Principles of Computer Architecture

Miles Murdocca and Vincent Heuring

Appendix B: Reduction of Digital Logic

Chapter Contents

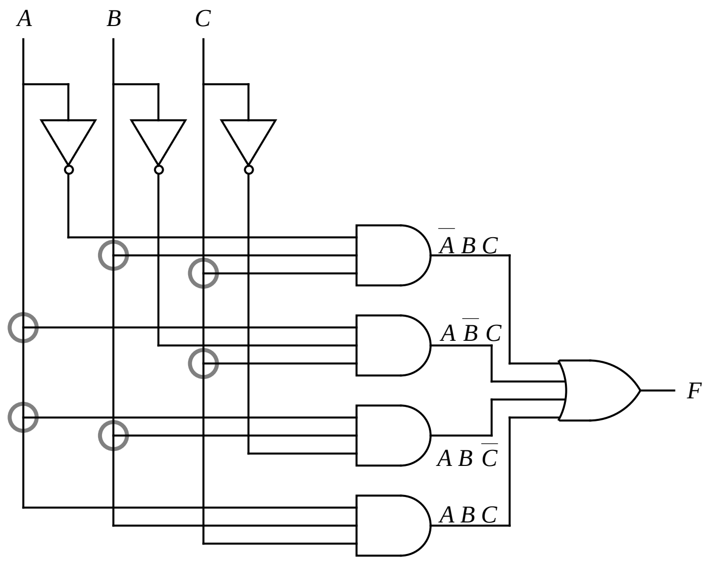
Reduction (Simplification) of Boolean Expressions

- It is usually possible to simplify the canonical SOP (or POS) forms.
- A smaller Boolean equation generally translates to a lower gate count in the target circuit.
- We cover three methods: algebraic reduction, Karnaugh map reduction, and tabular (Quine-McCluskey) reduction.

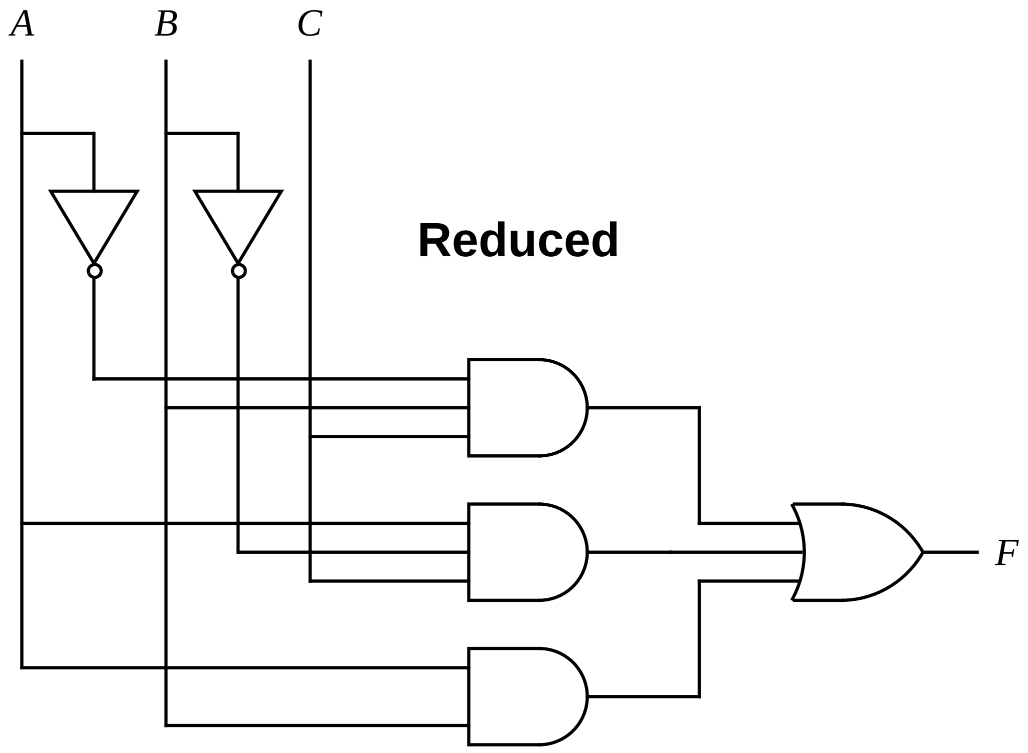
Reduced Majority Function Circuit

- Compared with the AND-OR circuit for the unreduced majority function, the inverter for C has been eliminated, one AND gate has been eliminated, and one AND gate has only two inputs instead of three inputs. Can the function by reduced further? How do we go about it?

Unreduced



Reduced



The Algebraic Method

- **Consider the majority function, *F*. We apply the algebraic method to reduce *F* to its minimal two-level form:**

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

$F = \overline{A}BC + A\overline{B}C + AB(\overline{C} + C)$ Distributive Property

$F = \overline{A}BC + A\overline{B}C + AB(1)$ Complement Property

$F = \overline{A}BC + A\overline{B}C + AB$ Identity Property

$F = \overline{A}BC + A\overline{B}C + AB + ABC$ Idempotence

$F = \overline{A}BC + AC(\overline{B} + B) + AB$ Identity Property

$F = \overline{A}BC + AC + AB$ Complement and Identity

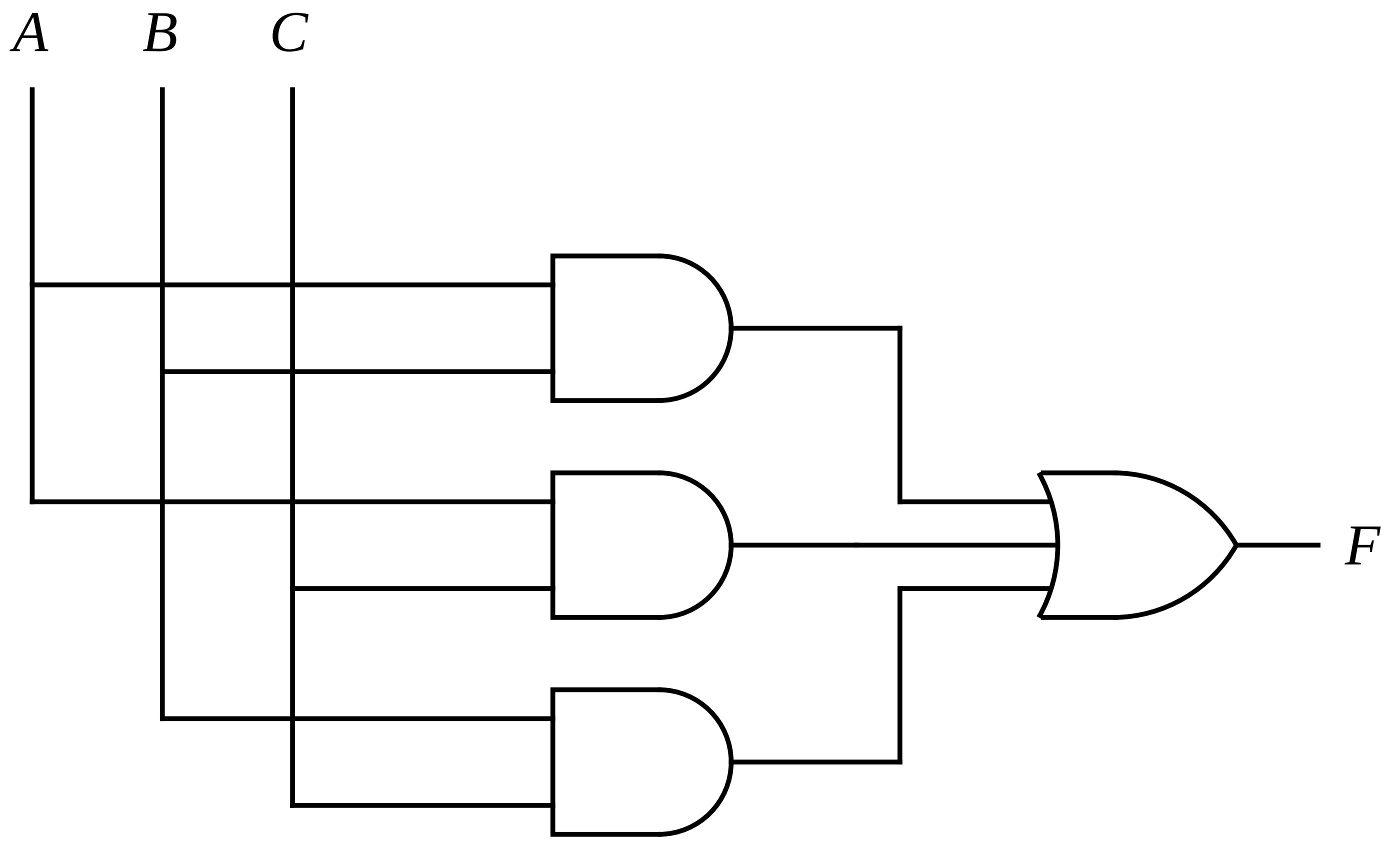
$F = \overline{A}BC + AC + AB + ABC$ Idempotence

$F = BC(\overline{A} + A) + AC + AB$ Distributive

$F = BC + AC + AB$ Complement and Identity

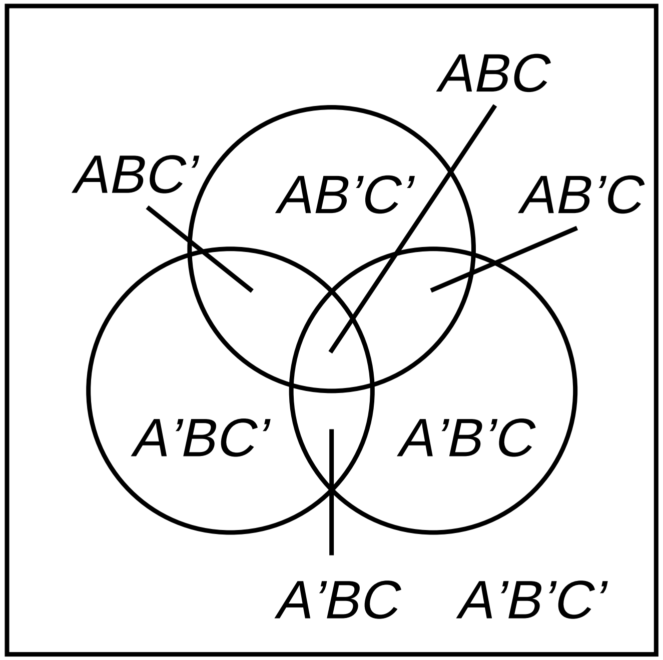
The Algebraic Method

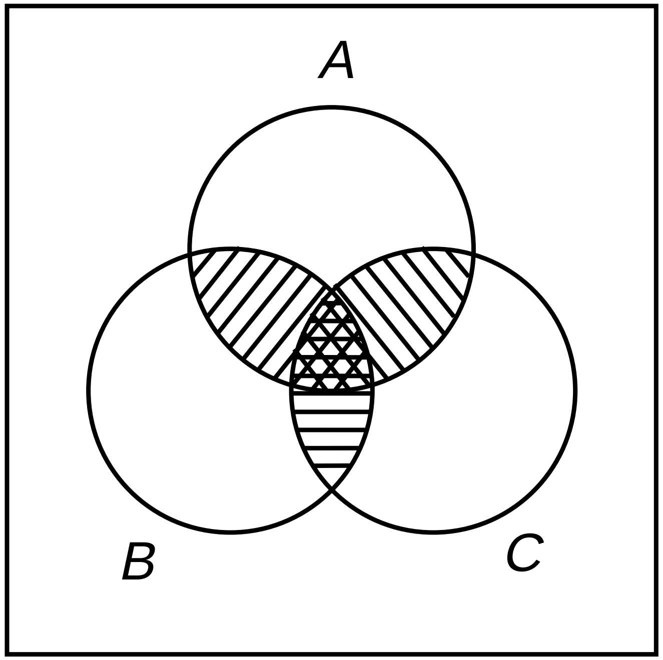
- This majority circuit is functionally equivalent to the previous majority circuit, but this one is in its minimal two-level form:



