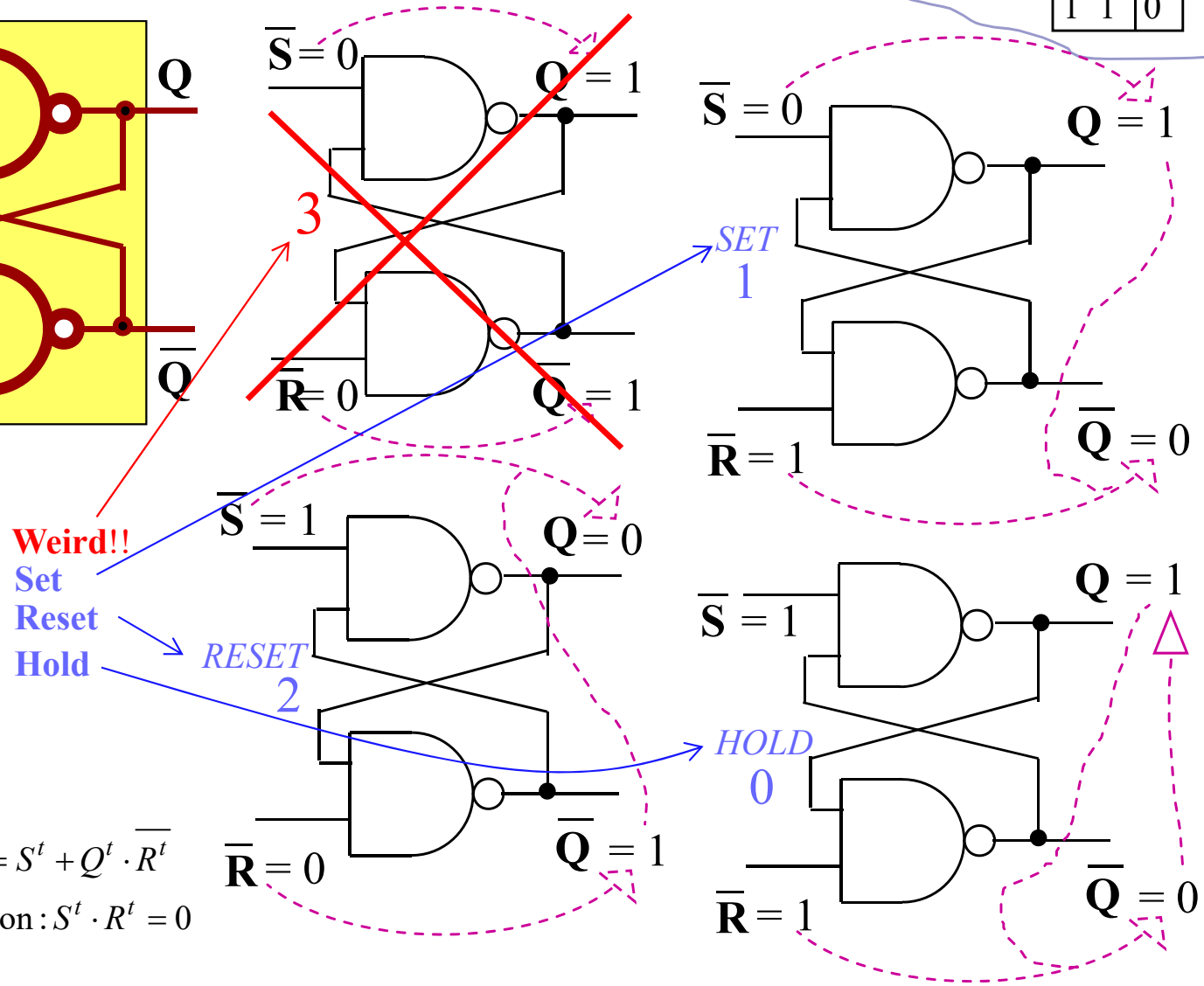
I1	I2	F
0	0	1
0	1	1
1	0	1
1	1	0

t	S	R	$\overline{S}$	$\overline{R}$	$Q^{t+\Delta t}$	$\overline{Q}^{t+\Delta t}$
3	1	1	0	0	1	1
2	1	0	0	1	1	0
1	0	1	1	0	0	1
0	0	0	1	1	Q^t	$\overline{Q}^t$

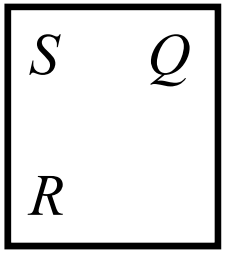


$$Q^{t+\Delta t} = S^t + Q^t \cdot \overline{R^t}$$

$$\text{condition}: S^t \cdot R^t = 0$$



SR Latch



BLOCK DIAGRAM

STATE TABLE

Present State	*Next State* $Q^{t+\Delta t}$			
	SR			
Q^t	00	01	10	11
0	0	0	1	-
1	1	0	1	-

CHARACTERISTIC or FUNCTIONAL TABLE

S^t	R^t	Q^t	$Q^{t+\Delta t}$	$\overline{Q}^{t+\Delta t}$
0	0	0	0	1
0	0	1	1	0
0	1	0	0	1
0	1	1	0	1
1	0	0	1	0
1	0	1	1	0
1	1	0	0?	0?
1	1	1	0?	0?

MEV representation (Map-Entered Variable)

S^t	R^t	$Q^{t+\Delta t}$	*Function*
0	0	Q^t	Hold
0	1	0	Reset
1	0	1	Set
1	1	0?	Indeterminate / Forbidden (non-sense)

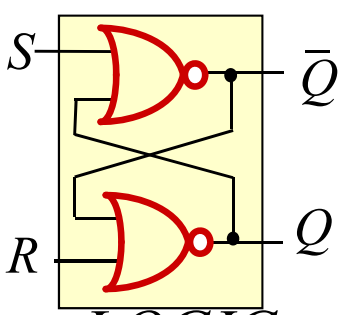
EXCITATION TABLE

Q^t	$Q^{t+\Delta t}$	S^t	R^t
0	0	0	x
0	1	1	0
1	0	0	1
1	1	x	0

Characteristic Equation:

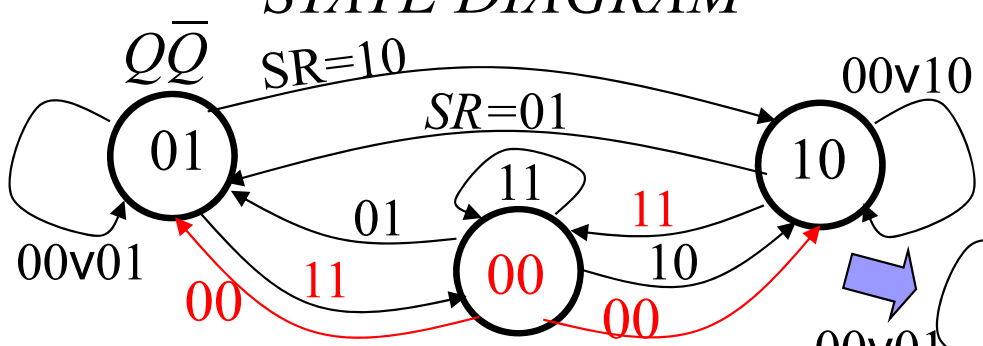
$$Q^{t+\Delta t} = S^t + Q^t \cdot \overline{R^t}$$

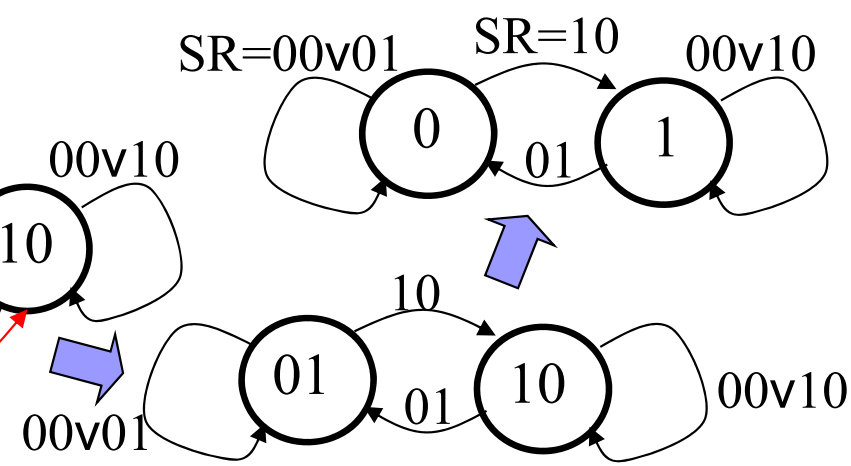
$$\text{condition}: S^t \cdot R^t = 0$$



LOGIC DIAGRAM

STATE DIAGRAM

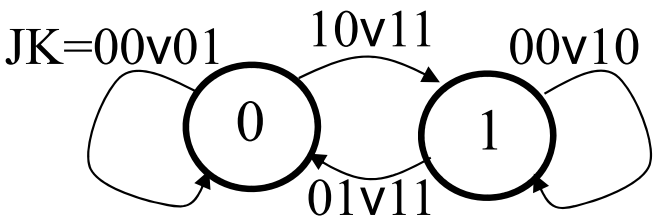




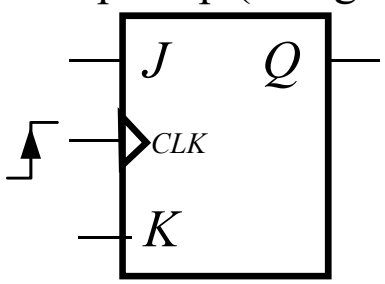
JK Flip Flop

(Synchronous Sequential Circuit)

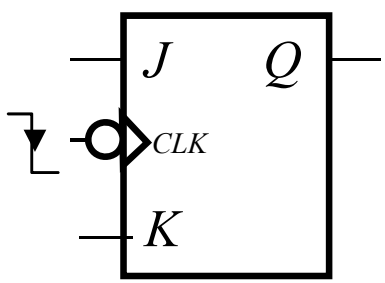
- or how to eliminate the forbidden state?



Positive-Edge -Triggered ***JK*** Flip-Flop (rising edge)



Negative-Edge -Triggered ***JK*** Flip-Flop (falling edge)



CHARACTERISTIC or FUNCTIONAL TABLE

J^n	K^n	Q^{n+1}	*Function*
0	0	Q^n	Hold
0	1	0	Reset
1	0	1	Set
1	1	$\overline{Q^n}$	Toggle

EXCITATION TABLE

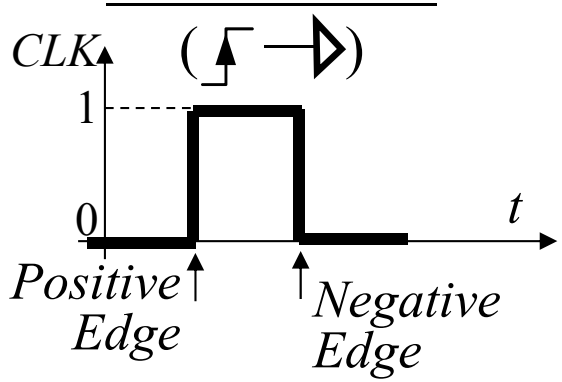
Q^n	Q^{n+1}	J^n	K^n
0	0	0	x
0	1	1	x
1	0	x	1
1	1	x	0

CHARACTERISTIC EQUATION:

$$Q^{n+1} = \overline{Q^n} \cdot J^n + Q^n \cdot \overline{K^n}$$

MEV representation (Map-Entered Variable)

