Logical Design

Zvi Kohavi and Niraj K. Jha

Design with Basic Logic Gates

Logic gates: perform logical operations on input signals

Positive (negative) logic polarity: constant 1 (0) denotes a high voltage and constant 0 a low (high) voltage

Synchronous circuits: driven by a clock that produces a train of equally spaced pulses

Asynchronous circuits: are almost free-running and do not depend on a clock; controlled by initiation and completion signals

Fanout: number of gate inputs driven by the output of a single gate

Fanin: bound on the number of inputs a gate can have

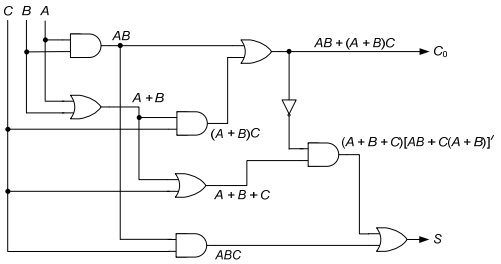
Propagation delay: time to propagate a signal through a gate

Analysis of Combinational Circuits

Circuit analysis: determine the Boolean function that describes the circuit

- Done by tracing the output of each gate, starting from circuit inputs and continuing towards each circuit output

Example: a multi-level realization of a full binary adder



$C_0 = AB + (A + B)C$
$= AB + AC + BC$

$S = (A + B + C)[AB + (A + B)C]' + ABC$
$= (A + B + C)(A' + B')(A' + C')(B' + C') + ABC$
$= AB'C' + A'BC' + A'B'C + ABC$
$= A \oplus B \oplus C$

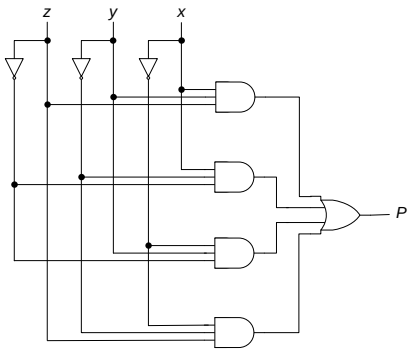
3

Simple Design Problems

Parallel parity-bit generator: produces output value 1 if and only if an odd number of its inputs have value 1

z \ xy	00	01	11	10
0	0	1	0	1
1	1	0	1	0

(*a*) Map.



(*b*) Implementation.

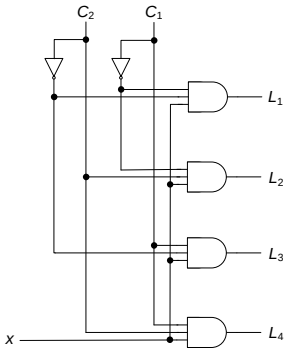
$P = x'y'z + x'yz' + xy'z' + xyz$

4

Simple Design Problems (Contd.)

Serial-to-parallel converter: distributes a sequence of binary digits on a serial input to a set of different outputs, as specified by external control signals

Control		*Output lines*				*Logic equations*
C_1	C_2	L_1	L_2	L_3	L_4	
0	0	x	0	0	0	$L_1 = xC_1'C_2'$
0	1	0	x	0	0	$L_2 = xC_1'C_2$
1	0	0	0	x	0	$L_3 = xC_1C_2'$
1	1	0	0	0	x	$L_4 = xC_1C_2$



5

Logic Design with Integrated Circuits

Small scale integration (SSI): integrated circuit packages containing a few gates; e.g., AND, OR, NOT, NAND, NOR, XOR

Medium scale integration (MSI): packages containing up to about 100 gates; e.g., code converters, adders

Large scale integration (LSI): packages containing thousands of gates; arithmetic unit

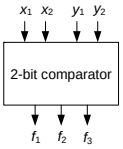
Very large scale integration (VLSI): packages with millions of gates

6

Comparators

n-bit comparator: compares the magnitude of two numbers *X* and *Y*, and has three outputs f_1, f_2, and f_3

- $f_1 = 1$ iff $X > Y$
- $f_2 = 1$ iff $X = Y$
- $f_3 = 1$ iff $X < Y$

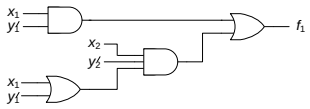


(*a*) Block diagram.