Karnaugh Maps: Venn Diagram Representation of Majority Function

- **Each distinct region in the “Universe” represents a minterm.**
- **This diagram can be transformed into a *Karnaugh Map*.**





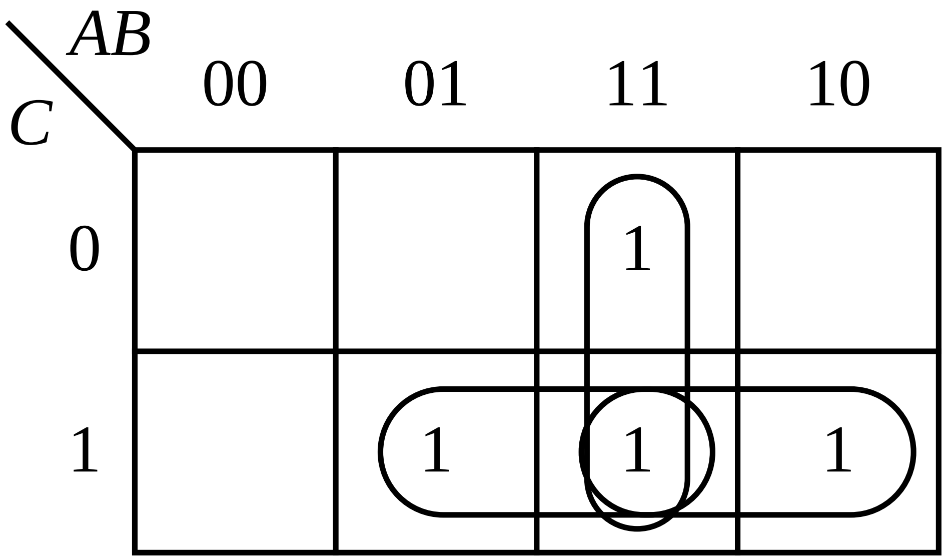
K-Map for Majority Function

- Place a “1” in each cell that corresponds to that minterm.
- Cells on the outer edge of the map “wrap around”

Minterm Index	A	B	C	F
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	1
6	1	1	0	1
7	1	1	1	1

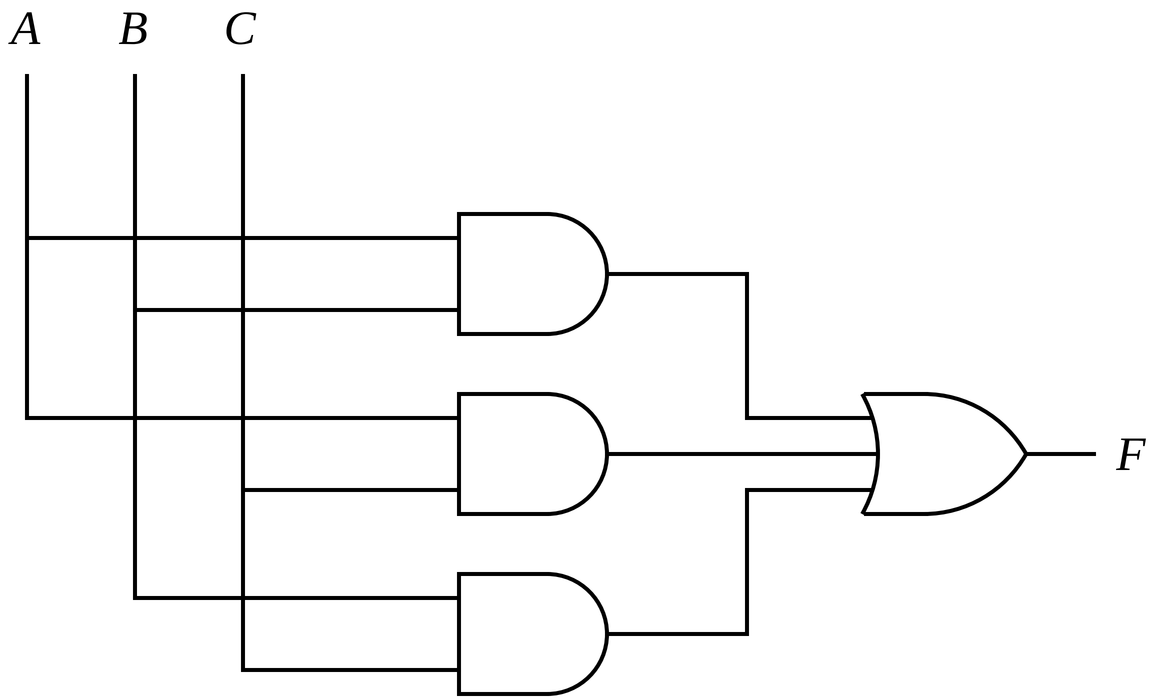
$C \backslash AB$	00	01	11	10
0			1	
1		1	1	1

Adjacency Groupings for Majority Function



- **F = BC + AC + AB**

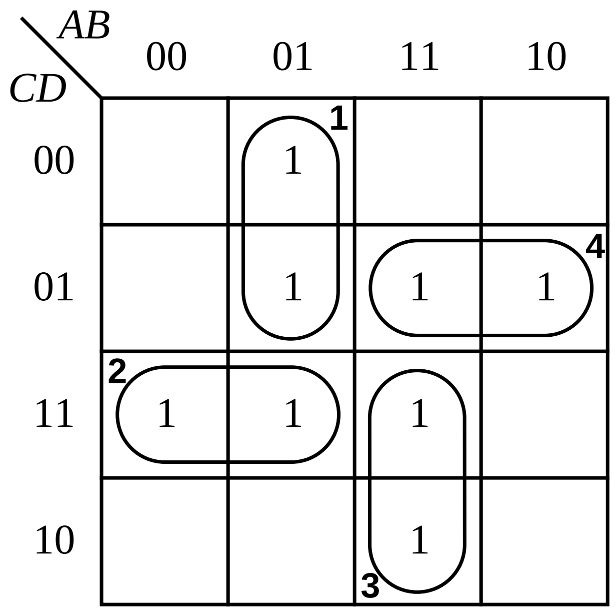
Minimized AND-OR Majority Circuit

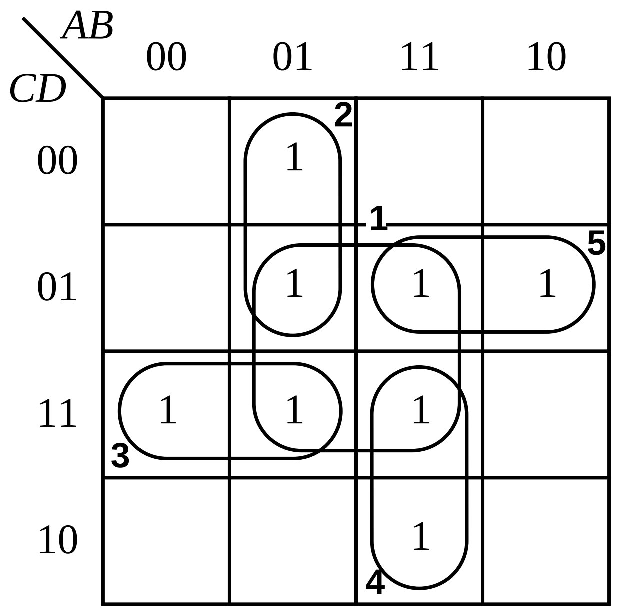


- **F = BC + AC + AB**
- **The K-map approach yields the same minimal two-level form as the algebraic approach.**

K-Map Groupings

- **Minimal grouping is on the left, non-minimal (but logically equivalent) grouping is on the right.**
- **To obtain minimal grouping, create *smallest* groups first.**

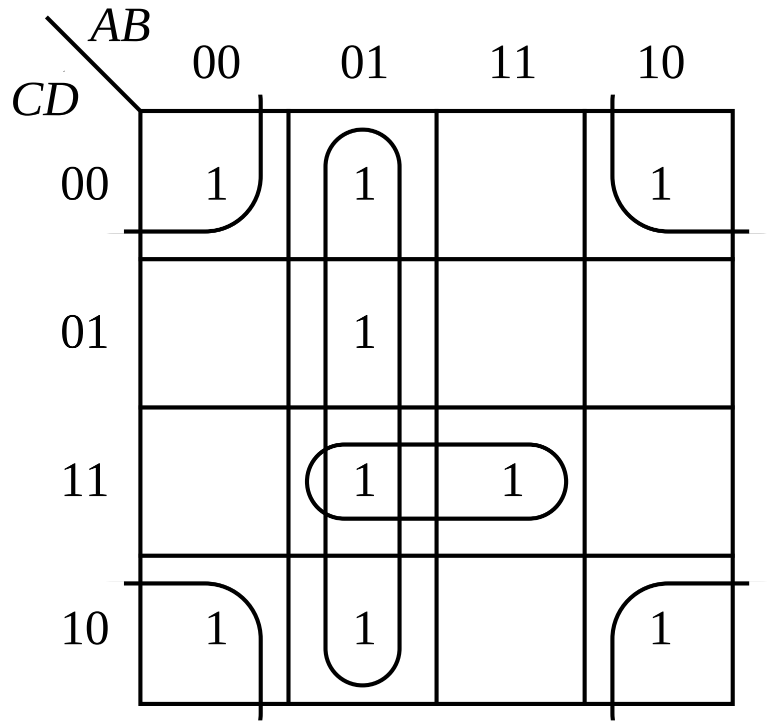




$$F = \overline{A}\,B\,\overline{C} + \overline{A}\,C\,D + A\,B\,C + A\,\overline{C}\,D$$

$$F = B\,D + \overline{A}\,B\,\overline{C} + \overline{A}\,C\,D + A\,B\,C + A\,\overline{C}\,D$$

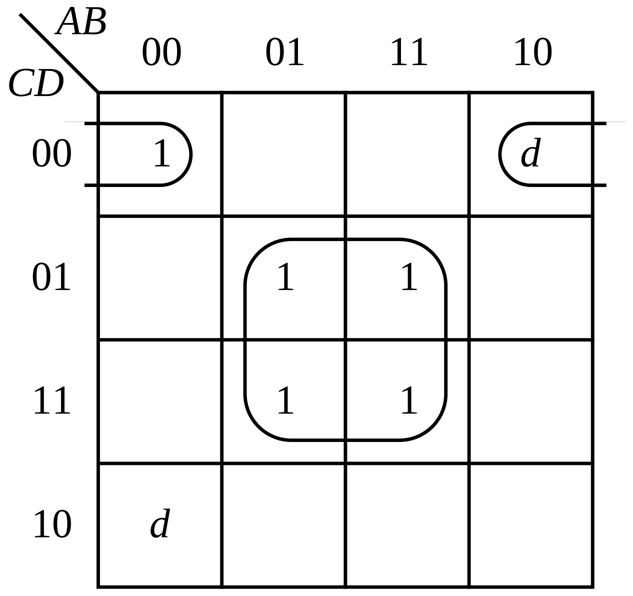
K-Map Corners are Logically Adjacent



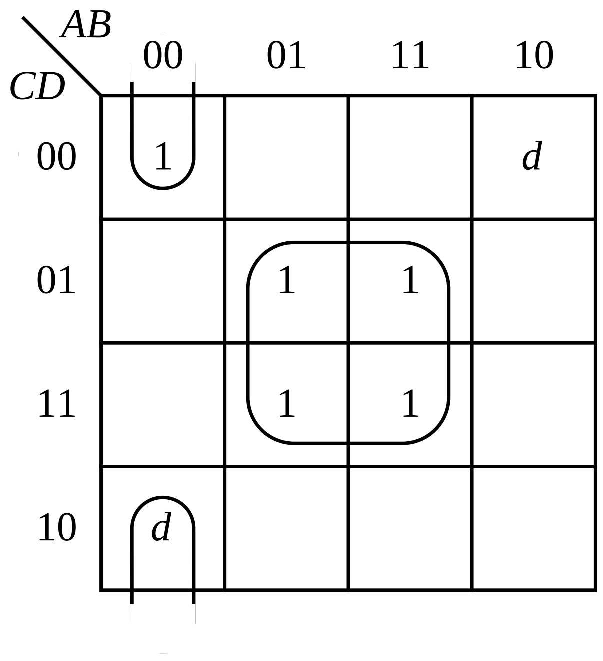
$$F = B\,C\,D + \overline{B}\,\overline{D} + \overline{A}\,B$$