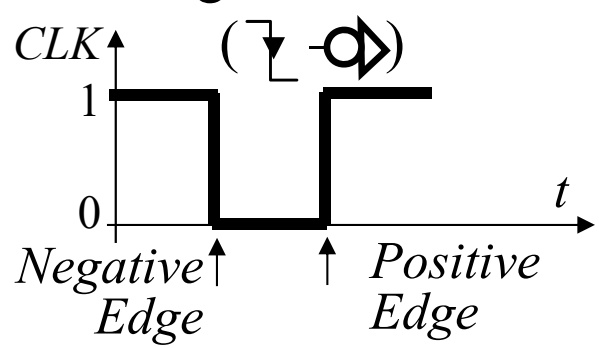
Positive Pulse



Leading edge

Trailing edge

Negative Pulse



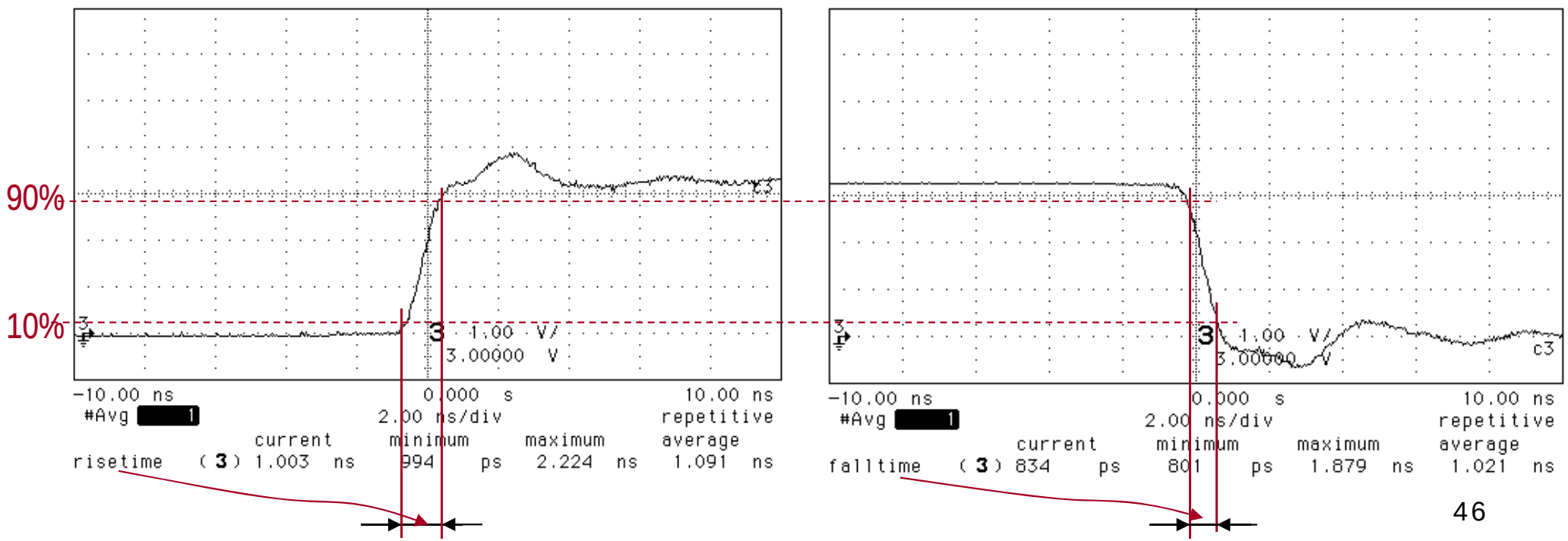
Leading edge

Trailing edge

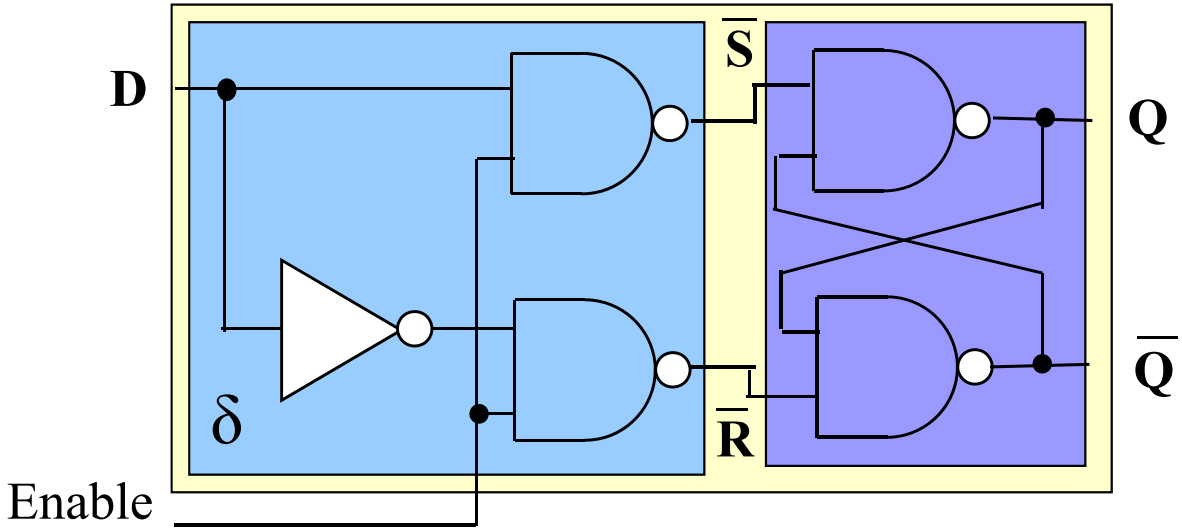
Clock Pulse Definition

IEEE Std 181™-2011 - "IEEE Standard for Transitions, Pulses, and Related Waveforms," IEEE Instrumentation and Measurement Society

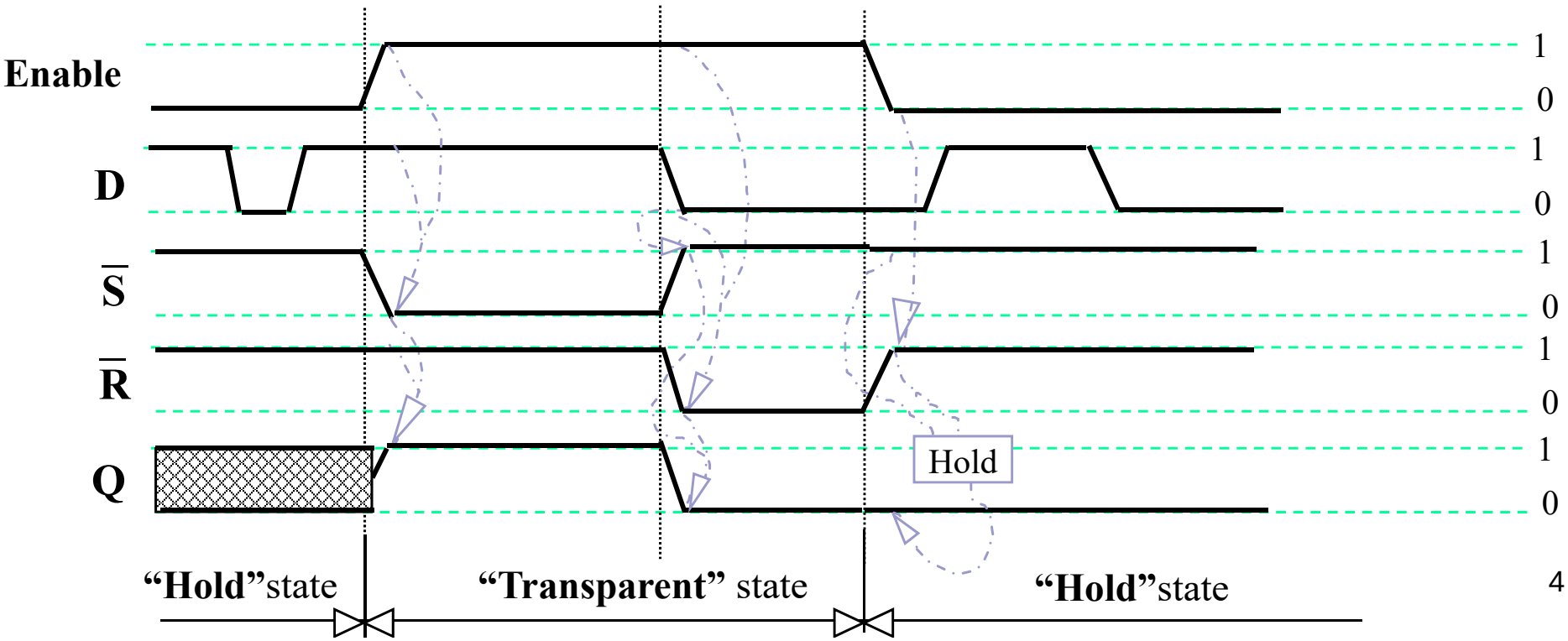
- *negative-going transition*: terminating state is more negative than its originating state.
- *positive-going transition*: terminating state is more positive than its originating state.
- **transition duration**: the difference between the two reference level instants of the same transition. Unless otherwise specified, they are the 10% and 90% of the reference levels.
- NOTE: The following terms are depreciated: risetime (rise time), falltime (fall time), leading edge, rising edge, trailing edge, falling edge, and transition.

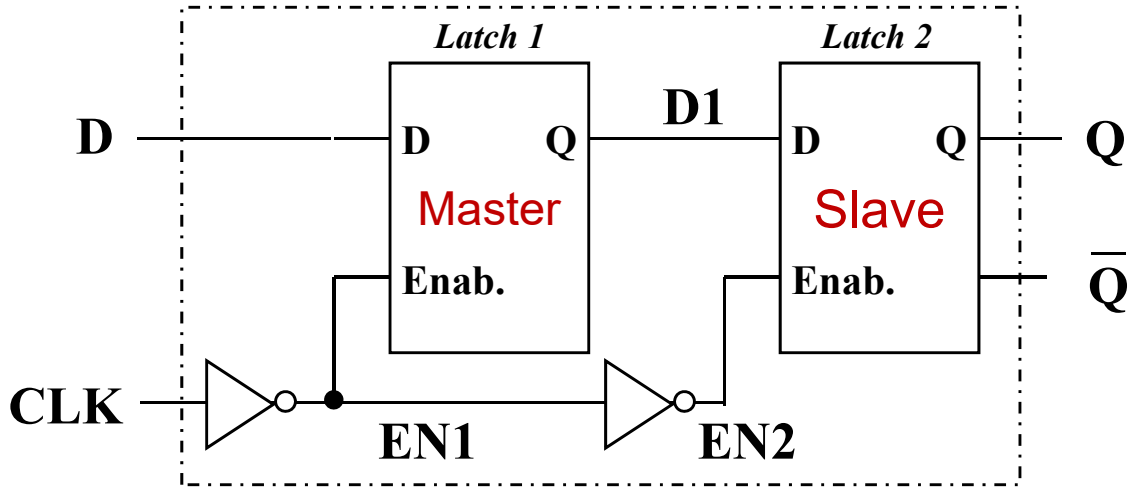


Gated D Latch

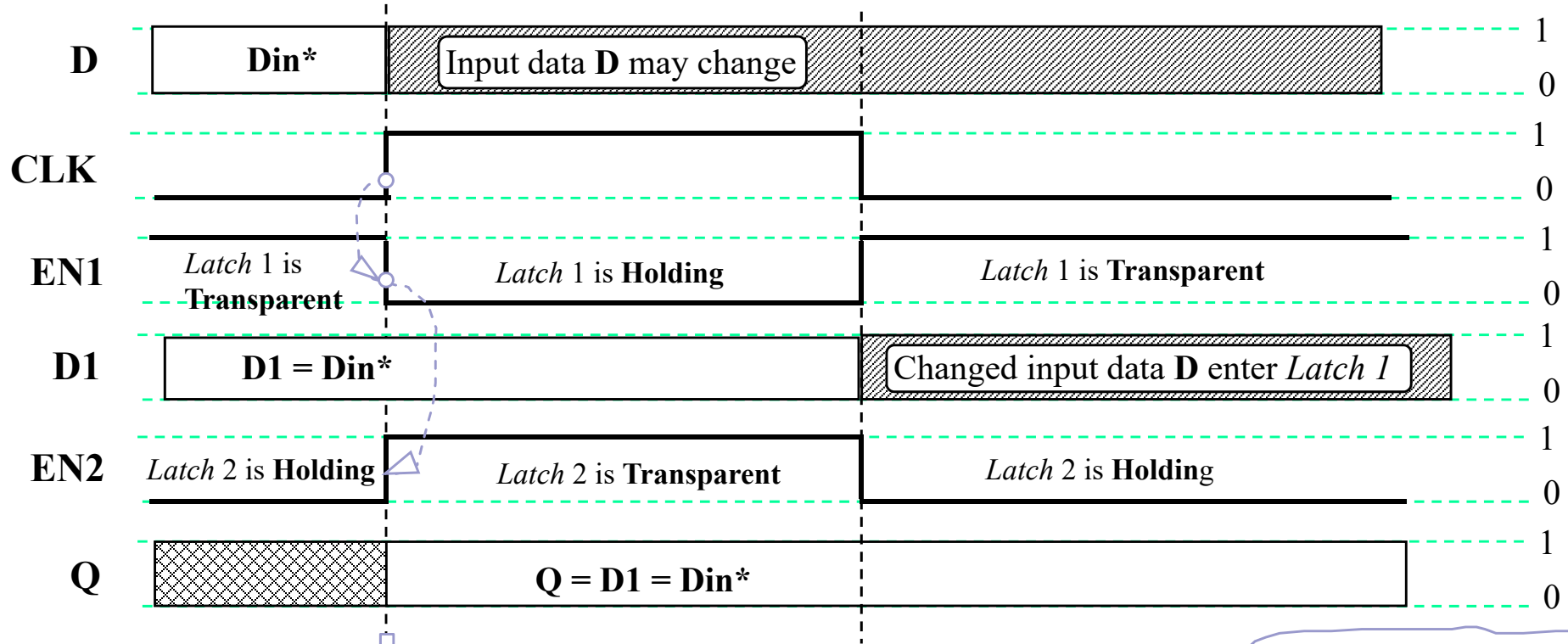


Enable	D	$\overline{S}$	$\overline{R}$	Q	$\overline{Q}$
0	0	1	1	Q	$\overline{Q}$
0	1	1	1	Q	$\overline{Q}$
1	0	1	0	0	1
1	1	0	1	1	0





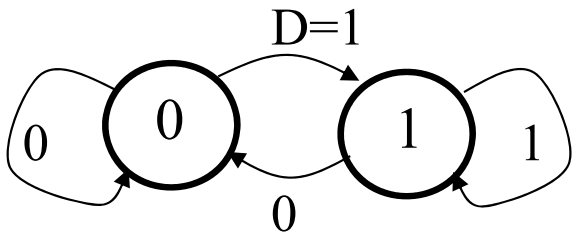
Synchronous *Master-Slave* *D* Flip-Flop



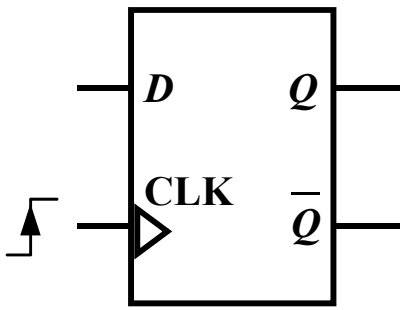
The state of the flip-flop's output **Q** copies input **D** when the positive edge of the clock **CLK** occurs

Positive-Edge-Triggered *D* Flip-Flop

Synchronous *D* Flip-Flop (Edge-triggered)