y_1y_2 \ x_1x_2	00	01	11	10
00	2	1	1	1
01	3	2	1	1
11	3	3	2	3
10	3	3	1	2

(*b*) Map for f_1, f_2, and f_3.

$$f_1 = x_1x_2y_2' + x_2y_1'y_2' + x_1y_1'$$
$$= (x_1 + y_1')x_2y_2' + x_1y_1'$$
$$f_2 = x_1'x_2'y_1'y_2' + x_1'x_2y_1'y_2 + x_1x_2'y_1y_2' + x_1x_2y_1y_2$$
$$= x_1'y_1'(x_2'y_2' + x_2y_2) + x_1y_1(x_2'y_2' + x_2y_2)$$
$$= (x_1'y_1' + x_1y_1)(x_2'y_2' + x_2y_2)$$
$$f_3 = x_2'y_1y_2 + x_1'x_2'y_2 + x_1'y_1$$
$$= x_2'y_2(y_1 + x_1') + x_1'y_1$$

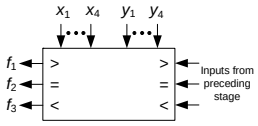


(*c*) Circuit for f_1.

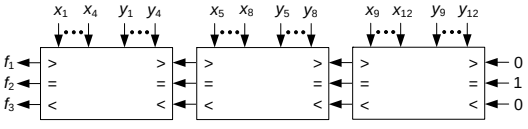
4-bit/12-bit Comparators

Four-bit comparator: 11 inputs (four for *X*, four for *Y*, and three connected to outputs f_1, f_2 and f_3 of the preceding stage)

12-bit comparator:



(*a*) A 4-bit comparator.



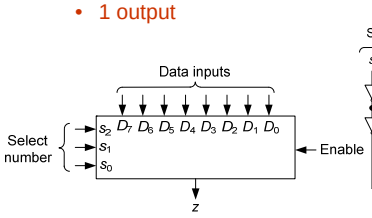
(*b*) A 12-bit comparator.

Data Selectors

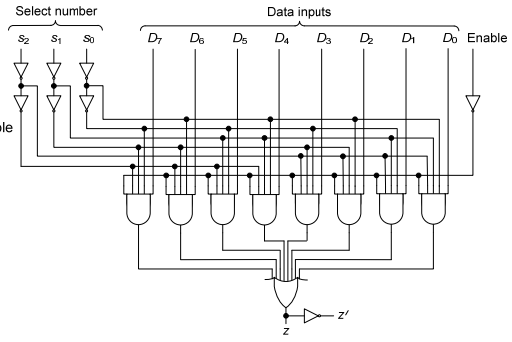
Multiplexer: electronic switch that connects one of *n* inputs to the output

Data selector: application of multiplexer

- *n* data input lines, D_0, D_1, …, D_{n-1}
- *m* select digit inputs s_0, s_1, …, s_{m-1}
- 1 output



(*a*) Block diagram.

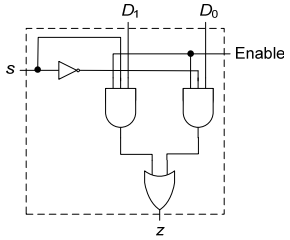


(*b*) Logic diagram.

Implementing Switching Functions with Data Selectors

Data selectors: can implement arbitrary switching functions

Example: implementing two-variable functions



$z = sD_1 + s'D_0$

If $s = A$, $B = D_0$, and $B' = D_1$, then $z = A \oplus B$.

If $s = A$, $D_0 = 1$, and $D_1 = B'$, then $z = A' + B'$.

Implementing Switching Functions with Data Selectors (Contd.)

To implement an n-variable function: a data selector with n-1 select inputs and 2^{n-1} data inputs

Implementing three-variable functions:

$$z = s_2's_1'D_0 + s_2's_1D_1 + s_2s_1'D_2 + s_2s_1D_3$$

Example: $s_1 = A$, $s_2 = B$, $D_0 = C$, $D_1 = 1$, $D_2 = 0$, $D_3 = C'$

$$z = A'B'C + AB' + ABC'$$
$$= AC' + B'C$$

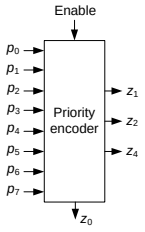
General case: Assign n-1 variables to the select inputs and last variable and constants 0 and 1 to the data inputs such that desired function results