K-Maps and Don’t Cares

- **There can be more than one minimal grouping, as a result of don’t cares.**



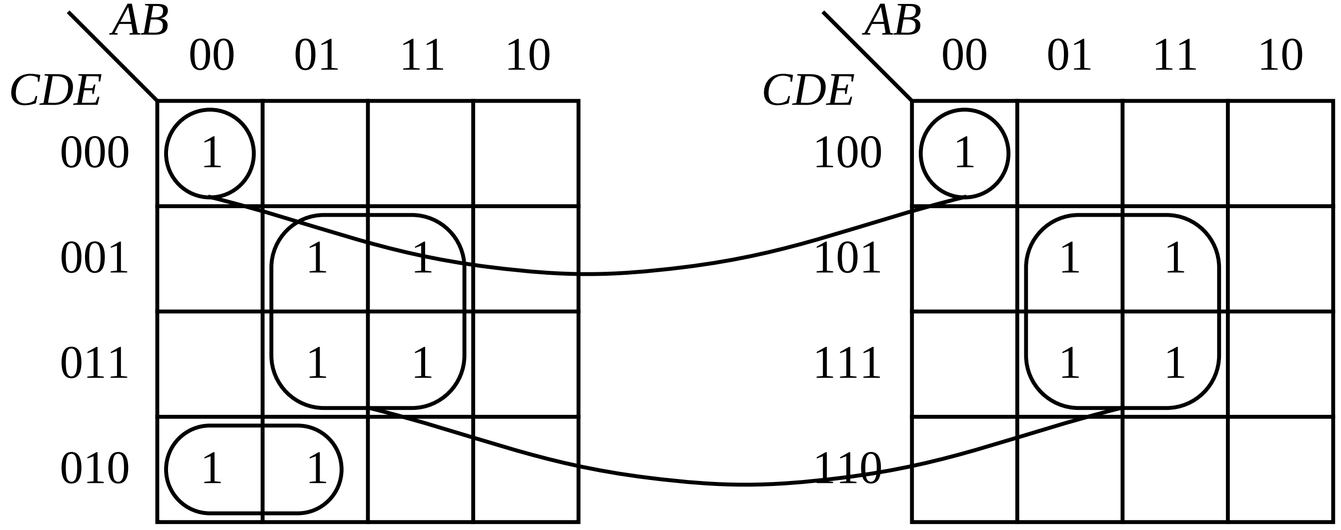
$$F = \overline{B}\,\overline{C}\,\overline{D} + BD$$



$$F = \overline{A}\,\overline{B}\,\overline{D} + BD$$

Five-Variable K-Map

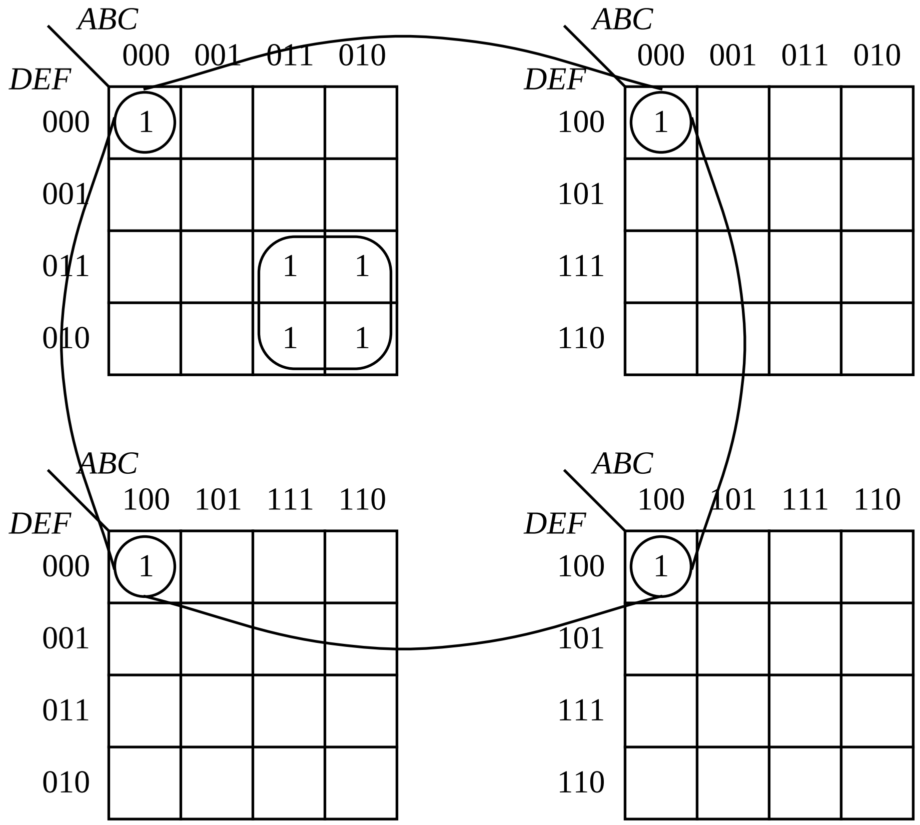
- **Visualize two 4-variable K-maps stacked one on top of the other; groupings are made in three dimensional cubes.**



$$F = \overline{A}\,\overline{C}\,D\,\overline{E} + \overline{A}\,\overline{B}\,\overline{D}\,\overline{E} + B\,E$$

Six-Variable K-Map

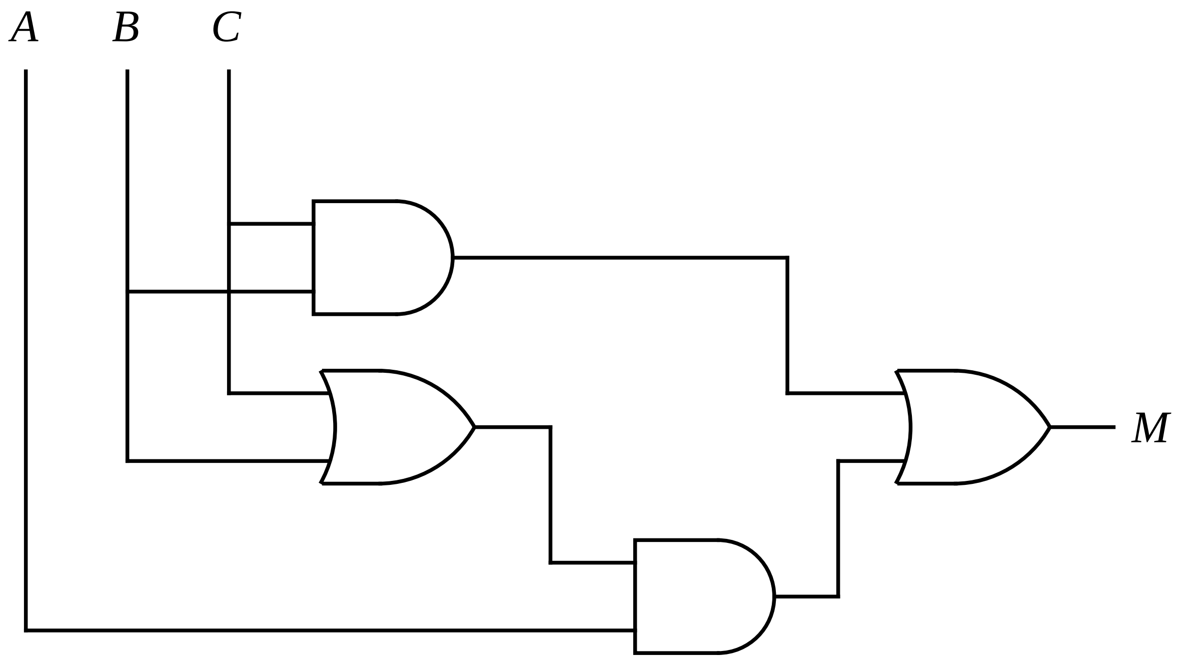
- **Visualize four 4-variable K-maps stacked one on top of the other; groupings are made in three dimensional cubes.**



$$G = \overline{B}\,\overline{C}\,\overline{E}\,\overline{F} + \overline{A}\,B\,\overline{D}\,E$$

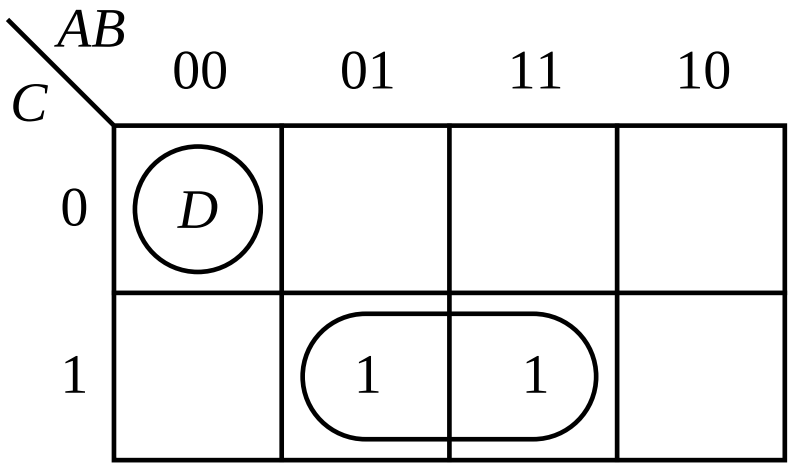
3-Level Majority Circuit

- **K-Kap Reduction results in a reduced two-level circuit (that is, AND followed by OR. Inverters are not included in the two-level count). Algebraic reduction can result in multi-level circuits with even fewer logic gates and fewer inputs to the logic gates.**



Map-Entered Variables

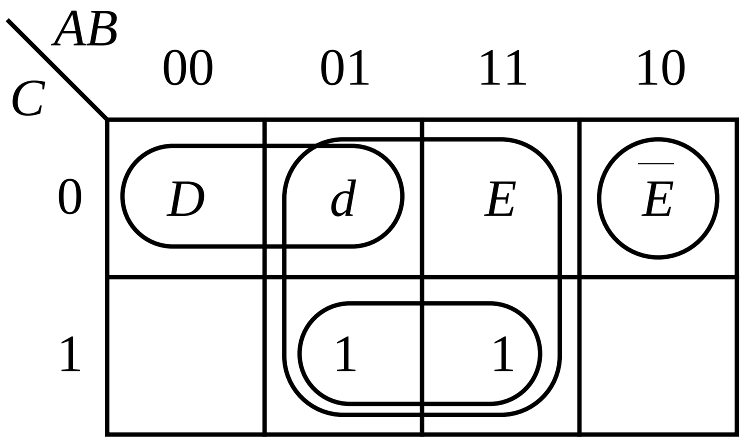
- **An example of a K-map with a map-entered variable *D*.**



$$F = BC + \overline{A}\,\overline{B}\,\overline{C}\,D$$

Two Map-Entered Variables

- **A K-map with two map-entered variables *D* and *E*.**
- $F = BC + \overline{A}\overline{C}D + BE + A\overline{B}\overline{C}\overline{E}$



Truth Table with Don't Cares

- **A truth table representation of a single function with don't cares.**

A	*B*	*C*	*D*	*F*
0	0	0	0	*d*
0	0	0	1	1
0	0	1	0	0
0	0	1	1	1
0	1	0	0	0
0	1	0	1	1
0	1	1	0	1
0	1	1	1	1
1	0	0	0	0
1	0	0	1	0
1	0	1	0	1
1	0	1	1	*d*
1	1	0	0	0
1	1	0	1	1
1	1	1	0	0
1	1	1	1	*d*

Tabular (Quine-McCluskey) Reduction

- Tabular reduction begins by grouping minterms for which *F* is nonzero according to the number of 1's in each minterm. Don't cares are considered to be nonzero.
- The next step forms a consensus (the logical form of a cross product) between each pair of adjacent groups for all terms that differ in only one variable.