STATE *DIAGRAM*



BLOCK DIAGRAM

FUNCTIONAL or CHARACTERISTIC TABLE

D^n	Q^n	Q^{n+1}
0	0	0
0	1	0
1	0	1
1	1	1

$Q^{n+1} = D^n$

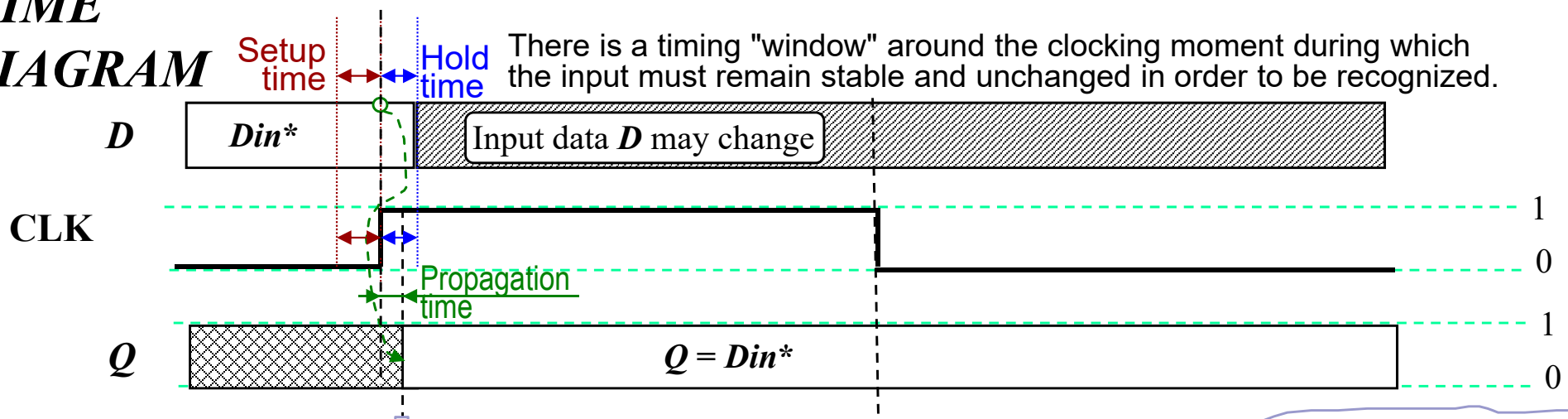
EXCITATION TABLE

Q^n	Q^{n+1}	D^n
0	0	0
0	1	1
1	0	0
1	1	1

$D^n = Q^{n+1}$

TIME DIAGRAM

There is a timing "window" around the clocking moment during which the input must remain stable and unchanged in order to be recognized.

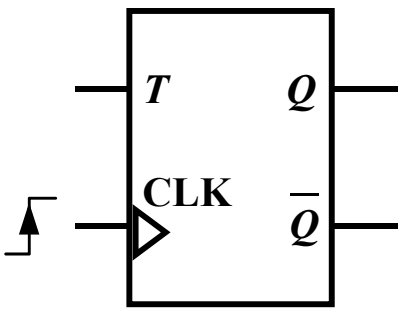


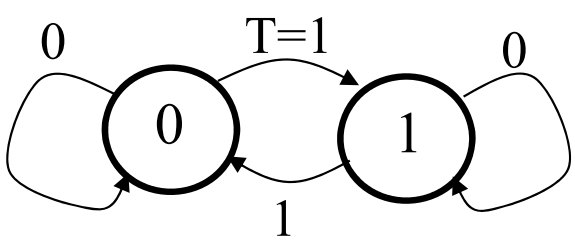
The state of the flip-flop's output **Q** copies input ***D*** when the positive edge of the clock **CLK** occurs

Positive-Edge-Triggered *D* Flip-Flop

T Flip-Flop (Synchronous Sequential Circuit)

Positive-Edge -Triggered
T Flip-Flop (rising edge)





FUNCTIONAL or CHARACTERISTIC TABLE

T^n	Q^n	Q^{n+1}
0	0	0
0	1	1
1	0	1
1	1	0

$$Q^{n+1} = \overline{Q^n} \cdot T + Q^n \cdot \overline{T}$$

EXCITATION TABLE

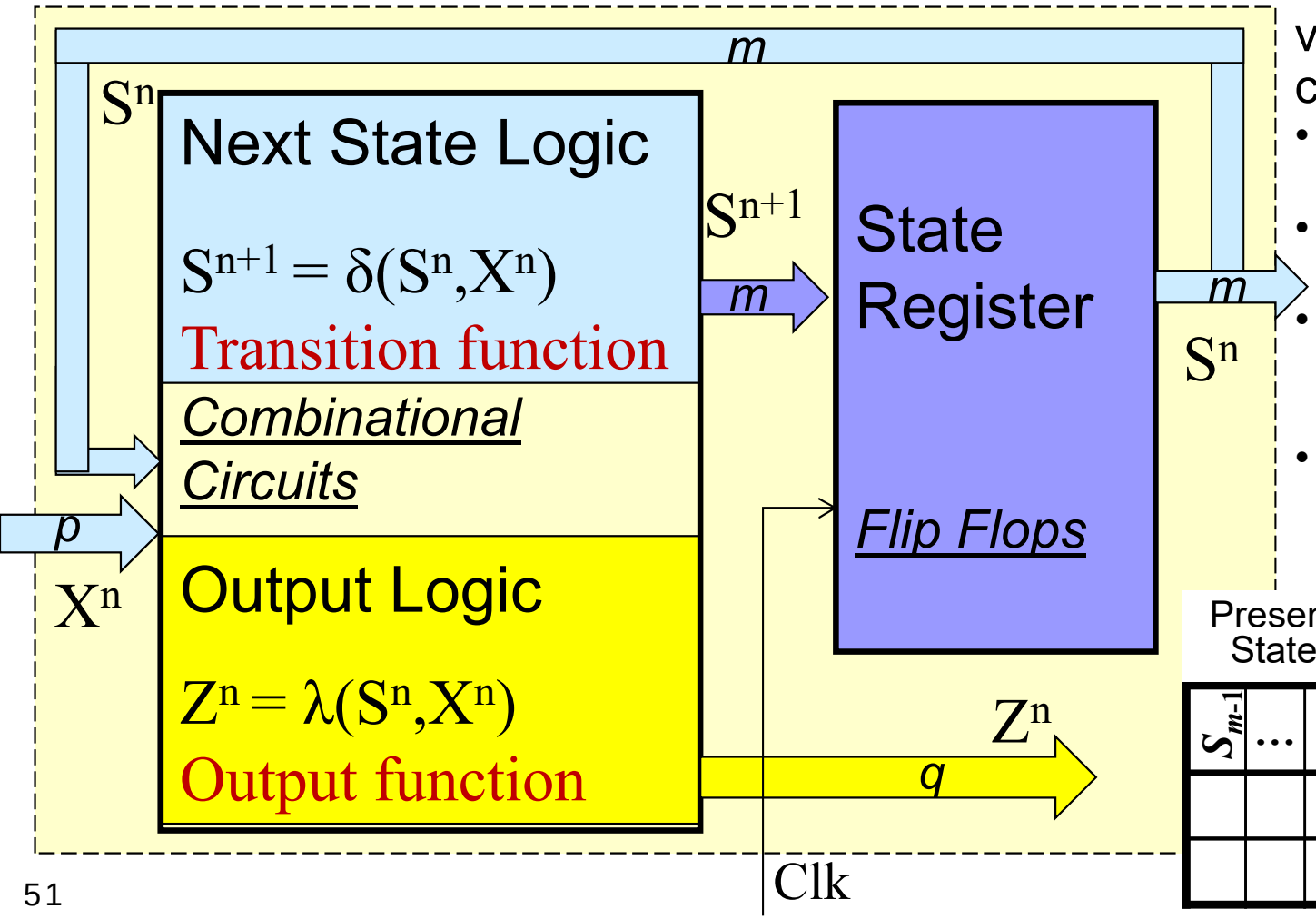
Q^n	Q^{n+1}	T^n
0	0	0
0	1	1
1	0	1
1	1	0

… 2^{m+n} rows of binary combinations ranging from 0 to $2^{m+n}-1$.

SEQUENTIAL CIRCUITS

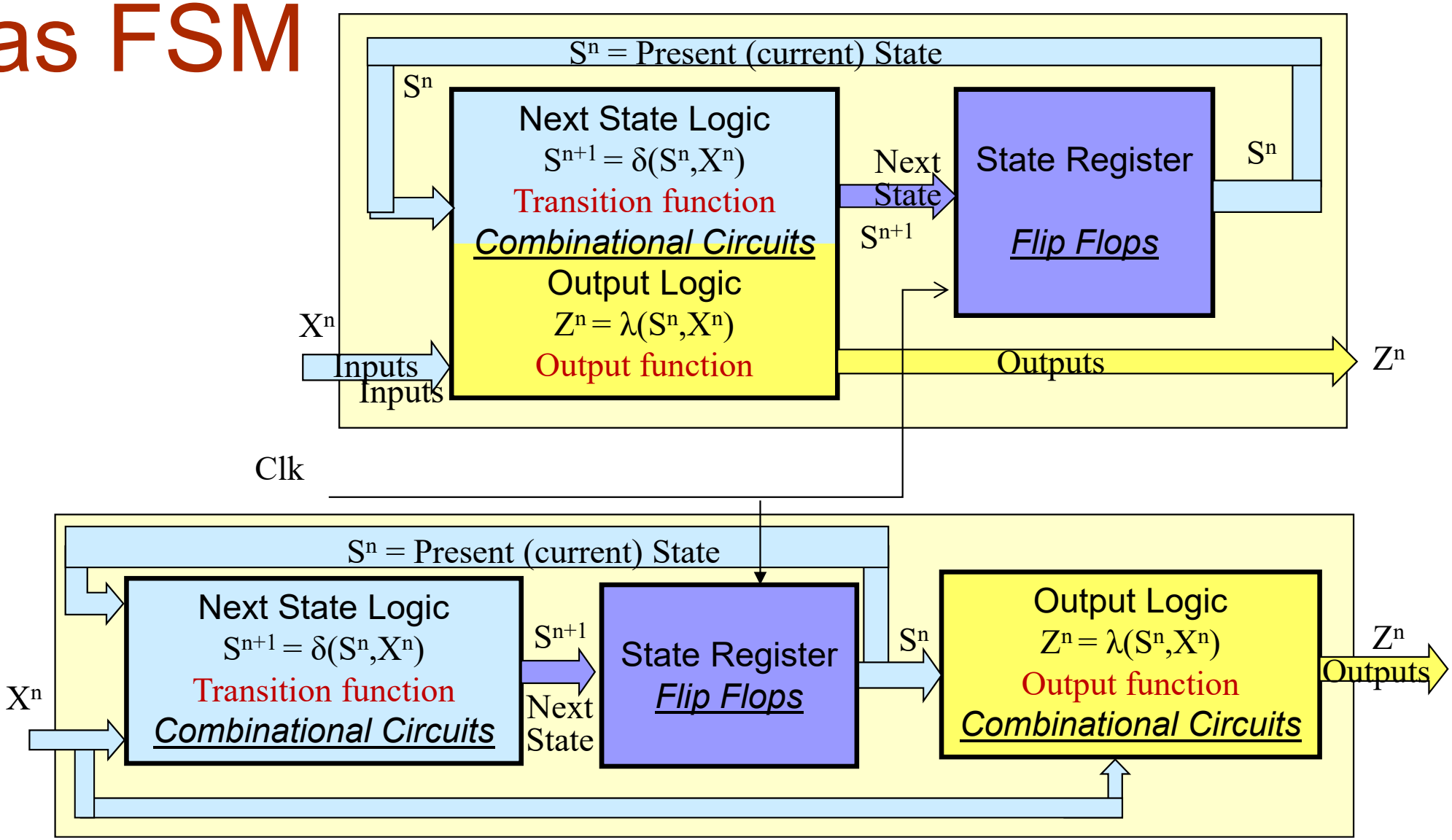
After the *n*-th clock, the outputs (Z^n) and the flip-flops' next-states (S^{n+1}), i.e., the *outputs* of the *Combinational Circuit,* depend on the circuit's *inputs* (X^n) and the flip-flops *current states* (S^n). The **state table** is a truth table showing the circuit's *outputs* $Z^n = \lambda(S^n, X^n)$ and the flip-flops' *next states* $S^{n+1} = \delta(S^n, X^n)$ for **each** circuit's *input* / flip-flops *present states* combination (X^n/S^n). For a sequential circuit with *m* flip-flops, *p input* variables, and *q output* variables, the **state table** consists of:

- *m* columns for the FF *present states* (S^n) $\boldsymbol{S} = \{S_i,\ i = 0, \ldots, m\text{-}1\}$
- *p* columns for the circuit's *inputs* $\boldsymbol{X} = \{X_j,\ j = 0, 1, \ldots, p\text{-}1\}$
- m columns representing the flip-flops' *next states* $S_i^{n+1} = \delta(S^n, X^n), \quad i = 0 \ldots m-1$
- *q* columns representing the circuit's *outputs* (*Z*) $Z_k^n = \delta(S^n, X^n), \quad k = 0 \ldots q-1$



Present State			Inputs			Next State δ			Outputs λ		
S_{m-1}	…	S_0	x_{p-1}	…	x_0	S_{m-1}	…	S_0	z_{q-1}	…	z_0

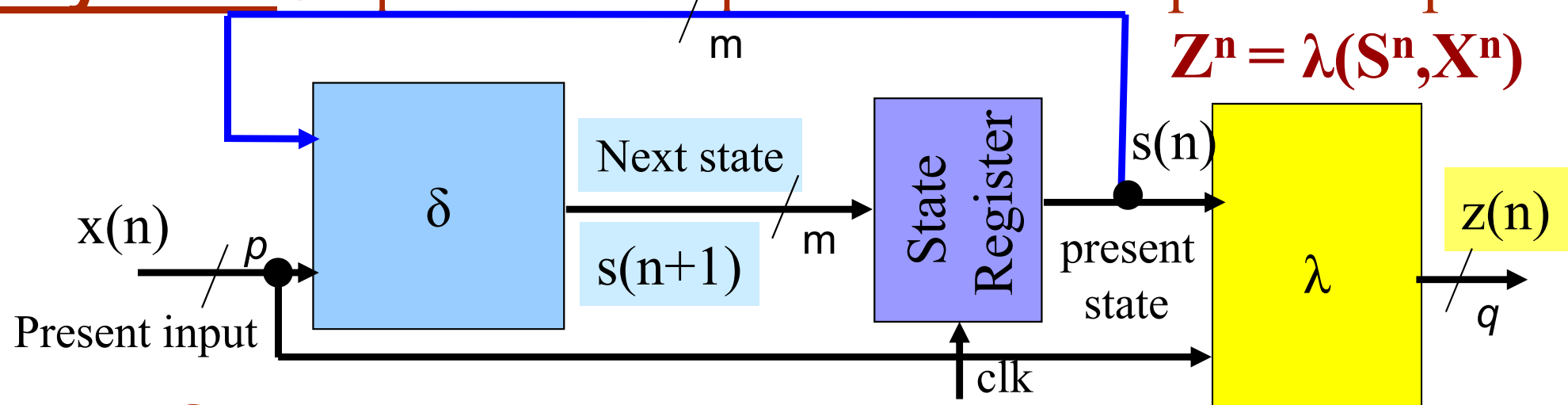
Synchronous Sequential Circuits as FSM



The behavior of sequential circuits can be described using the Finite State Machine (FSM) model

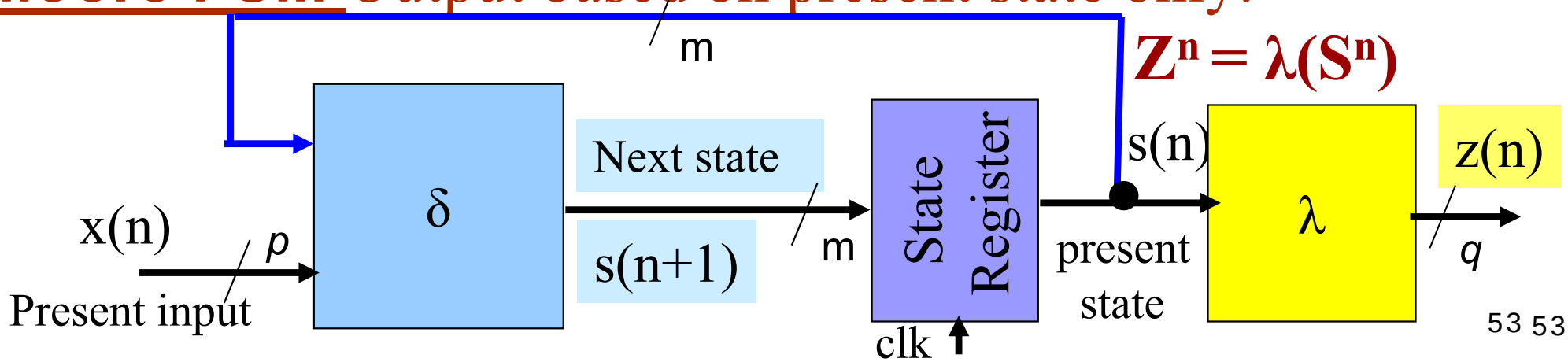
Mealy FSM Output based on present state and present input:

$$Z^n = \lambda(S^n, X^n)$$



Moore FSM Output based on present state only:

$$Z^n = \lambda(S^n)$$

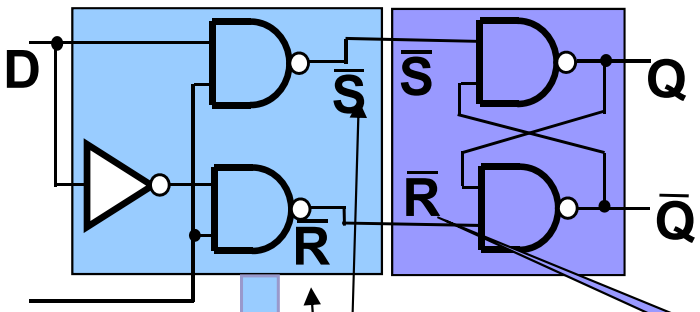


uOttawa

SEQUENTIAL CIRCUITS ANALYSIS

Find the Transition Function of a ... ? ... *D* (*Transparent*) Latch

0. Being given the logic diagram of sequential circuit:



2a. Derive the **State Table** (i.e., the truth table of the transition function) of the analyzed sequential circuit, which is … discovering the D-FF Characteristic (or Functional) Table!

ANALYSIS =
from circuit to transition table
= find $Q(t+\Delta t) = \delta(D(t),Q(t))$

2a

Enable	D	$Q^{t+\Delta t}$	$\bar{Q}^{t+\Delta t}$
0	x	Q^t	$\bar{Q}^t$
1	0	0	1
1	1	1	0

1

CC outputs:
$\bar{S}$=D.Enable
$\bar{R}$=$\bar{D}$.Enable

Enable	D	$\bar{S}$	$\bar{R}$	$Q^{t+\Delta t}$	$\bar{Q}^{t+\Delta t}$
0	0	1	1	Q^t	$\bar{Q}^t$
0	1	1	1	Q^t	$\bar{Q}^t$
1	0	1	0	0	1
1	1	0	1	1	0

Latch's inputs:

$\bar{S}$	$\bar{R}$	$Q^{t+\Delta t}$	$\bar{Q}^{t+\Delta t}$
0	0	1	1
0	1	1	0
1	0	0	1
1	1	Q^t	$\bar{Q}^t$

2. USE SR Characteristic (functional) table

Conclusion: the analyzed circuit is a ***D Latch*** implemented with a ***SR Latch***. When ***Enable*** =1 the information present at the ***D*** input is stored in the latch and will "appear as it is" at the ***Q*** output (=> it is like that there is a "***transparent***" path from the ***D*** input to the ***Q*** output)

uOttawa

SEQUENTIAL CIRCUITS ANALYSIS

from circuit to state diagram = find

$$A(n+1) = \delta_A(x(n),A(n),B(n))$$
$$B(n+1) = \delta_B(x(n),A(n),B(n))$$
$$y(n) = \lambda(x(n),A(n),B(n))$$

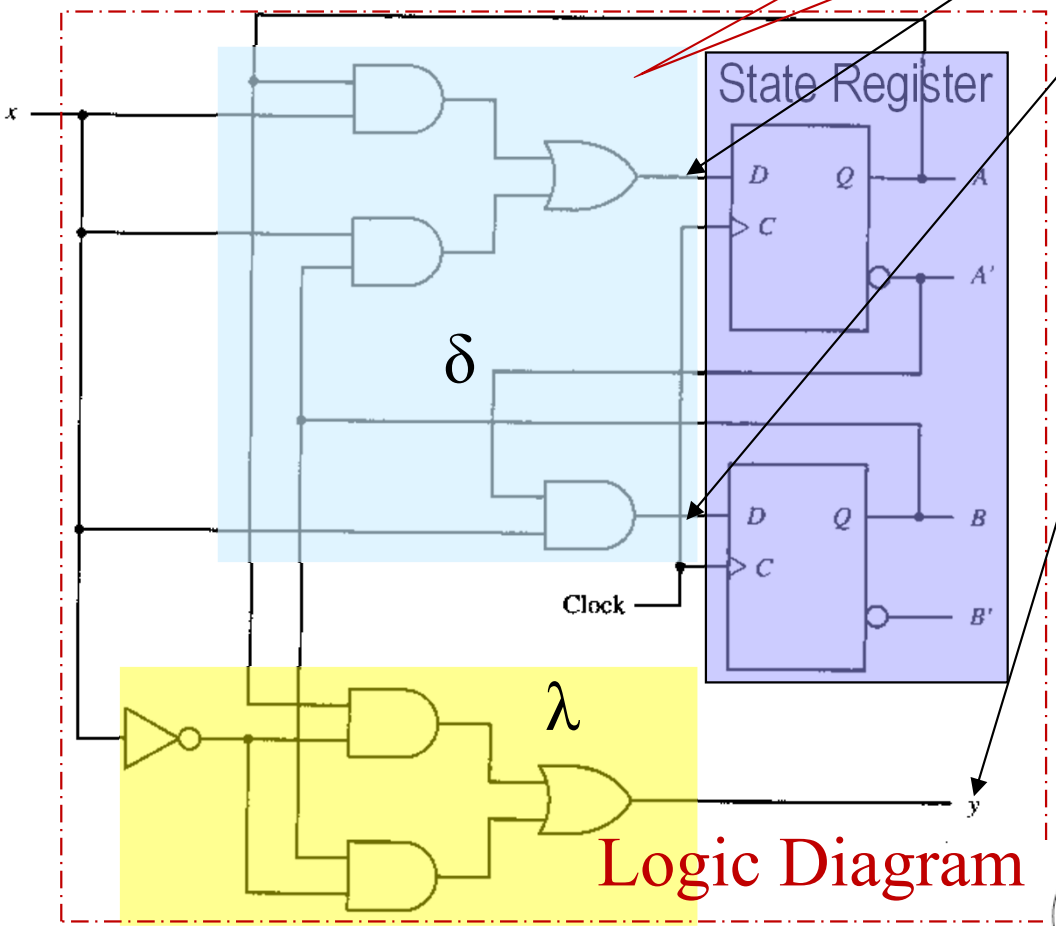
1. Find the outputs of the combinational circuit (δ and λ)

$$D_A = x\,A + x\,B = A^+$$
$$D_B = x\,A' = B^+$$
$$y = x'A + x'B$$

Transition function δ

Output function λ

0. Being given the logic diagram of sequential circuit:

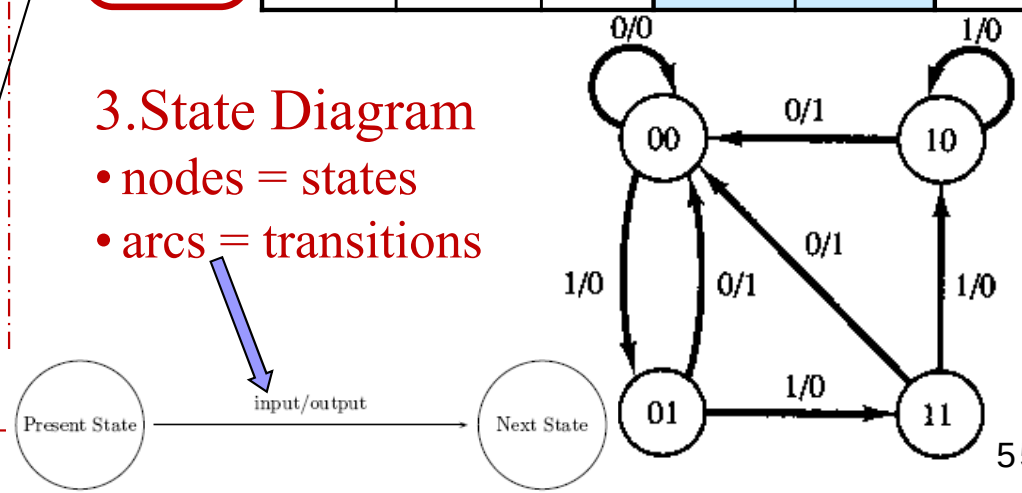


2. Use characteristic equations of D-FF to extract State Table

Present State		In	Next State		Out
A	B	x	A^+	B^+	y
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	0	0	1
0	1	1	1	1	0
1	0	0	0	0	1
1	0	1	1	0	0
1	1	0	0	0	1
1	1	1	1	0	0

3.State Diagram

- nodes = states
- arcs = transitions



Sequential Circuit **Design =** from **Finite State Machine FSM => Logic Circuit**

Sequential Circuit Design Procedure

Step 0: From the **problem specification,** create the **state diagram** which shows all of the states that the **FSM** can be in, and how to switch from one state to another

Step 1: State Table - transition (δ) and output (λ) functions

Make a **state table** based on the problem specification or state diagram. It may show the next states S^{n+1} and outputs Z^n as **functions** of present states S^n and inputs X^n (transition function: $S^{n+1} = \delta(S^n, X^n)$, output function $Z_n = \lambda(S_n, X_n)$.

State minimization = reduce the number of states in the table (not in CEG2136)

Step 2: **State Assignment** (Encoding). Assign binary codes to the states in the state table, if they were not assigned already. If you have N states, your binary codes will have at least $\lceil \log_2 N \rceil$ digits, and your circuit will have at least $\lceil \log_2 N \rceil$ flip-flops ➔ **Transition table** (i.e., binary-coded state table) = a state table with encoded states.

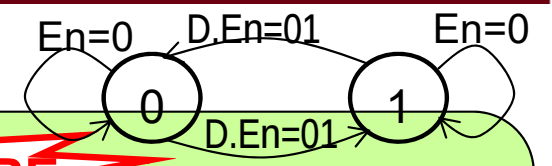
Step 3: Excitation Table and Equations. Choose the type of flip-flops to be used. For each flip-flop and each row of your transition table, find the flip-flop input values that are needed to generate the next state from the present state. You may use flip-flop excitation tables.

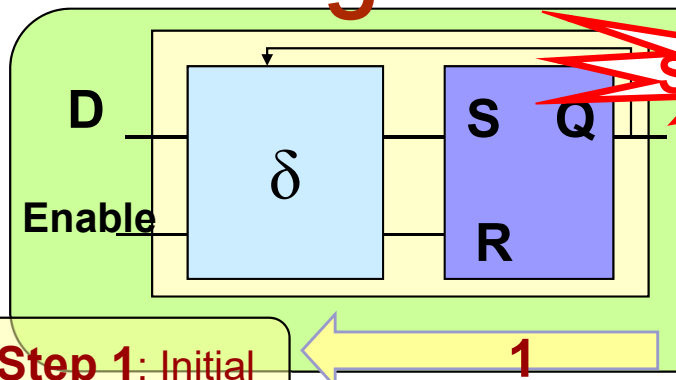
Step 4: Find simplified equations for the flip-flops inputs & the circuit outputs Z.