Initial setup

A	*B*	*C*	*D*	
0	0	0	0	√
0	0	0	1	√
0	0	1	1	√
0	1	0	1	√
0	1	1	0	√
1	0	1	0	√
0	1	1	1	√
1	0	1	1	√
1	1	0	1	√
1	1	1	1	√

(a)

After first reduction

A	*B*	*C*	*D*	
0	0	0	_	*
0	0	_	1	√
0	_	0	1	√
0	_	1	1	√
_	0	1	1	√
0	1	_	1	√
_	1	0	1	√
0	1	1	_	*
1	0	1	_	*
_	1	1	1	√
1	_	1	1	√
1	1	_	1	√

(b)

After second reduction

A	*B*	*C*	*D*	
0	_	_	1	*
_	_	1	1	*
_	1	_	1	*

(c)

Table of Choice

- **The prime implicants form a set that completely covers the function, although not necessarily minimally.**
- **A *table of choice* is used to obtain a minimal cover set.**

Prime Implicants	Minterms						
	0001	0011	0101	0110	0111	1010	1101
0 0 0 _	√						
* 0 1 1 _				√	√		
* 1 0 1 _						√	
0 _ _ 1	√	√	√		√		
_ _ 1 1		√			√		
* _ 1 _ 1			√		√		√

Reduced Table of Choice

- **In a reduced table of choice, the essential prime implicants and the minterms they cover are removed, producing the *eligible set*.**
- $F = \overline{A}BC + A\overline{B}C + BD + \overline{A}D$

Eligible Set		Minterms	
		0001	0011
X	0 0 0 _	√	
Y	0 _ _ 1	√	√
Z	_ _ 1 1		√

Set 1	**Set 2**
0 0 0 _	0 _ _ 1
_ _ 1 1	

Multiple Output Truth Table

- The power of tabular reduction comes into play for multiple functions, in which minterms can be shared among the functions.

Minterm	A	B	C	F_0	F_1	F_2
m_0	0	0	0	1	0	0
m_1	0	0	1	0	1	0
m_2	0	1	0	0	0	1
m_3	0	1	1	1	1	1
m_4	1	0	0	0	1	0
m_5	1	0	1	0	0	0
m_6	1	1	0	0	1	1
m_7	1	1	1	1	1	1

Multiple Output Table of Choice

$F_0(A,B,C) = \bar{A}\bar{B}\bar{C} + BC$

$F_1(A,B,C) = \bar{A}C + A\bar{C} + BC$

$F_2(A,B,C) = B$

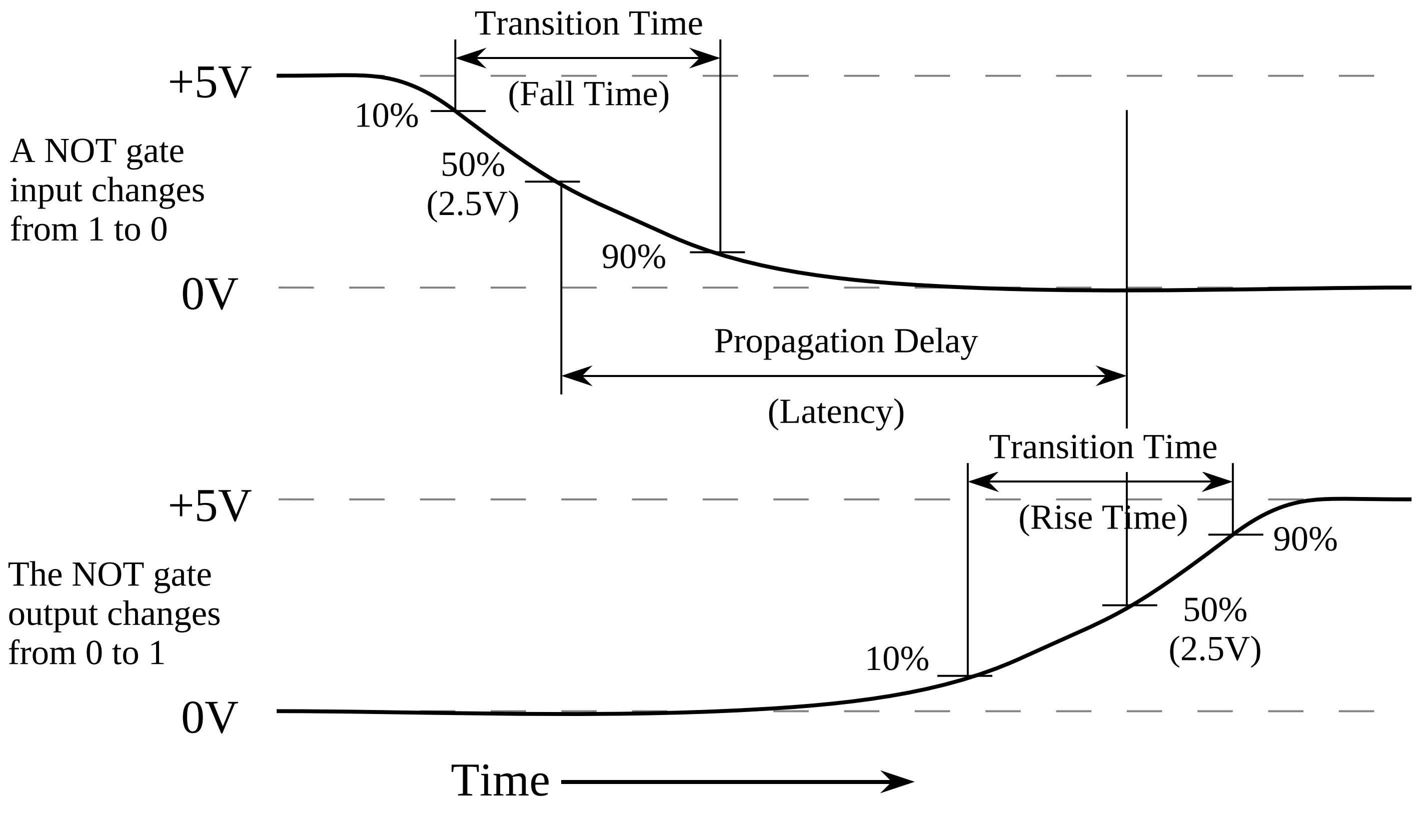
Prime Implicants \ Min-terms		$F_0(A,B,C)$			$F_1(A,B,C)$					$F_2(A,B,C)$			
		m_0	m_3	m_7	m_1	m_3	m_4	m_6	m_7	m_2	m_3	m_6	m_7
F_0	* 0 0 0	√			▨	▨	▨	▨	▨	▨	▨	▨	▨
F_1	* 0 _ 1	▨	▨	▨	√	√				▨	▨	▨	▨
F_1	* 1 _ 0	▨	▨	▨			√	√		▨	▨	▨	▨
F_2	* _ 1 _	▨	▨	▨	▨	▨	▨	▨	▨	√	√	√	√
$F_{1,2}$	1 1 _	▨	▨	▨				√	√			√	√
$F_{0,1,2}$	* _ 1 1		√	√		√			√		√		√

Speed and Performance

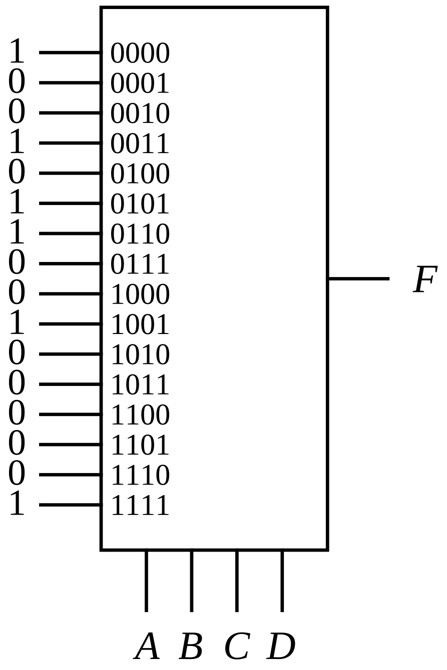
- The speed of a digital system is governed by:
  - the propagation delay through the logic gates and
  - the propagation delay across interconnections.
- We will look at characterizing the delay for a logic gate, and a method of reducing circuit depth using function decomposition.

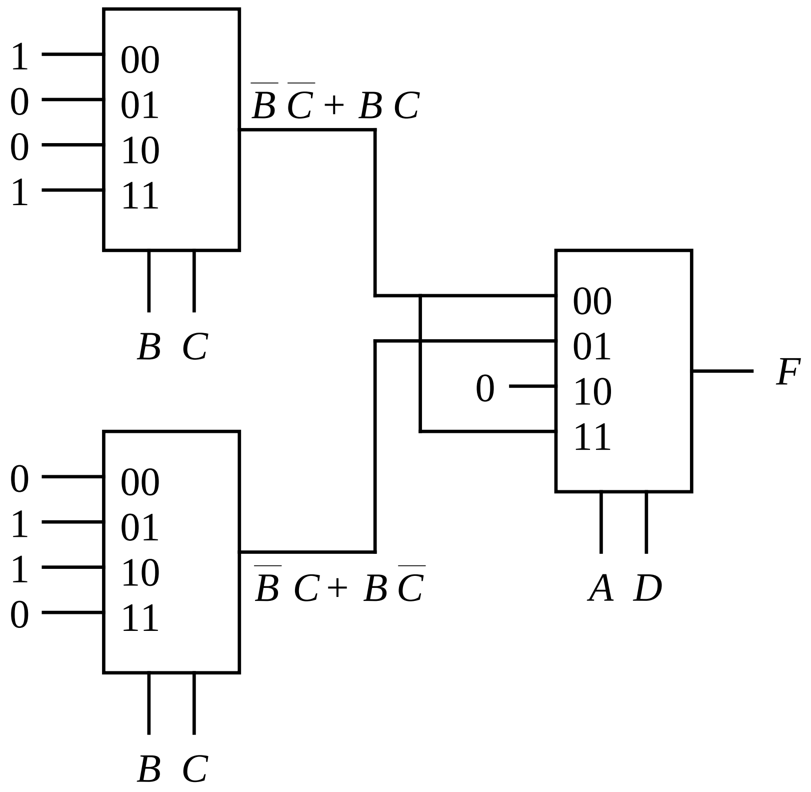
Propagation Delay for a NOT Gate

- (From Hamacher *et. al.* 1990)