Step 5: Build the circuit!

uOttawa

Design D-Latch using SR Latch





START HERE

En	D	$Q^{t+\Delta t}$
0	x	Q^t
1	0	0
1	1	1

Given: D-Latch State Diagram or Characteristic Table

1

Step 1: Initial State table

SR Excitation Table respects SR=0

Q^t	$Q^{t+\Delta t}$	S^t	R^t
0	0	0	x
0	1	1	0
1	0	0	1
1	1	x	0

I

II

State table of Sequential Circuit to be Designed

En	D	Q^t	$Q^{t+\Delta t}$	S	R
0	0	0	0	0	x
0	0	1	1	x	0
0	1	0	0	0	x
0	1	1	1	x	0
1	0	0	0	0	x
1	0	1	0	0	1
1	1	0	1	1	0
1	1	1	1	x	0

Excitation table for Sequential Circuit to be Designed

Step 3

4

4

Step 4: SR input equations

S

En DQ_t	00	01	11	10
0	0 [0]	x [1]	x [3]	0 [2]
1	0 [4]	0 [5]	x [7]	1 [6]

$En.D$

R

En DQ_t	00	01	11	10
0	x [0]	0 [1]	0 [3]	x [2]
1	x [4]	1 [5]	0 [7]	0 [6]

$En.\bar{D}$

4

$S = En.D$

$R = En.\bar{D}$

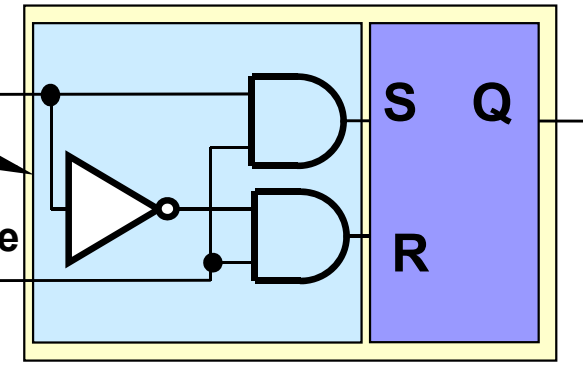
5

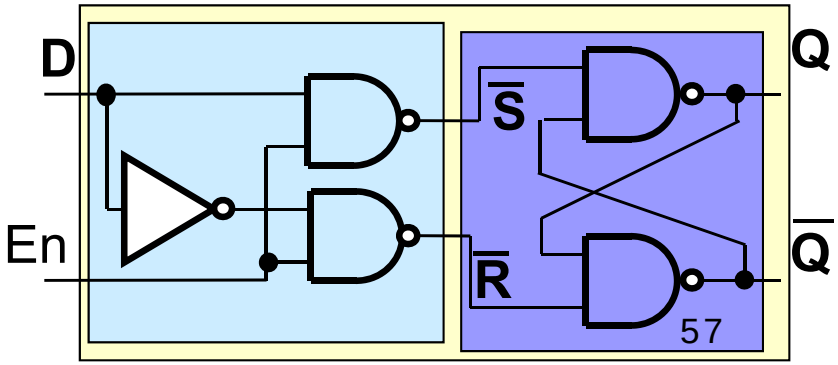
$\bar{S} = \overline{En.D}$

$\bar{R} = \overline{En.\bar{D}}$

5'

Step 5





Design a D-Latch using SR Latch ++

START

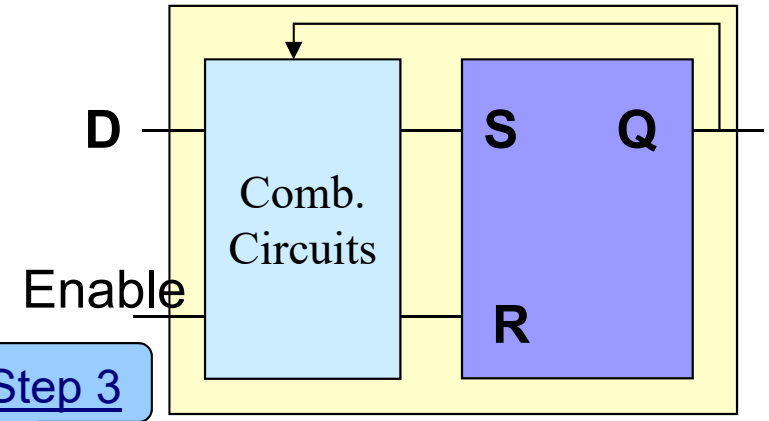
Given: D-Latch Characteristic (functional) table

En	D	$Q^{t+\Delta t}$
0	x	Q^t
1	0	0
1	1	1

Step 1

SR Excitation Table

Q^t	$Q^{t+\Delta t}$	S^t	R^t
0	0	0	x
0	1	1	0
1	0	0	1
1	1	x	0



Step 3

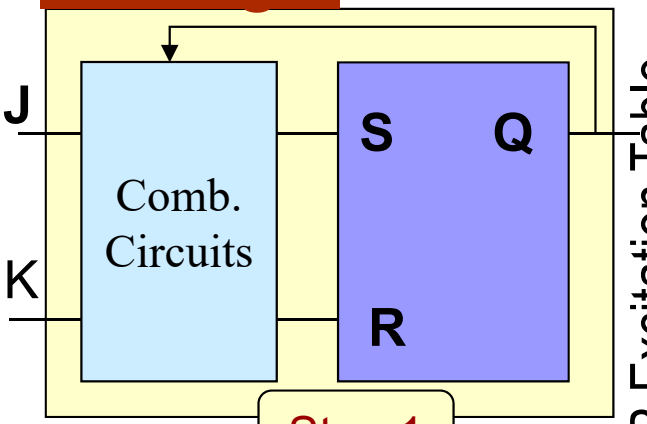
En	D	Q^t	$Q^{t+\Delta t}$
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

State table of the Sequential Circuit to be Designed (D-Latch)

Q^t	$Q^{t+\Delta t}$	S	R
0	0	0	x
1	1	x	0
0	0	0	x
1	1	x	0
0	0	0	x
1	0	0	1
0	1	1	0
1	1	x	0

En	D	Q^t	S	R
0	0	0	0	x
0	0	1	x	0
0	1	0	0	x
0	1	1	x	0
1	0	0	0	x
1	0	1	0	1
1	1	0	1	0
1	1	1	x	0

Design a JK-FF using SR Latch



SR Excitation Table

Q^t	$Q^{t+\Delta t}$	S^t	R^t
0	0	0	x
0	1	1	0
1	0	0	1
1	1	x	0

$S = JQ'$

$R = KQ$

J \ KQ^t	00	01	11	10
0	0 (0)	x (1)	0 (3)	0 (2)
1	1 (4)	x (5)	0 (7)	1 (6)

J \ KQ^t	00	01	11	10
0	x (0)	0 (1)	1 (3)	x (2)
1	0 (4)	0 (5)	1 (7)	0 (6)

Step 1

Step 3

J	K	Q^t	$Q^{t+\Delta t}$
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

State table of the Sequential Circuit to be Designed (JK)

Q^t	$Q^{t+\Delta t}$	S	R
0	0	0	x
1	1	x	0
0	0	0	x
1	0	0	1
0	1	1	0
1	1	x	0
0	1	1	0
1	0	0	1

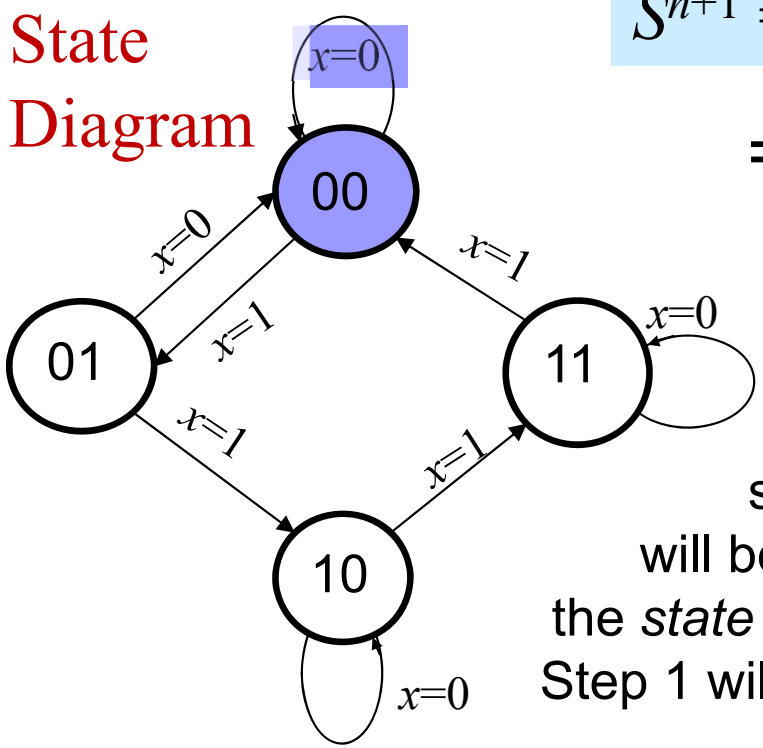
J	K	Q^t	S	R
0	0	0	0	x
0	0	1	x	0
0	1	0	0	x
0	1	1	0	1
1	0	0	1	0
1	0	1	x	0
1	1	0	1	0
1	1	1	0	1

SEQUENTIAL CIRCUITS DESIGN

FSM (Finite State Machine) Representation
Design = from State Diagram => Logic Circuit Diagram

Problem: Apply the Sequential Circuit Design Procedure to derive the logic diagram of a logic circuit which implements the following FSM

State Diagram



$$S^{n+1} = \delta(S^n, X^n)$$

=> => => =>

NOTE: The states are already encoded, so the *Transition Table* will be derived directly from the *state diagram*. Hence, after Step 1 will go directly to Step 3.

Step 1 State Table

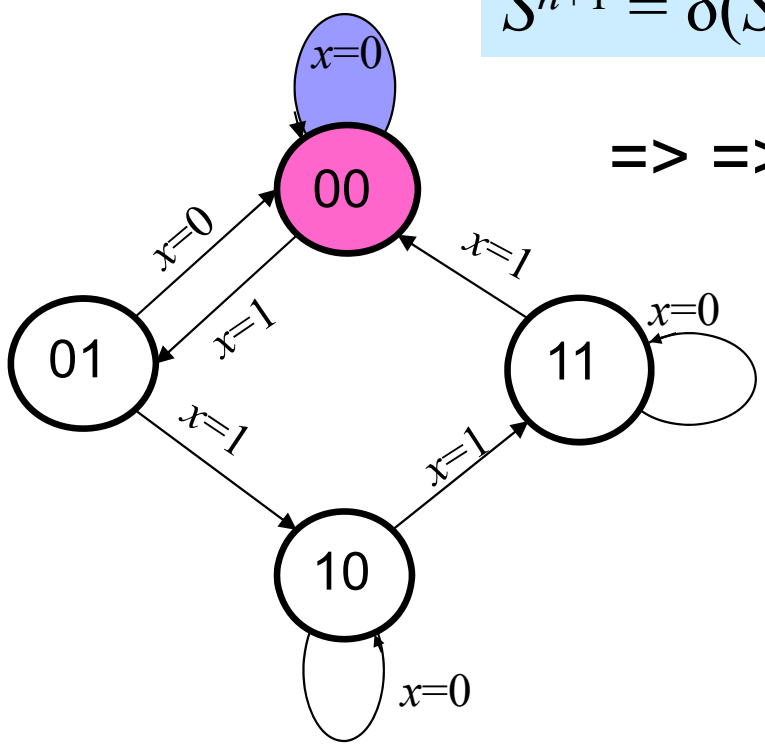
Present State S^n		In	Next State S^{n+1}	
A	B	x	A^+	B^+
0	0	0		
0	0	1		
0	1	0		
0	1	1		
1	0	0		
1	0	1		
1	1	0		
1	1	1		

SEQUENTIAL CIRCUITS DESIGN

FSM => => => Transition Table Step 1

$$S^{n+1} = \delta(S^n, X^n)$$

=> => => =>

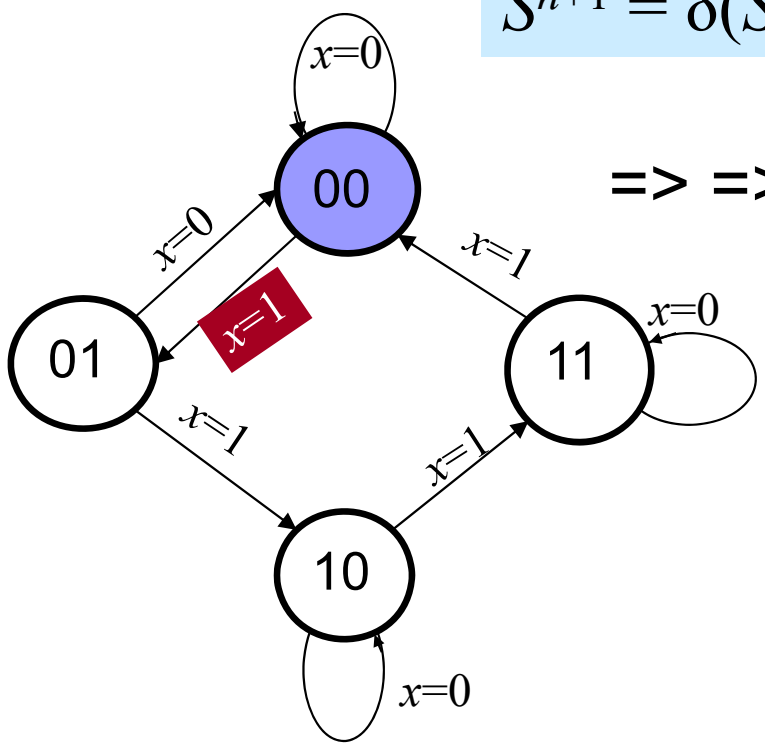


Present State S^n		In	Next State S^{n+1}	
A	B	x	A^+	B^+
0	0	0	0	0
0	0	1		
0	1	0		
0	1	1		
1	0	0		
1	0	1		
1	1	0		
1	1	1		

SEQUENTIAL CIRCUITS DESIGN

FSM => => => Transition Table Step 1

$$S^{n+1} = \delta(S^n, X^n)$$



=> => => =>

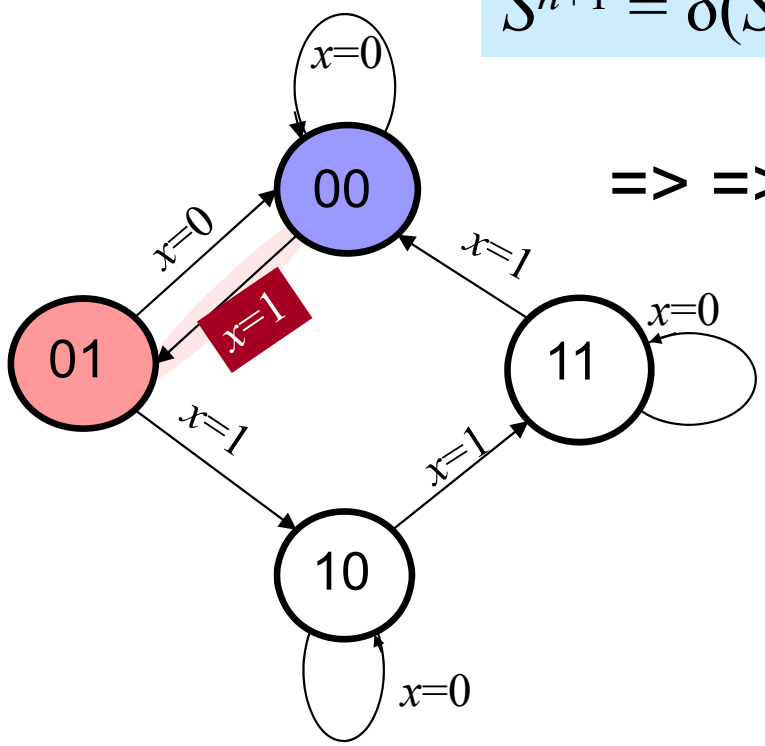
Present State S^n		In	Next State S^{n+1}	
A	B	x	A^+	B^+
0	0	0	0	0
0	0	1		
0	1	0		
0	1	1		
1	0	0		
1	0	1		
1	1	0		
1	1	1		

SEQUENTIAL CIRCUITS DESIGN

FSM => => => Transition Table Step 1

$$S^{n+1} = \delta(S^n, X^n)$$

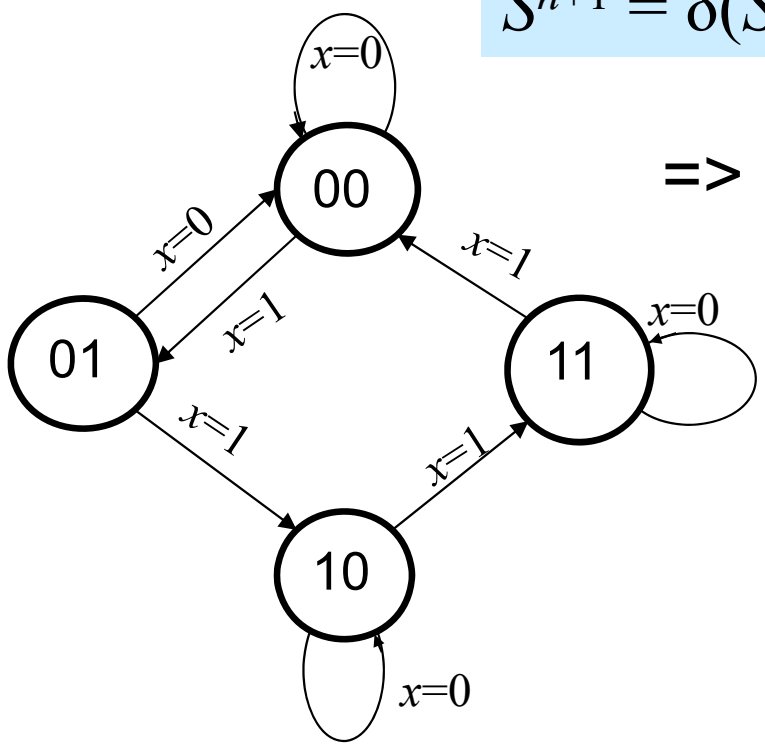
=> => => =>



Present State S^n		In	Next State S^{n+1}	
A	B	x	A^+	B^+
0	0	0	0	0
0	0	1	0	1
0	1	0		
0	1	1		
1	0	0		
1	0	1		
1	1	0		
1	1	1		

SEQUENTIAL CIRCUITS DESIGN

FSM => => => Transition Table Step 1



$$S^{n+1} = \delta(S^n, X^n)$$

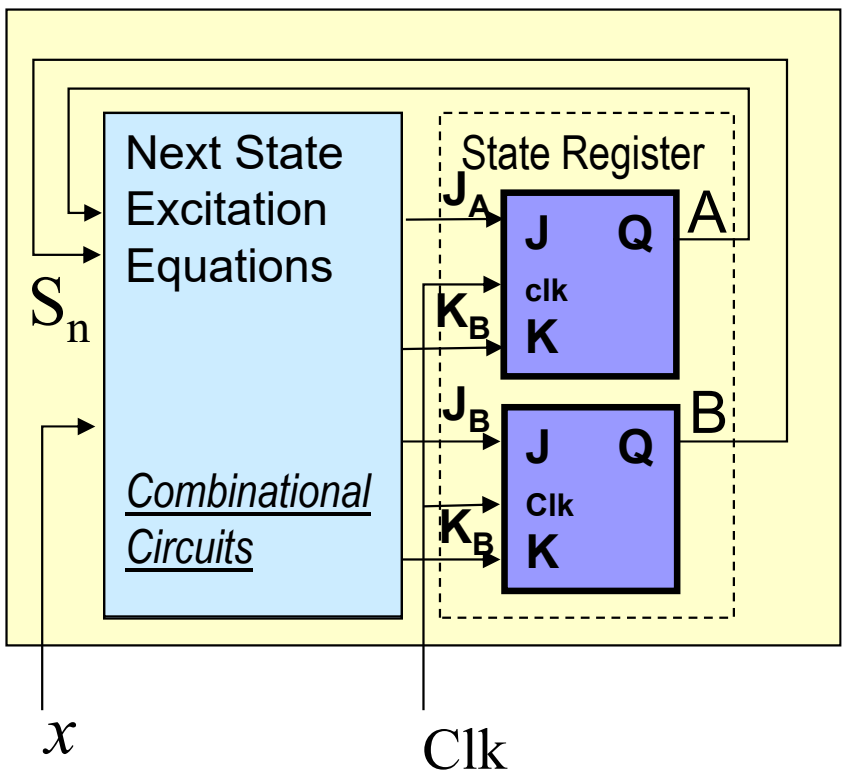
=> => => =>

Present State S^n		In	Next State S^{n+1}	
A	B	x	A^+	B^+
0	0	0	0	0
0	0	1	0	1
0	1	0	0	0
0	1	1	1	0
1	0	0	1	0
1	0	1	1	1
1	1	0	1	1
1	1	1	0	0

SEQUENTIAL CIRCUITS DESIGN

FSM Excitation Table

Step 3



Present State S^n		In	Next State S^{n+1}		Next State control gates' out = Flip-flops' Inputs			
A	B	x	A^+	B^+	J_A	K_A	J_B	K_B
0	0	0	0	0				
0	0	1	0	1				
0	1	0	0	0				
0	1	1	1	0				
1	0	0	1	0				
1	0	1	1	1				
1	1	0	1	1				
1	1	1	0	0				

SEQUENTIAL CIRCUITS DESIGN

FSM Excitation Table

Step 3

					Next State control gates' out = Flip-flops' Inputs			
A	B	x	A^+	B^+	J_A	K_A	J_B	K_B
0	0	0	0	0	0	x		
0	0	1	0	1				
0	1	0	0	0				
0	1	1	1	0				
1	0	0	1	0				
1	0	1	1	1				
1	1	0	1	1				
1	1	1	0	0				

Q^n	Q^{n+1}	J^n	K^n
0	0	0	x
0	1	1	x
1	0	x	1
1	1	x	0

JK FF
Excitation Table

SEQUENTIAL CIRCUITS DESIGN

FSM Excitation Table

Step 3

					Next State control gates' out = Flip-flops' Inputs			
A	B	x	A^+	B^+	J_A	K_A	J_B	K_B
0	0	0	0	0	0	x	0	x
0	0	1	0	1				
0	1	0	0	0				
0	1	1	1	0				
1	0	0	1	0				
1	0	1	1	1				
1	1	0	1	1				
1	1	1	0	0				

Q^n	Q^{n+1}	J^n	K^n
0	0	0	x
0	1	1	x
1	0	x	1
1	1	x	0

JK FF
Excitation Table

SEQUENTIAL CIRCUITS DESIGN

FSM Excitation Table

Step 3

Next State control gates' out = Flip-flops' Inputs

A	B	x	A^+	B^+	J_A	K_A	J_B	K_B
0	0	0	0	0	0	x	0	x
0	0	1	0	1	0	x	1	x
0	1	0	0	0				
0	1	1	1	0				
1	0	0	1	0				
1	0	1	1	1				
1	1	0	1	1				
1	1	1	0	0				

Q^n	Q^{n+1}	J^n	K^n
0	0	0	x
0	1	1	x
1	0	x	1
1	1	x	0

JK FF
Excitation Table

SEQUENTIAL CIRCUITS DESIGN Step 3

Efficient Derivation of JK Excitation Equations

Q^n	Q^{n+1}	J^n	K^n
0	0	0	x
0	1	1	x
1	0	x	1
1	1	x	0

Q^n	Q^{n+1}	J^n	K^n
0	0	0	x
0	1	1	x
1	0	x	1
1	1	x	0

J <= Q^{n+1} if $Q^n = 0$ (*copy next state where present state is* 0)

x if $Q^n = 1$ (*put* <u>don't care</u> *where present state is* 1)

K <= $\overline{Q}^{n+1}$ if $Q^n = 1$ (*copy complement of the next state where present state is* 1)

x if $Q^n = 0$ (*put* <u>don't care</u> *where present state is* 0)

NOTE: Use this "recipe" just to expedite generation of the JK input excitation functions of a given FSM. <u>Boolean expressions of J and K has to be expressed in terms of available signals, like present states (Q^n) and inputs (X), **not next states (Q^{n+1})**, which of course are not available in the present!!!</u>

SEQUENTIAL CIRCUITS DESIGN

FSM Excitation Table (J_A)

Step 3

					Next State control gates' out = Flip-flops' Inputs			
A	B	x	A^+	B^+	J_A	K_A	J_B	K_B
0	0	0	0	0	0	x	0	x
0	0	1	0	1	0	x	1	x
0	1	0	0	0	0			
0	1	1	1	0	1			
1	0	0	1	0	x			
1	0	1	1	1	x			
1	1	0	1	1	x			
1	1	1	0	0	x			

Q^n	Q^{n+1}	J^n	K^n
0	0	0	x
0	1	1	x
1	0	x	1
1	1	x	0

J => copy Q^{n+1} if Q^n= 0

fill with x if Q^n= 1

SEQUENTIAL CIRCUITS DESIGN

FSM Excitation Table (J_B) Step 3

Next State control gates' out = Flip-flops' Inputs

A	B	x	A^+	B^+	J_A	K_A	J_B	K_B
0	0	0	0	0	0	x	0	
0	0	1	0	1	0	x	1	
0	1	0	0	0	0		x	
0	1	1	1	0	1		x	
1	0	0	1	0	x		0	
1	0	1	1	1	x		1	
1	1	0	1	1	x		x	
1	1	1	0	0	x		x	

Q^n	Q^{n+1}	J^n	K^n
0	0	0	x
0	1	1	x
1	0	x	1
1	1	x	0

$$J = \begin{cases} Q^{n+1} & \text{if } Q^n = 0 \\ x & \text{if } Q^n = 1 \end{cases}$$

SEQUENTIAL CIRCUITS DESIGN

FSM Excitation Table (K_A)

Step 3

					Next State control gates' out = Flip-flops' Inputs			
A	B	x	A^+	B^+	J_A	K_A	J_B	K_B
0	0	0	0	0	0	x	0	
0	0	1	0	1	0	x	0	
0	1	0	0	0	0	x	x	
0	1	1	1	0	1	x	x	
1	0	0	1	0	x	0	0	
1	0	1	1	1	x	0	1	
1	1	0	1	1	x	0	x	
1	1	1	0	0	x	1	x	

Q^n	Q^{n+1}	J^n	K^n
0	0	0	x
0	1	1	x
1	0	x	1
1	1	x	0

$$K = \begin{cases} \overline{Q}^{n+1} & \text{if } Q^n = 1 \\ x & \text{if } Q^n = 0 \end{cases}$$

SEQUENTIAL CIRCUITS DESIGN

FSM Excitation Table (K_B)