MUX Decomposition



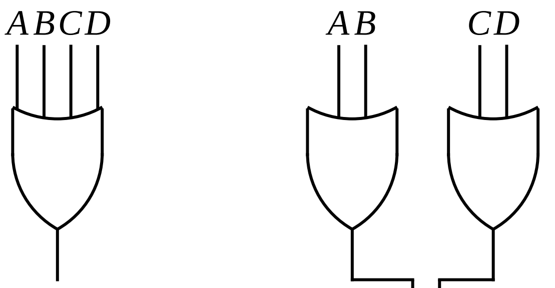

1
0
0
1
00
01
10
11
$\overline{B}\,\overline{C} + B\,C$
B C
0
1
1
0
00
01
10
11
$\overline{B}\,C + B\,\overline{C}$
B C
0
00
01
10
11
F
A D

$$F(ABCD) = \overline{A}\,\overline{B}\,\overline{C}\,\overline{D} + \overline{A}\,\overline{B}CD + \overline{A}B\overline{C}D + \overline{A}BC\overline{D} + A\overline{B}\,\overline{C}D + ABCD$$

$$= (\overline{B}\,\overline{C} + BC)AD + (\overline{B}C + B\overline{C})\overline{A}D + (\overline{B}\,\overline{C} + BC)$$

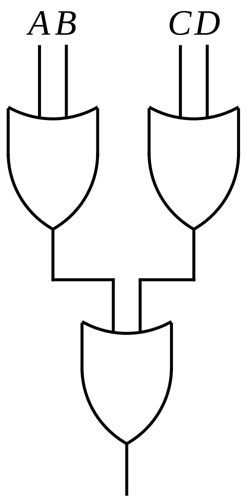
OR-Gate Decomposition

- **Fanin affects circuit depth.**



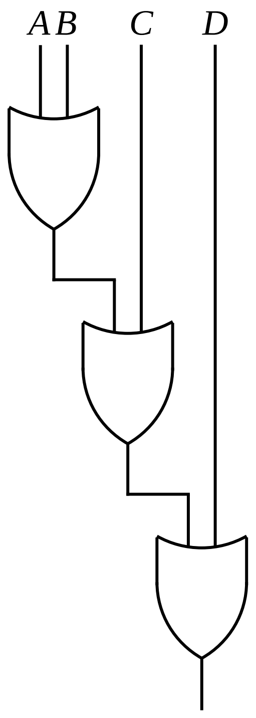
$A + B + C + D$

Initial high fan-in gate



$(A + B) + (C + D)$

Balanced tree



Associative law of Boolean algebra:

$$A + B + C + D = (A + B) + (C + D)$$

$((A + B) + C) + D$

Degenerate tree

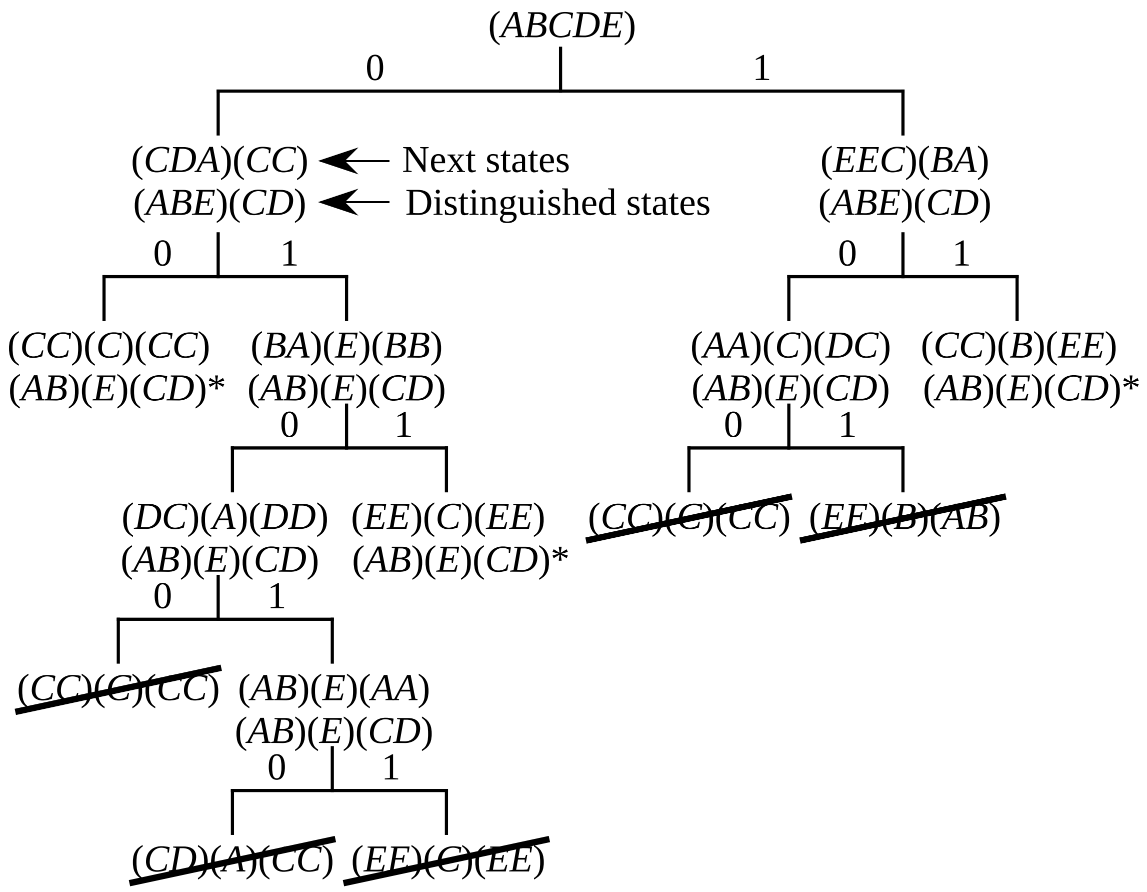
State Reduction

- **Description of state machine M_0 to be reduced.**

Present state \ Input	X = 0	X = 1
A	C/0	E/1
B	D/0	E/1
C	C/1	B/0
D	C/1	A/0
E	A/0	C/1

Distinguishing Tree

- **A next state tree for M_0.**



Reduced State Table

- A reduced state table for machine M_1.

Input / Current state	X = 0	X = 1
$AB: A'$	$B'/0$	$C'/1$
$CD: B'$	$B'/1$	$A'/0$
$E: C'$	$A'/0$	$B'/1$

The State Assignment Problem

- **Two state assignments for machine *M_2*.**

P.S. \ Input	X = 0	X = 1
A	B/1	A/1
B	C/0	D/1
C	C/0	D/0
D	B/1	A/0

Machine M_2

S_0S_1 \ Input	X = 0	X = 1
A: 00	01/1	00/1
B: 01	10/0	11/1
C: 10	10/0	11/0
D: 11	01/1	00/0

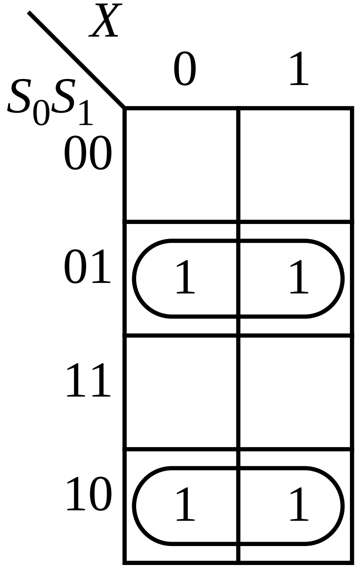
State assignment SA_0

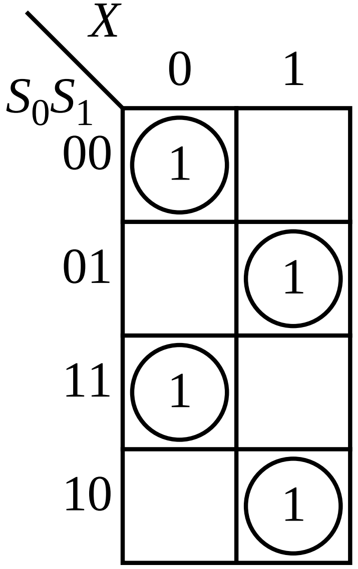
S_0S_1 \ Input	X = 0	X = 1
A: 00	01/1	00/1
B: 01	11/0	10/1
C: 11	11/0	10/0
D: 10	01/1	00/0

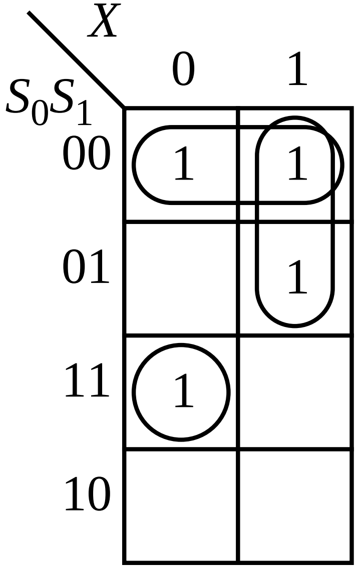
State assignment SA_1

State Assignment SA_0

- **Boolean equations for machine M_2 using state assignment SA_0.**







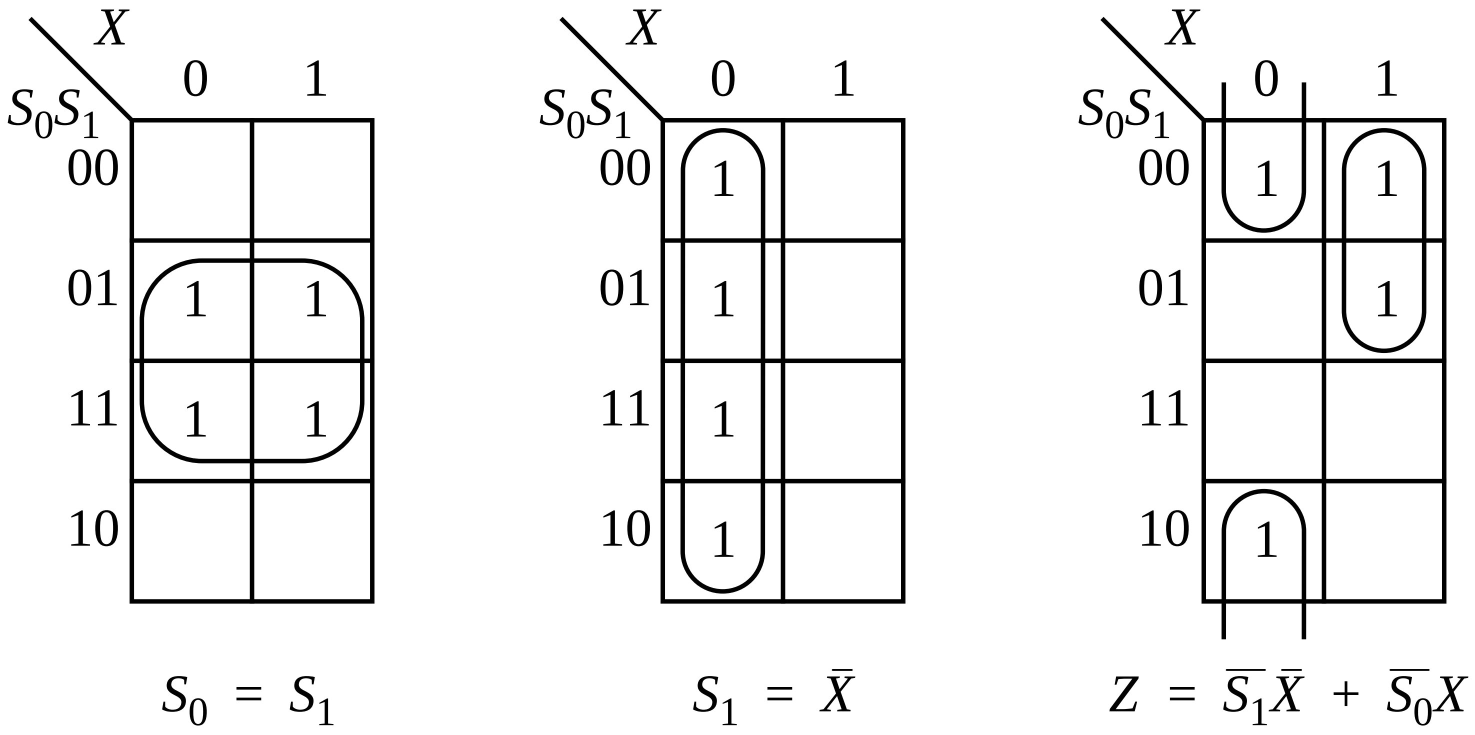
$$S_0 = \overline{S_0}S_1 + S_0\overline{S_1}$$

$$S_1 = \overline{S_0}\,\overline{S_1}\,\overline{X} + \overline{S_0}S_1X + S_0S_1\overline{X} + S_0\overline{S_1}X$$

$$Z = \overline{S_0}\,\overline{S_1} + \overline{S_0}X + S_0S_1\overline{X}$$

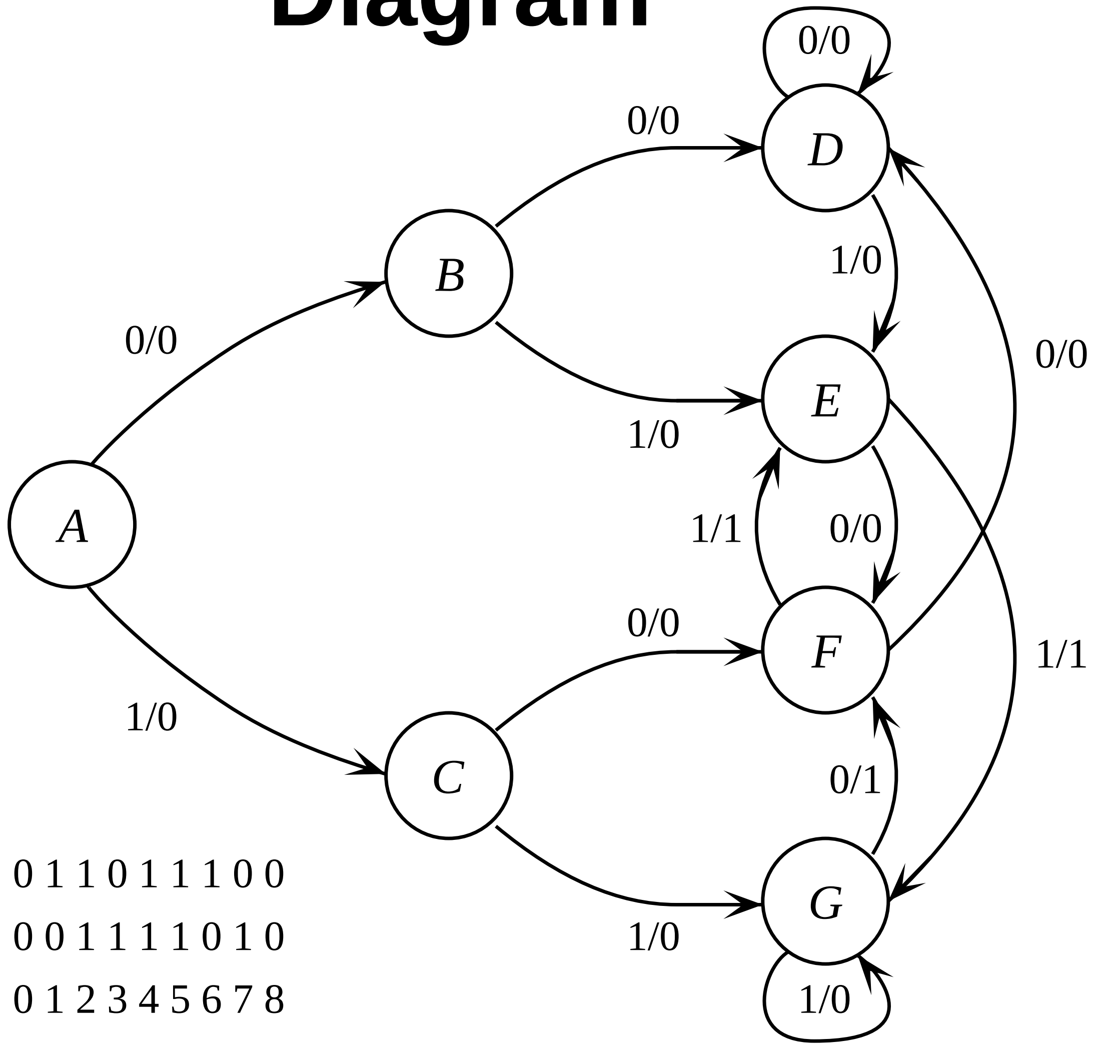
State Assignment SA$_1$

- **Boolean equations for machine *M*$_2$ using state assignment SA$_1$.**



$S_0 = S_1$ $\quad$ $S_1 = \bar{X}$ $\quad$ $Z = \overline{S_1}\bar{X} + \overline{S_0}X$

Sequence Detector State Transition Diagram



Input:	0 1 1 0 1 1 1 0 0
Output:	0 0 1 1 1 1 0 1 0
Time:	0 1 2 3 4 5 6 7 8

Sequence Detector State Table

Input / Present state	X = 0	X = 1
A	B/0	C/0
B	D/0	E/0
C	F/0	G/0
D	D/0	E/0
E	F/0	G/1
F	D/0	E/1
G	F/1	G/0

Sequence Detector Reduced State Table

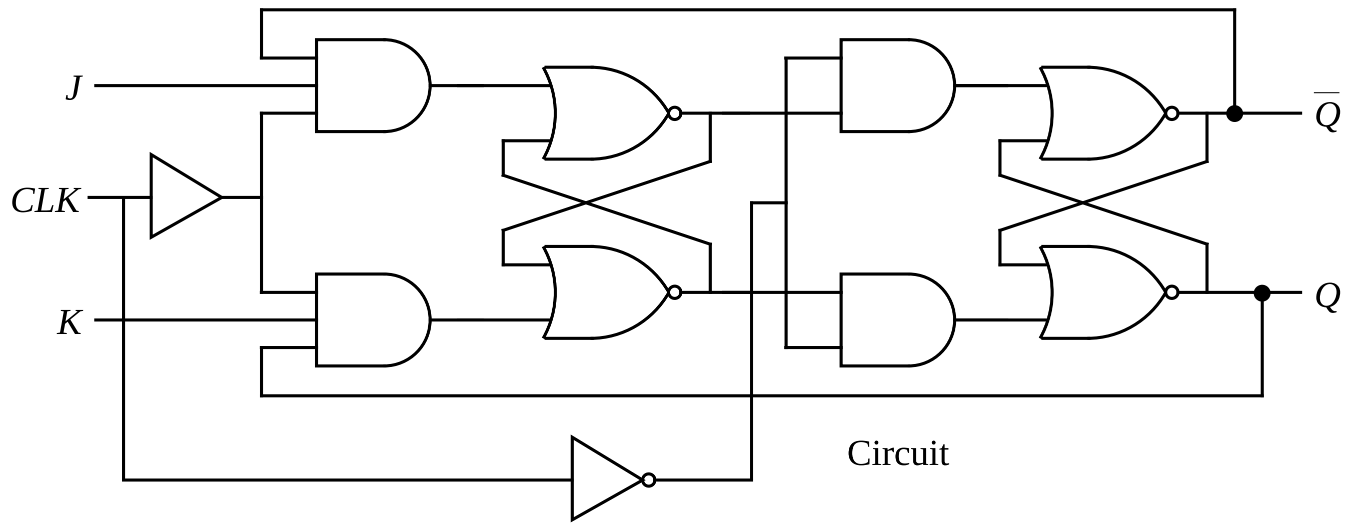
Present state \ Input	X = 0	X = 1
A: A'	B'/0	C'/0
BD: B'	B'/0	D'/0
C: C'	E'/0	F'/0
E: D'	E'/0	F'/1
F: E'	B'/0	D'/1
G: F'	E'/1	F'/0

Sequence Detector State Assignment

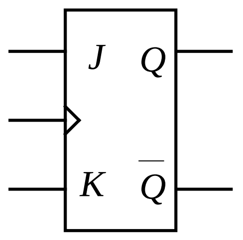
Present state \ Input	X = 0	X = 1
$S_2S_1S_0$	$S_2S_1S_0Z$	$S_2S_1S_0Z$
A': 000	001/0	010/0
B': 001	001/0	011/0
C': 010	100/0	101/0
D': 011	100/0	101/1
E': 100	001/0	011/1
F': 101	100/1	101/0

Excitation Tables

- In addition to the D flip-flop, the S-R, J-K, and T flip-flops are used as delay elements in finite state machines.
- A Master-Slave J-K flip-flop is shown below.



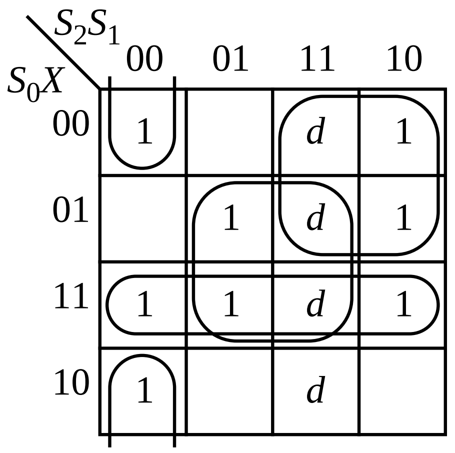
Circuit



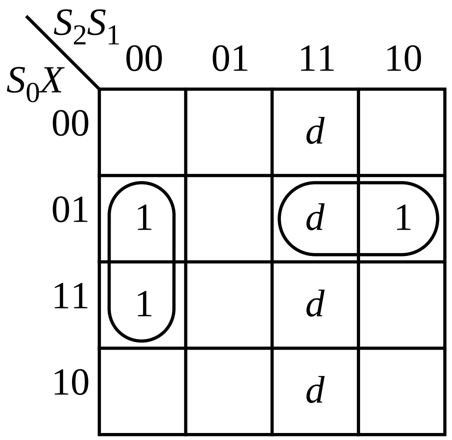
Symbol

Sequence Detector K-Maps

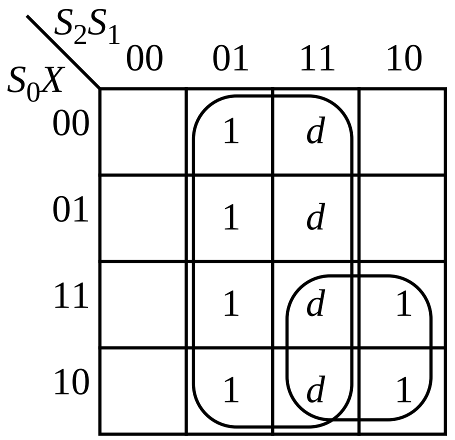
- **K-map reduction of next state and output functions for sequence detector.**