Step 3

Next State control gates' out = JK Flip-flops' Inputs

A	B	x	A^+	B^+	J_A	K_A	J_B	K_B
0	0	0	0	0	0	x	0	x
0	0	1	0	1	0	x	0	x
0	1	0	0	0	0	x	x	1
0	1	1	1	0	1	x	x	1
1	0	0	1	0	x	0	0	x
1	0	1	1	1	x	0	1	x
1	1	0	1	1	x	0	x	0
1	1	1	0	0	x	1	x	1

Q^n	Q^{n+1}	J^n	K^n
0	0	0	x
0	1	1	x
1	0	x	1
1	1	x	0

$$K = \begin{cases} \overline{Q}^{n+1} & \text{if } Q^n = 1 \\ x & \text{if } Q^n = 0 \end{cases}$$

Excitation Equations & Implementation

Present State S^n		In			Next State control gates' out = Flip-flops' Inputs			
A	B	x	A^+	B^+	J_A	K_A	J_B	K_B
0	0	0	0	0	0	x	0	x
0	0	1	0	1	0	x	1	x
0	1	0	0	0	0	x	x	1
0	1	1	1	0	1	x	x	1
1	0	0	1	0	x	0	0	x
1	0	1	1	1	x	0	1	x
1	1	0	1	1	x	0	x	0
1	1	1	0	0	x	1	x	1

Step 4

J_A

A \ Bx	00	01	11	10
0	0^0	0^1	1^3	0^2
1	x^4	x^5	x^7	x^6

$J_A = Bx$

K_A

A \ Bx	00	01	11	10
0	x^0	x^1	x^3	x^2
1	0^4	0^5	1^7	0^6

$K_A = Bx$

J_B

A \ Bx	00	01	11	10
0	0^0	1^1	x^3	x^2
1	0^4	1^5	x^7	x^6

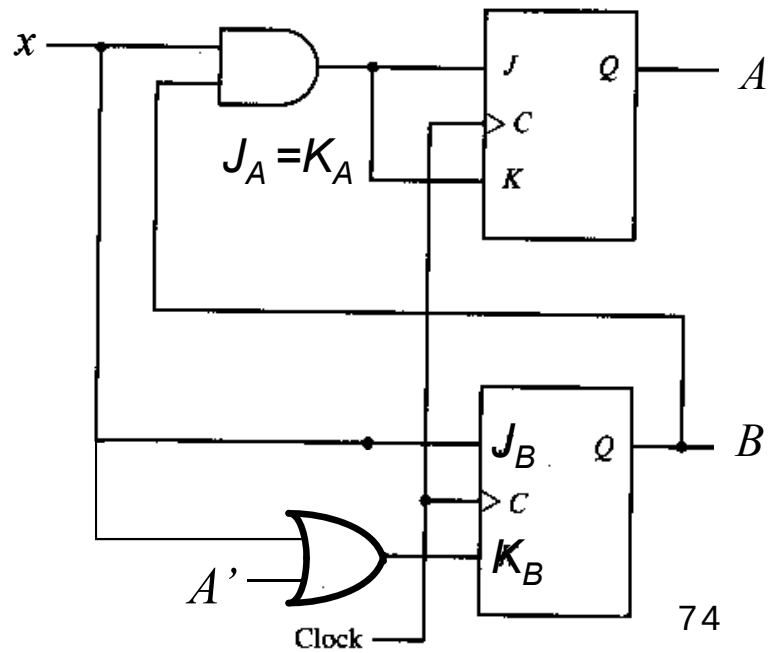
$J_B = x$

K_B

A \ Bx	00	01	11	10
0	x^0	x^1	1^3	1^2
1	x^4	x^5	1^7	0^6

$K_B = x + A'$

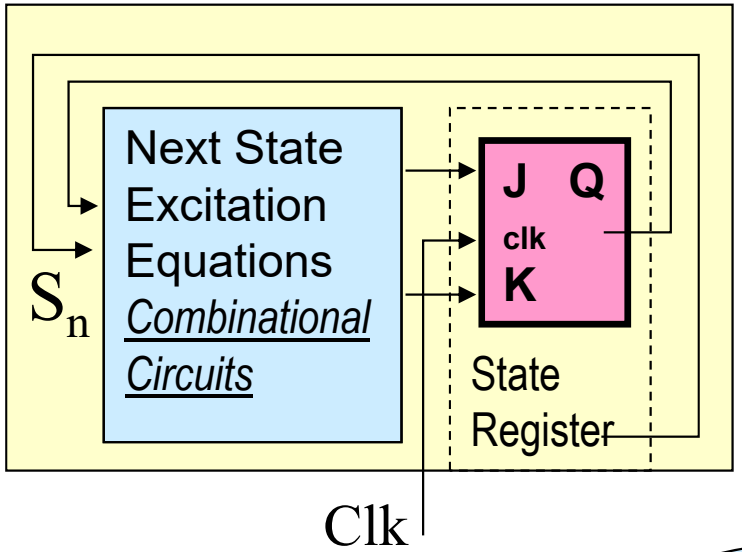
Step 5



Synchronous 4-bit Counter Design

Step 1

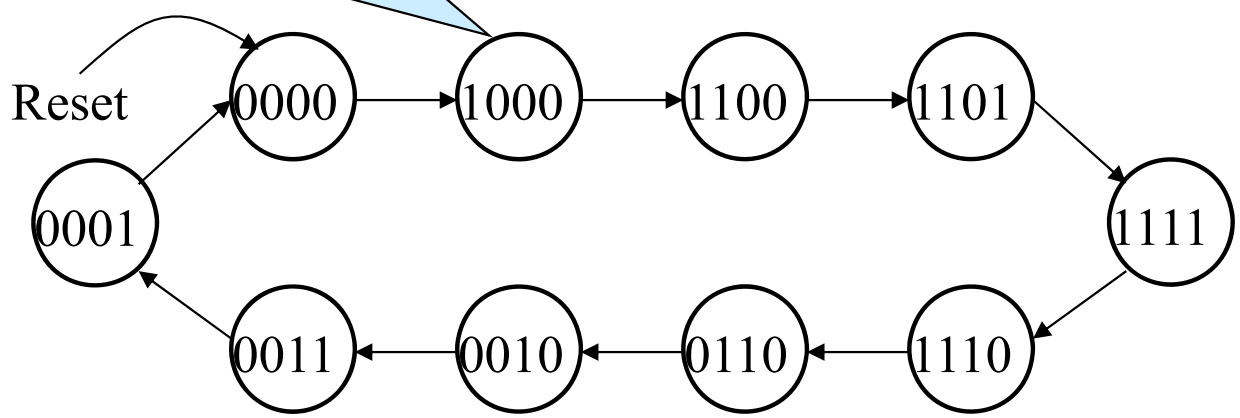
Sequential Circuit Block Diagram



JK-FF
Excitation Table

Q^n	Q^{n+1}	J^n	K^n
0	0	0	x
0	1	1	x
1	0	x	1
1	1	x	0

State Diagram



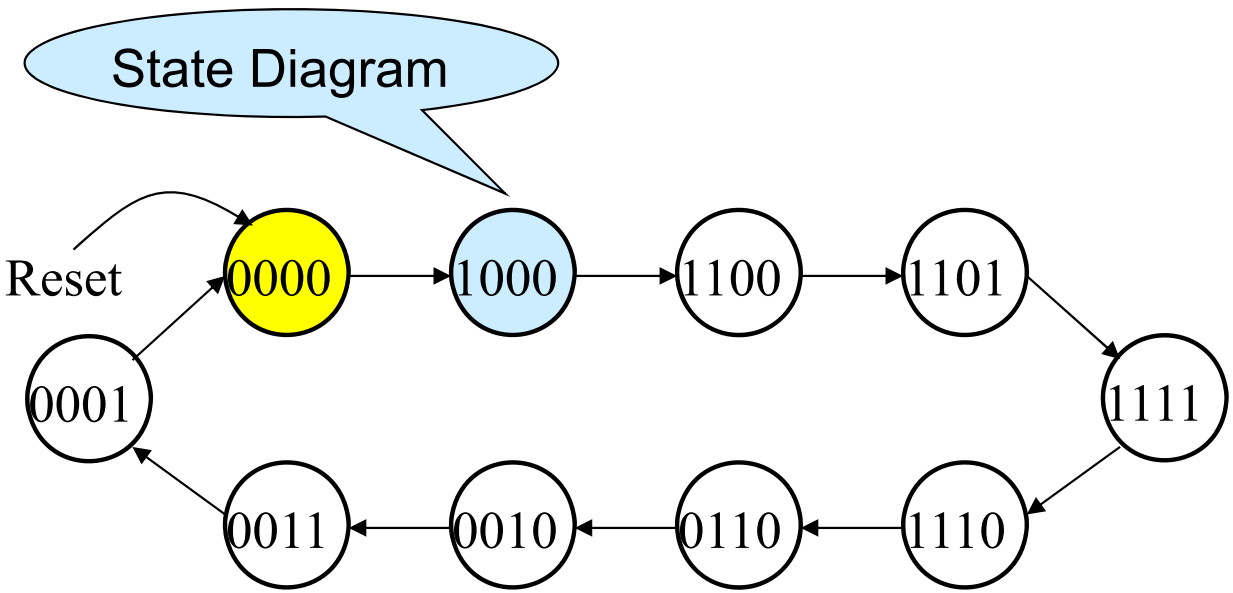
Transition Table

Present state S^n				Next state S^{n+1}			
Q_3	Q_2	Q_1	Q_0	Q_3^+	Q_2^+	Q_1^+	Q_0^+
0	0	0	0	1	0	0	0
0	0	0	1	0	0	0	0
0	0	1	0	0	0	1	1
0	0	1	1	0	0	0	1
0	1	0	0	**x**	**x**	**x**	**x**
0	1	0	1	**x**	**x**	**x**	**x**
0	1	1	0	0	0	1	0
0	1	1	1	**x**	**x**	**x**	**x**
1	0	0	0	1	1	0	0
1	0	0	1	**x**	**x**	**x**	**x**
1	0	1	0	**x**	**x**	**x**	**x**
1	0	1	1	**x**	**x**	**x**	**x**
1	1	0	0	1	1	0	1
1	1	0	1	1	1	1	1
1	1	1	0	0	1	1	0
1	1	1	1	1	1	1	0

Synchronous 4-bit Counter Design

Step 1

To derive the State Table, go through the state diagram, starting from initial state (0000).



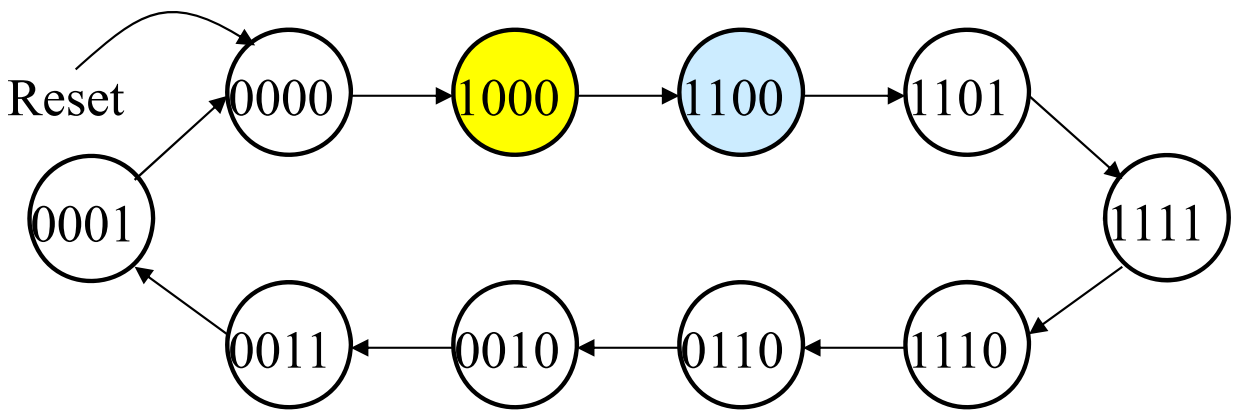
Transition Table

Present state S^n				Next state S^{n+1}			
Q_3	Q_2	Q_1	Q_0	Q_3^+	Q_2^+	Q_1^+	Q_0^+
0	0	0	0	1	0	0	0
0	0	0	1				
0	0	1	0				
0	0	1	1				
0	1	0	0				
0	1	0	1				
0	1	1	0				
0	1	1	1				
1	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				

Synchronous 4-bit Counter Design

State Diagram

Step 1



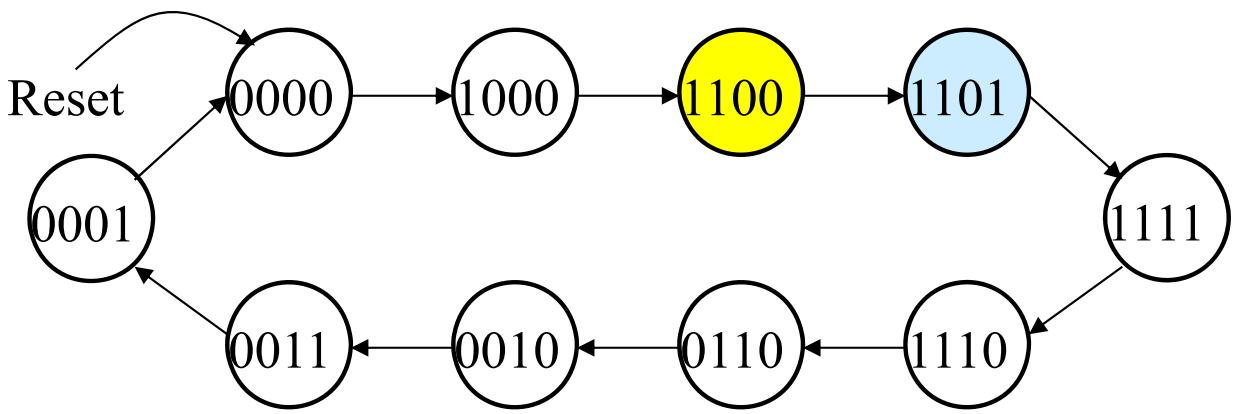
Transition Table

Present state S^n				Next state S^{n+1}			
Q_3	Q_2	Q_1	Q_0	Q_3^+	Q_2^+	Q_1^+	Q_0^+
0	**0**	**0**	**0**	**1**	**0**	**0**	**0**
0	0	0	1				
0	0	1	0				
0	0	1	1				
0	1	0	0				
0	1	0	1				
0	1	1	0				
0	1	1	1				
1	**0**	**0**	**0**	**1**	**1**	**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				