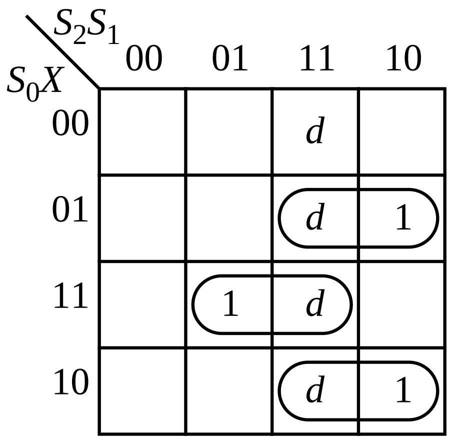
$$S_0 = \overline{S_2}\,\overline{S_1}\,\overline{X} + S_0X + S_2\overline{S_0} + S_1X$$



$$S_1 = \overline{S_2}\,\overline{S_1}X + S_2\overline{S_0}X$$



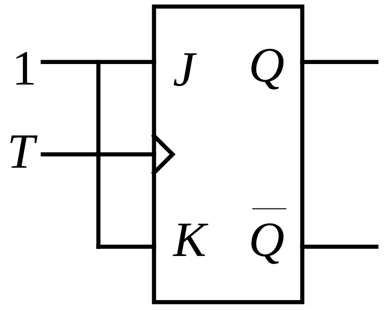
$$S_2 = S_2S_0 + S_1$$



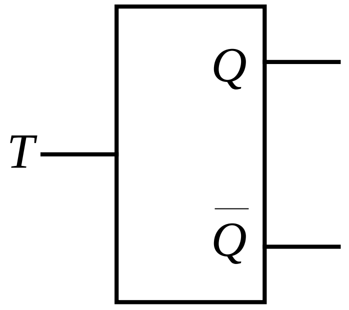
$$Z = S_2\overline{S_0}X + S_1S_0X + S_2S_0\overline{X}$$

Clocked T Flip-Flop

- **Logic diagram and symbol for a T flip-flop.**

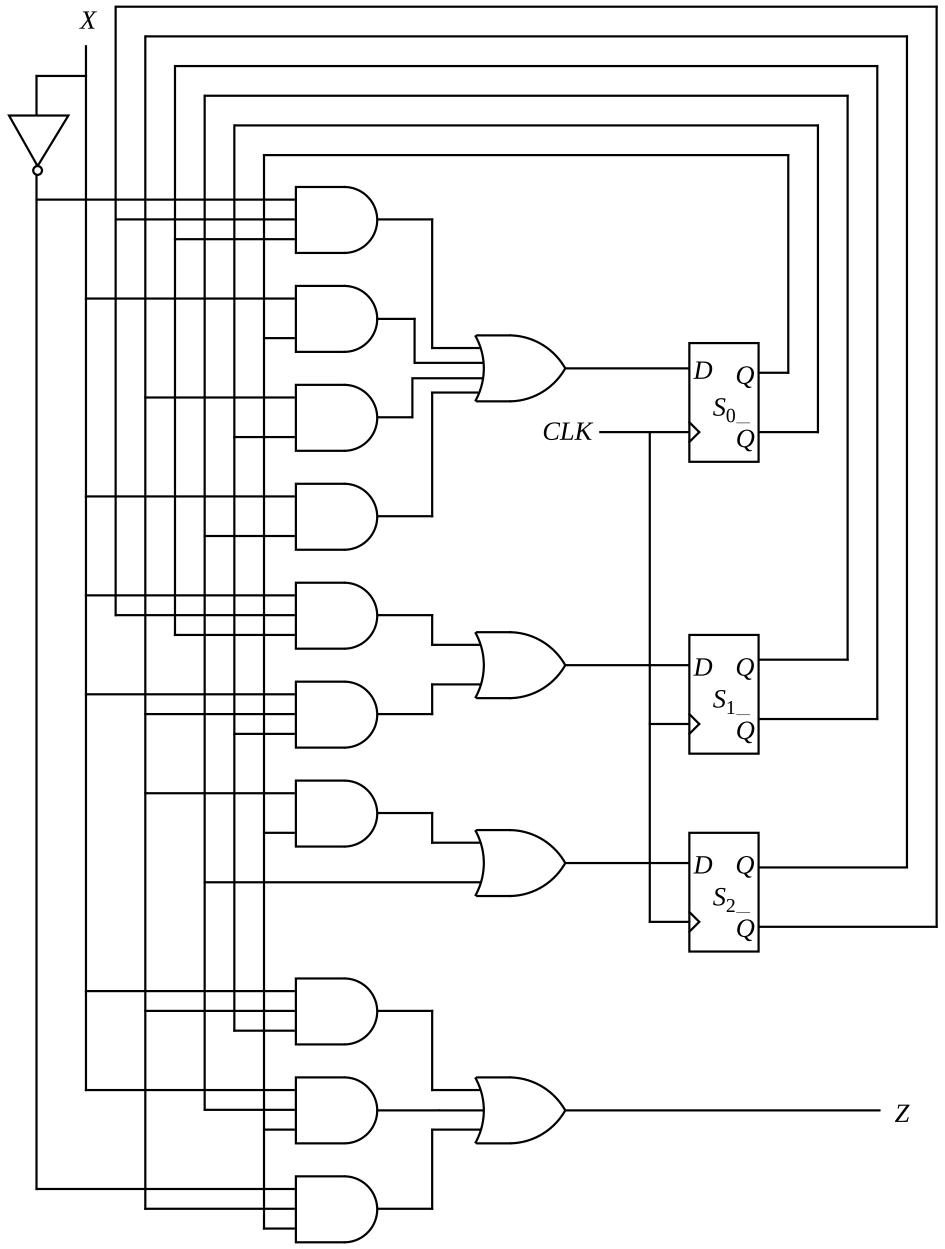


Circuit



Symbol

Sequence Detector Circuit



Excitation Tables

- **Each table shows the settings that must be applied at the inputs at time *t* in order to change the outputs at time *t*+1.**

S-R flip-flop

Q_t	Q_{t+1}	S	R
0	0	0	0
0	1	1	0
1	0	0	1
1	1	0	0

D flip-flop

Q_t	Q_{t+1}	D
0	0	0
0	1	1
1	0	0
1	1	1

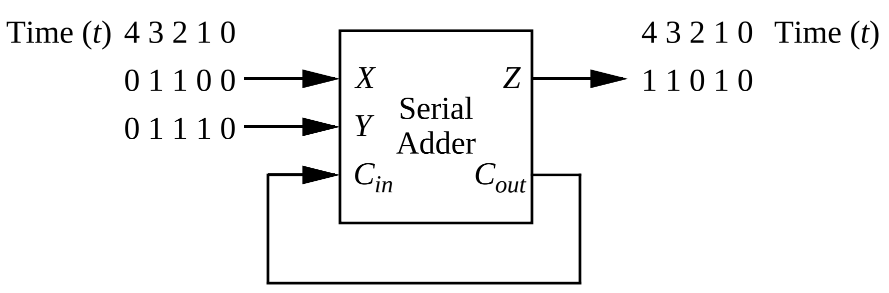
J-K flip-flop

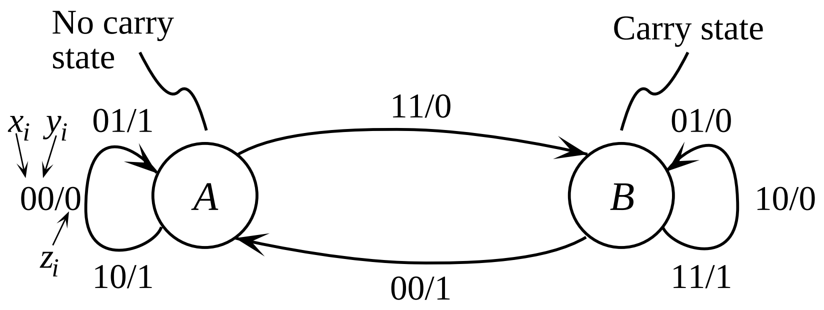
Q_t	Q_{t+1}	J	K
0	0	0	*d*
0	1	1	*d*
1	0	*d*	1
1	1	*d*	0

T flip-flop

Q_t	Q_{t+1}	T
0	0	0
0	1	1
1	0	1
1	1	0

Serial Adder





- **State transition diagram, state table, and state assignment for a serial adder.**

Present state \ Input	XY 00	01	10	11
A	A/0	A/1	A/1	B/0
B	A/1	B/0	B/0	B/1

Next state   Output

Present state (S_t) \ Input	XY 00	01	10	11
A:0	0/0	0/1	0/1	1/0
B:1	0/1	1/0	1/0	1/1

Serial Adder Next-State Functions

- **Truth table showing next-state functions for a serial adder for D, S-R, T, and J-K flip-flops. Shaded functions are used in the example.**

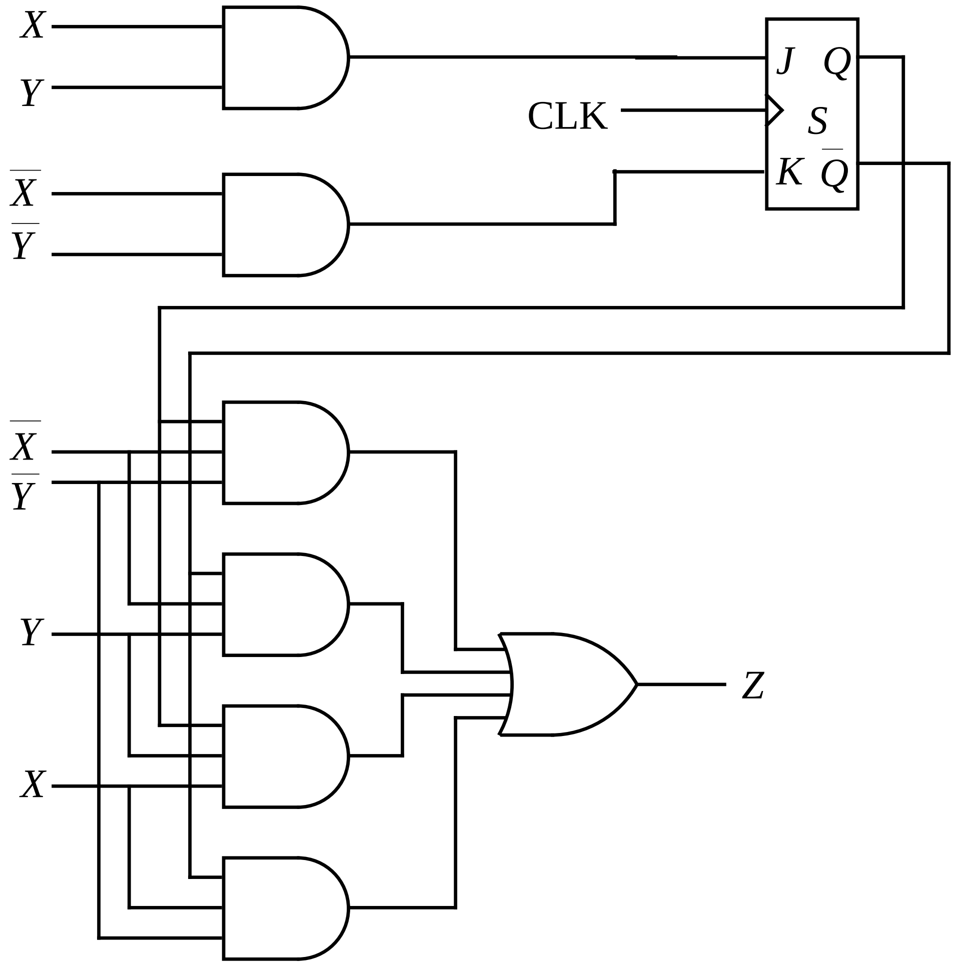
		Present State		(Set)	(Reset)				
X	Y	S_t	D	S	R	T	J	K	Z
0	0	0	0	0	0	0	0	d	0
0	0	1	0	0	1	1	d	1	1
0	1	0	0	0	0	0	0	d	1
0	1	1	1	0	0	0	d	0	0
1	0	0	0	0	0	0	0	d	1
1	0	1	1	0	0	0	d	0	0
1	1	0	1	1	0	1	1	d	0
1	1	1	1	0	0	0	d	0	1

J-K Flip-Flop Serial Adder Circuit

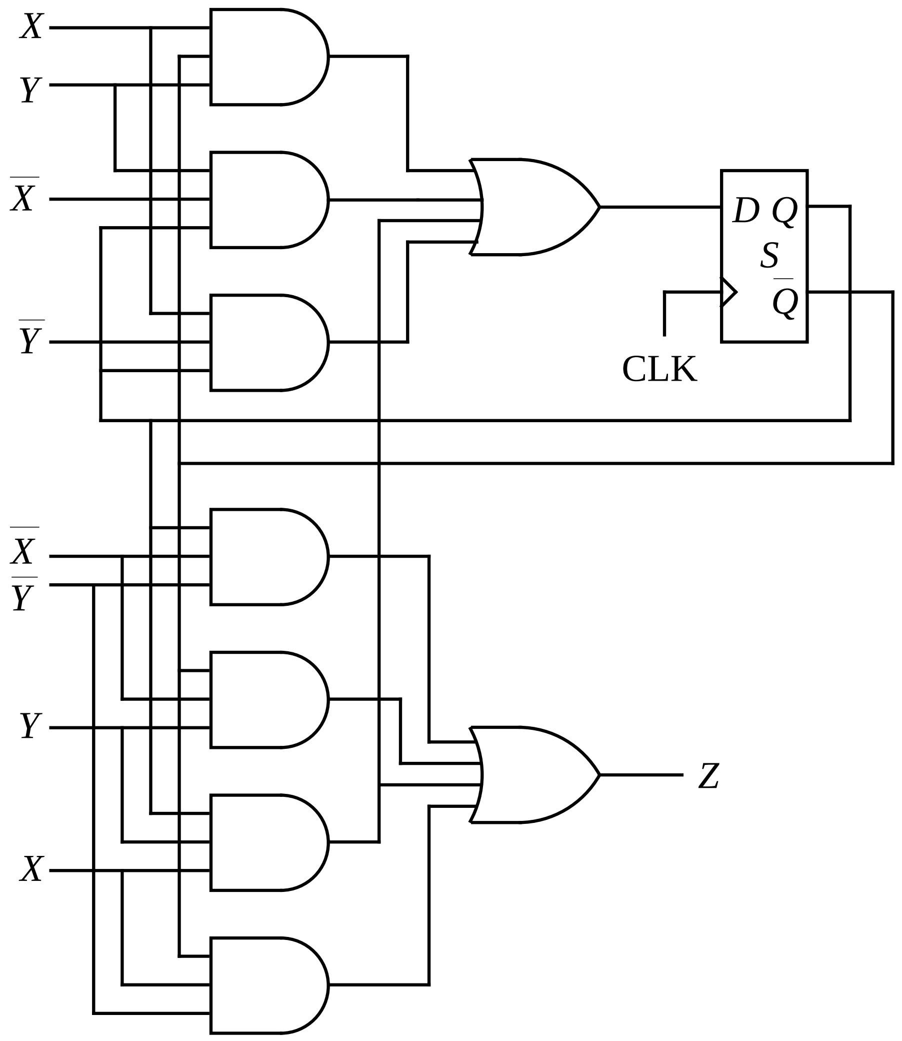
$$J = XY$$

$$K = \overline{X}\,\overline{Y}$$

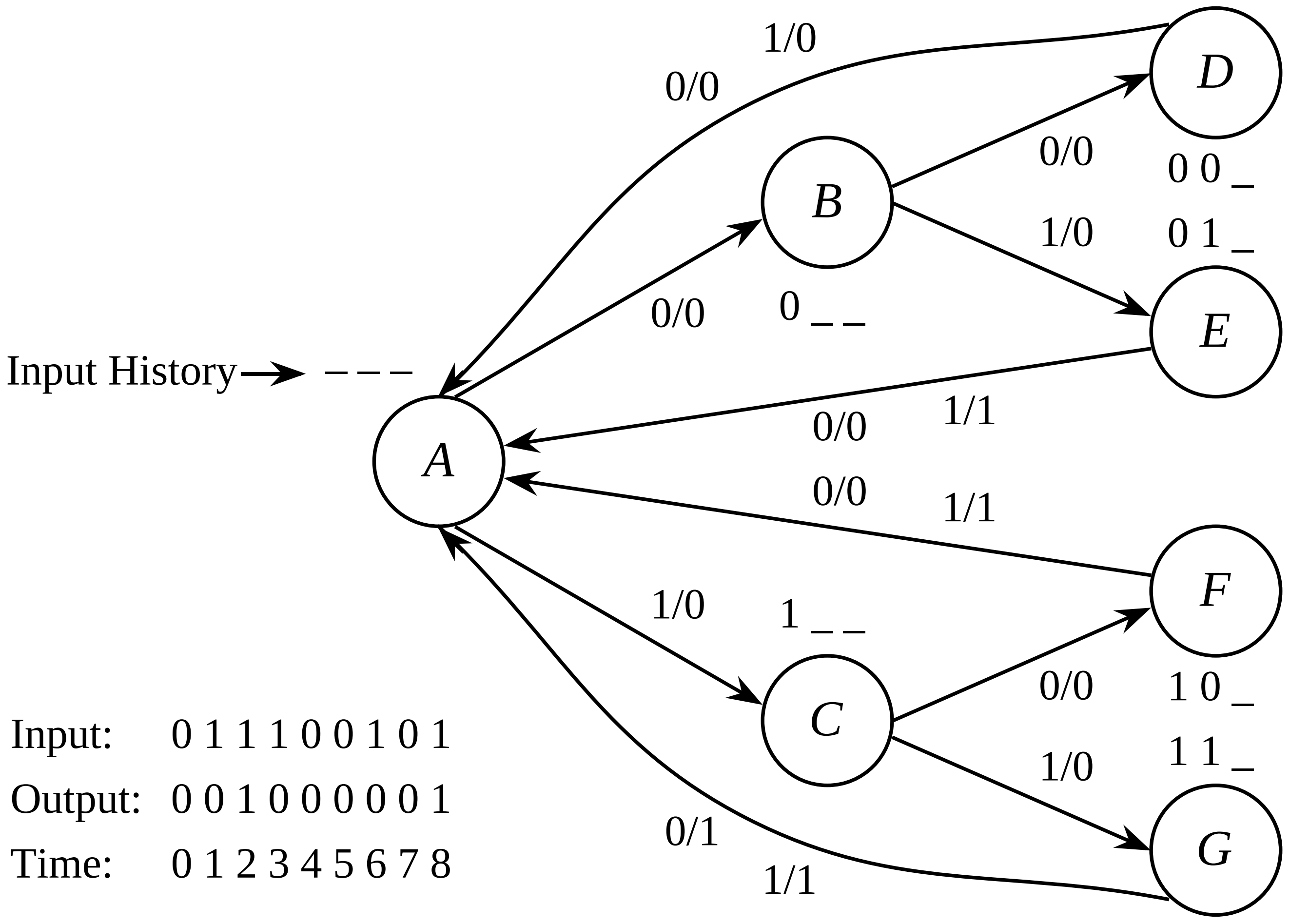
$$Z = \overline{X}\,\overline{Y}S + \overline{X}Y\overline{S} + XYS + X\overline{Y}\,\overline{S}$$



D Flip-Flop Serial Adder Circuit



Majority Finite State Machine



Input:	0 1 1 1 0 0 1 0 1
Output:	0 0 1 0 0 0 0 0 1
Time:	0 1 2 3 4 5 6 7 8

Majority FSM State Table

- **(a) State table for majority FSM; (b) partitioning; (c) reduced state table.**

P.S. \ Input	X = 0	X = 1
A	$B/0$	$C/0$
B	$D/0$	$E/0$
C	$F/0$	$G/0$
D	$A/0$	$A/0$
E	$A/0$	$A/1$
F	$A/0$	$A/1$
G	$A/1$	$A/1$

(a)

$$P_0 = (ABCDEFG)$$
$$P_1 = (ABCD)(EF)(G)$$
$$P_2 = (AD)(B)(C)(EF)(G)$$
$$P_3 = (A)(B)(C)(D)(EF)(G)$$
$$P_4 = (A)(B)(C)(D)(EF)(G) \checkmark$$

(b)

P.S. \ Input	X = 0	X = 1
$A: A'$	$B'/0$	$C'/0$
$B: B'$	$D'/0$	$E'/0$
$C: C'$	$E'/0$	$F'/0$
$D: D'$	$A'/0$	$A'/0$
$EF: E'$	$A'/0$	$A'/1$
$G: F'$	$A'/1$	$A'/1$

(c)

Majority FSM State Assignment

- **(a) State assignment for reduced majority FSM using D flip-flops; and (b) using T flip-flops.**

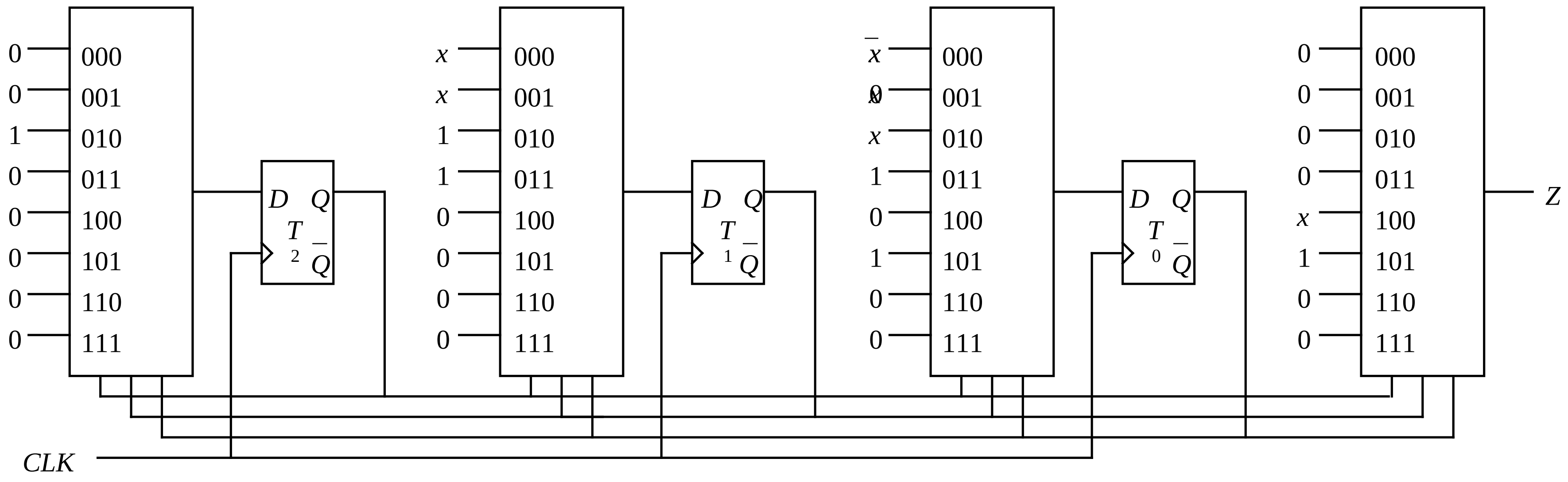
P.S. \ Input	X = 0	X = 1
$S_2S_1S_0$	$S_2S_1S_0Z$	$S_2S_1S_0Z$
A': 000	001/0	010/0
B': 001	011/0	100/0
C': 010	100/0	101/0
D': 011	000/0	000/0
E': 100	000/0	000/1
F': 101	000/1	000/1

(a)

P.S. \ Input	X = 0	X = 1
$S_2S_1S_0$	$T_2T_1T_0Z$	$T_2T_1T_0Z$
A': 000	001/0	010/0
B': 001	000/0	010/0
C': 010	110/0	111/0
D': 011	011/0	011/0
E': 100	100/0	100/1
F': 101	101/1	101/1

(b)

Majority FSM Circuit



0
0
1
0
0
0
0
0
000
001
010
011
100
101
110
111
D
Q
T
2
Q
x
x
1
1
0
0
0
0
D
Q
T
1
Q
x
0
x
1
0
1
0
0
D
Q
T
0
Q
0
0
0
0
x
1
0
0
Z
CLK