Synchronous 4-bit Counter Design

State Diagram

Step 1



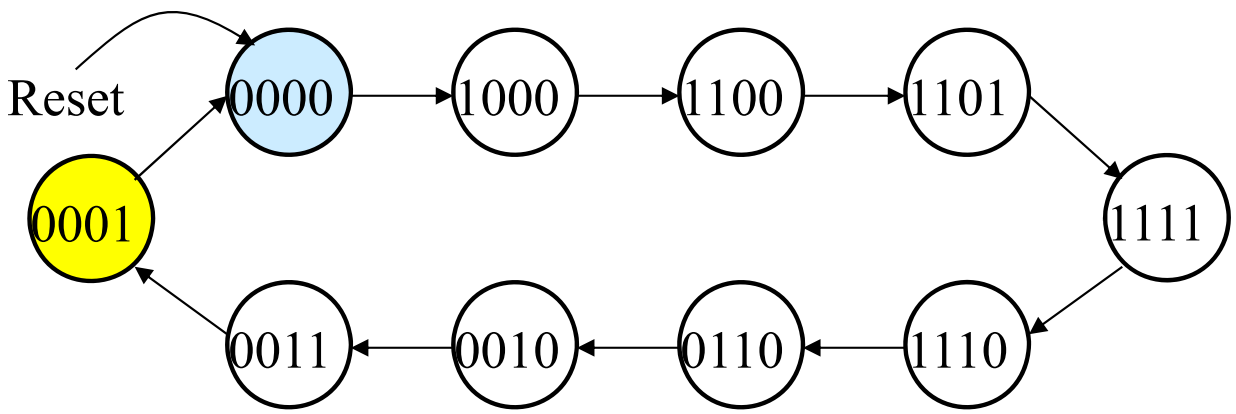
Transition Table

Present state S^n				Next state S^{n+1}			
Q_3	Q_2	Q_1	Q_0	Q_3^+	Q_2^+	Q_1^+	Q_0^+
0	**0**	**0**	**0**	**1**	**0**	**0**	**0**
0	0	0	1				
0	0	1	0				
0	0	1	1				
0	1	0	0				
0	1	0	1				
0	1	1	0				
0	1	1	1				
1	**0**	**0**	**0**	**1**	**1**	**0**	**0**
1	0	0	1				
1	0	1	0				
1	0	1	1				
1	**1**	**0**	**0**	**1**	**1**	**0**	**1**
1	1	0	1				
1	1	1	0				
1	1	1	1				

Synchronous 4-bit Counter Design

State Diagram

Step 1



Transition Table

Present state S^n				Next state S^{n+1}			
Q_3	Q_2	Q_1	Q_0	Q_3^+	Q_2^+	Q_1^+	Q_0^+
0	0	0	0	1	0	0	0
0	0	0	1	0	0	0	0
0	0	1	0	0	0	1	1
0	0	1	1	0	0	0	1
0	1	0	0				
0	1	0	1				
0	1	1	0	0	0	1	0
0	1	1	1				
1	0	0	0	1	1	0	0
1	0	0	1				
1	0	1	0				
1	0	1	1				
1	1	0	0	1	1	0	1
1	1	0	1	1	1	1	1
1	1	1	0	0	1	1	0
1	1	1	1	1	1	1	0

Synchronous 4-bit Counter Design

Present state S^n				Next state S^{n+1}			
Q_3	Q_2	Q_1	Q_0	Q_3^+	Q_2^+	Q_1^+	Q_0^+
0	0	0	0	1	0	0	0
0	0	0	1	0	0	0	0
0	0	1	0	0	0	1	1
0	0	1	1	0	0	0	1
0	1	0	0				
0	1	0	1				
0	1	1	0	0	0	1	0
0	1	1	1				
1	0	0	0	1	1	0	0
1	0	0	1				
1	0	1	0				
1	0	1	1				
1	1	0	0	1	1	0	1
1	1	0	1	1	1	1	1
1	1	1	0	0	1	1	0
1	1	1	1	1	1	1	0

Step 1

Empty cells represent "**Can't happen conditions**" and they can be used as ***don't care*** terms for minimization

Present state S^n				Next state S^{n+1}			
Q_3	Q_2	Q_1	Q_0	Q_3^+	Q_2^+	Q_1^+	Q_0^+
0	0	0	0	1	0	0	0
0	0	0	1	0	0	0	0
0	0	1	0	0	0	1	1
0	0	1	1	0	0	0	1
0	1	0	0	x	x	x	x
0	1	0	1	x	x	x	x
0	1	1	0	0	0	1	0
0	1	1	1	x	x	x	x
1	0	0	0	1	1	0	0
1	0	0	1	x	x	x	x
1	0	1	0	x	x	x	x
1	0	1	1	x	x	x	x
1	1	0	0	1	1	0	1
1	1	0	1	1	1	1	1
1	1	1	0	0	1	1	0
1	1	1	1	1	1	1	0

Excitation Table

Step 3

Present state S^n				Next state S^{n+1}				Q_3 input		Q_2 input		Q_1 input		Q_0 input	
Q_3	Q_2	Q_1	Q_0	Q_3^+	Q_2^+	Q_1^+	Q_0^+	J_3	K_3	J_2	K_2	J_1	K_1	J_0	K_0
0	0	0	0	1	0	0	0	1	x	0	x	0	x	0	x
0	0	0	1	0	0	0	0	0	x	0	x	0	x	x	1
0	0	1	0	0	0	1	1	0	x	0	x	x	0	1	x
0	0	1	1	0	0	0	1	0	x	0	x	x	1	x	0
0	1	0	0	**x**	**x**	**x**	**x**	**x**	**x**	**x**	**x**	**x**	**x**	**x**	**x**
0	1	0	1	**x**	**x**	**x**	**x**	**x**	**x**	**x**	**x**	**x**	**x**	**x**	**x**
0	1	1	0	0	0	1	0	0	x	x	1	x	0	0	x
0	1	1	1	**x**	**x**	**x**	**x**	**x**	**x**	**x**	**x**	**x**	**x**	**x**	**x**
1	0	0	0	1	1	0	0	x	0	1	x	0	x	0	x
1	0	0	1	**x**	**x**	**x**	**x**	**x**	**x**	**x**	**x**	**x**	**x**	**x**	**x**
1	0	1	0	**x**	**x**	**x**	**x**	**x**	**x**	**x**	**x**	**x**	**x**	**x**	**x**
1	0	1	1	**x**	**x**	**x**	**x**	**x**	**x**	**x**	**x**	**x**	**x**	**x**	**x**
1	1	0	0	1	1	0	1	x	0	x	0	0	x	1	x
1	1	0	1	1	1	1	1	x	0	x	0	1	x	x	0
1	1	1	0	0	1	1	0	x	1	x	0	x	0	0	x
1	1	1	1	1	1	1	0	x	0	x	0	x	0	x	1

Synchronous 4-bit Counter Design

Excitation Equations

Step 4

Q_3Q_2 \ Q_1Q_0	00	01	11	10
00	1	0	0	0
01	**x**	**x**	**x**	
11	x	x	x	x
10	x	**x**	**x**	**x**

$$J_3 = \overline{Q_1} \cdot \overline{Q_0}$$

Q_3Q_2 \ Q_1Q_0	00	01	11	10
00	x	x	x	x
01	**x**	**x**	**x**	x
11	0	0	0	1
10	0	**x**	**x**	**x**

$$K_3 = Q_1 \cdot \overline{Q_0}$$

Q_3Q_2 \ Q_1Q_0	00	01	11	10
00	0	0	0	0
01	**x**	**x**	**x**	x
11	x	x	x	x
10	1	**x**	**x**	**x**

$$J_2 = Q_3$$

Q_3Q_2 \ Q_1Q_0	00	01	11	10
00	x	x	x	x
01	**x**	**x**	**x**	1
11	0	0	0	0
10	x	**x**	**x**	**x**

$$K_2 = \overline{Q_3}$$

Q_3Q_2 \ Q_1Q_0	00	01	11	10
00	0	0	x	x
01	**x**	**x**	**x**	x
11	0	1	x	x
10	0	**x**	**x**	**x**

$$J_1 = Q_3 \cdot Q_0$$

Q_3Q_2 \ Q_1Q_0	00	01	11	10
00	x	x	1	0
01	**x**	**x**	**x**	0
11	x	x	0	0
10	x	**x**	**x**	**x**

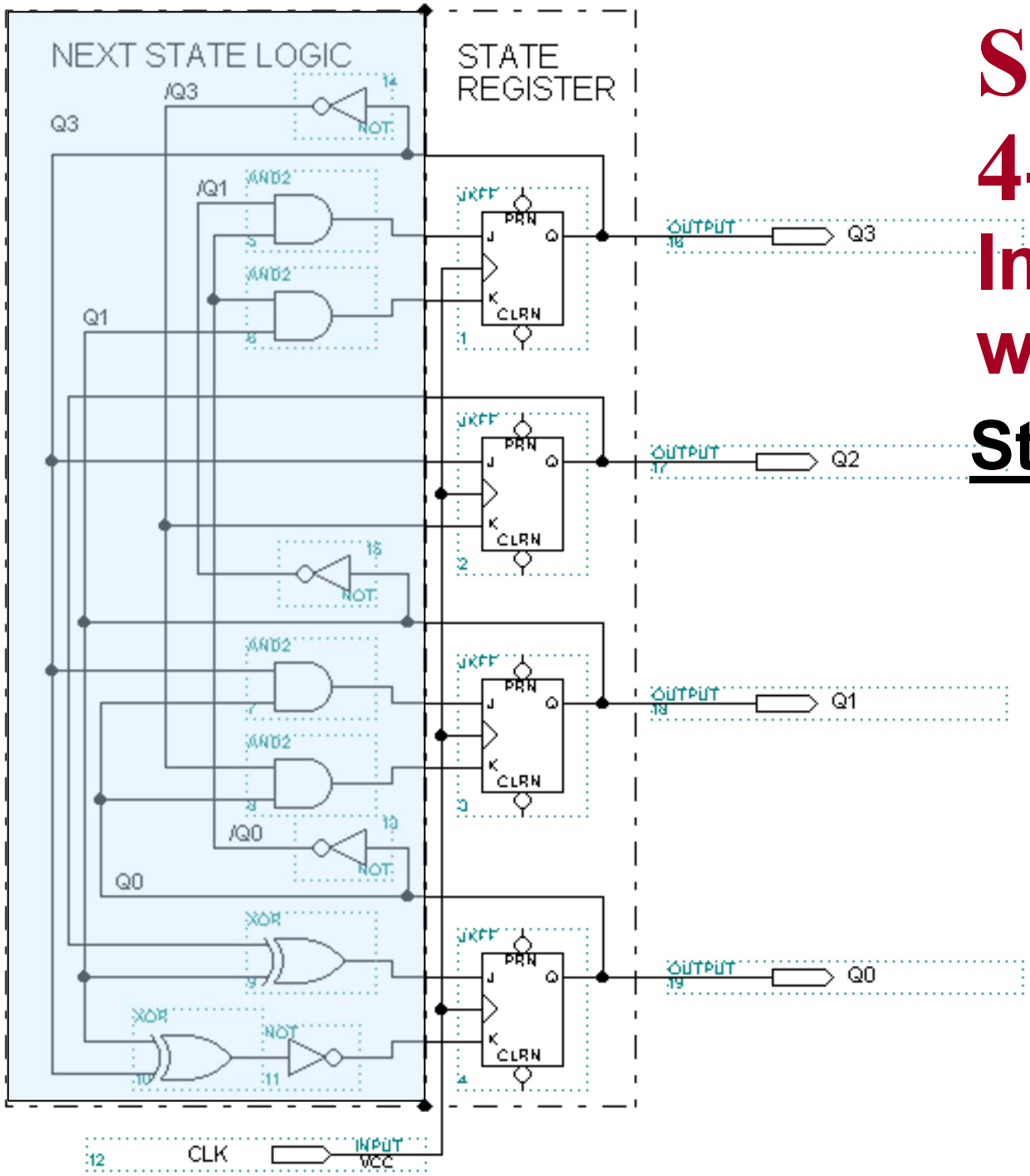
$$K_1 = \overline{Q_3} \cdot Q_0$$

Q_3Q_2 \ Q_1Q_0	00	01	11	10
00	0	x	x	1
01	**x**	**x**	**x**	0
11	1	x	x	0
10	0	**x**	**x**	**x**

$$J_0 = Q_2 \oplus Q_1$$

Q_3Q_2 \ Q_1Q_0	00	01	11	10
00	x	1	0	x
01	**x**	**x**	**x**	x
11	x	0	1	x
10	x	**x**	**x**	**x**

$$K_0 = \overline{Q_3 \oplus Q_1}$$



Synchronous 4-bit Counter Implementation with JK flip-flops

Step 5

$$J_3 = \overline{Q_1} \cdot \overline{Q_0}$$
$$K_3 = Q_1 \cdot \overline{Q_0}$$

$$J_2 = Q_3$$
$$K_2 = \overline{Q_3}$$

$$J_1 = Q_3 \cdot Q_0$$
$$K_1 = \overline{Q_3} \cdot Q_0$$

$$J_0 = Q_2 \oplus Q_1$$
$$K_0 = \overline{Q_3 \oplus Q_1}$$