Priority Encoders

Priority encoder: n input lines and $\log_2 n$ output lines

- Input lines represent units that may request service
- When inputs p_i and p_j, such that $i > j$, request service simultaneously, line p_i has priority over line p_j
- Encoder produces a binary output code indicating which of the input lines requesting service has the highest priority

Example: Eight-input, three-output priority encoder



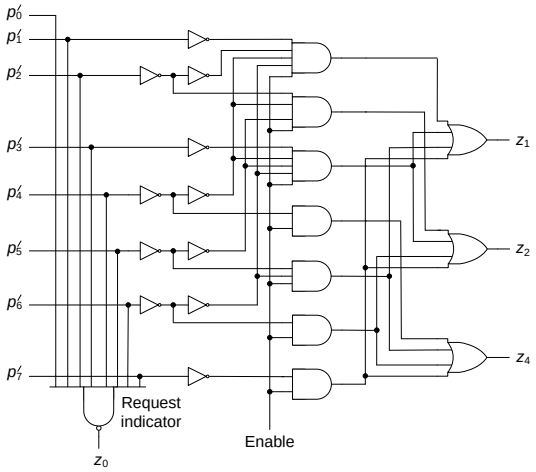
(*a*) Block diagram.

Input lines							Outputs		
p_0	p_1	p_2	p_3	p_4	p_5	p_6	p_7	z_4	z_2	z_1
1	0	0	0	0	0	0	0	0	0	0
ϕ	1	0	0	0	0	0	0	0	0	1
ϕ	ϕ	1	0	0	0	0	0	0	1	0
ϕ	ϕ	ϕ	1	0	0	0	0	0	1	1
ϕ	ϕ	ϕ	ϕ	1	0	0	0	1	0	0
ϕ	ϕ	ϕ	ϕ	ϕ	1	0	0	1	0	1
ϕ	ϕ	ϕ	ϕ	ϕ	ϕ	1	0	1	1	0
ϕ	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ	1	1	1	1

(*b*) Truth table.

$$z_4 = p_4p_5'p_6'p_7' + p_5p_6'p_7' + p_6p_7' + p_7 = p_4 + p_5 + p_6 + p_7$$
$$z_2 = p_2p_3'p_4'p_5'p_6'p_7' + p_3p_4'p_5'p_6'p_7' + p_6p_7' + p_7 = p_2p_4'p_5' + p_3p_4'p_5' + p_6 + p_7$$
$$z_1 = p_1p_2'p_3'p_4'p_5'p_6'p_7' + p_3p_4'p_5'p_6'p_7' + p_5p_6'p_7' + p_7 = p_1p_2'p_4'p_6' + p_3p_4'p_6' + p_5p_6' + p_7$$

Priority Encoders (Contd.)



(*c*) Logic diagram.

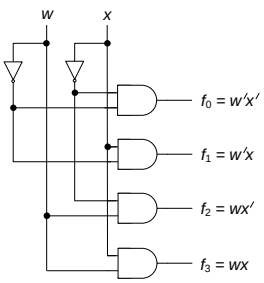
Decoders

Decoders with n inputs and 2^n outputs: for any input combination, only one output is 1

Useful for:

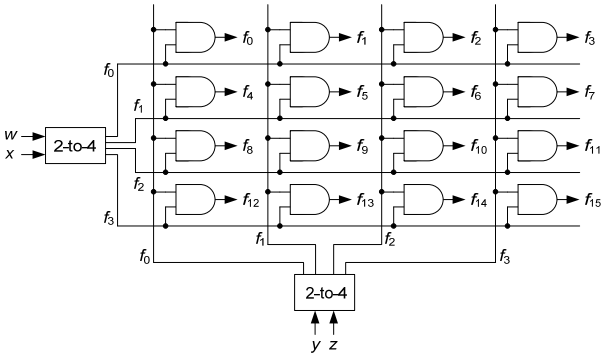
- Routing input data to a specified output line, e.g., in addressing memory
- Basic building blocks for implementing arbitrary switching functions
- Code conversion
- Data distribution

Example: 2-to-4- decoder



Decoders (Contd.)

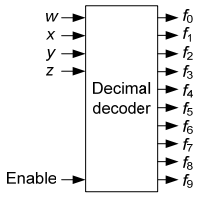
Example: 4-to-16 decoder made of two 2-to-4 decoders and a gate-switching matrix



15

Decimal Decoder

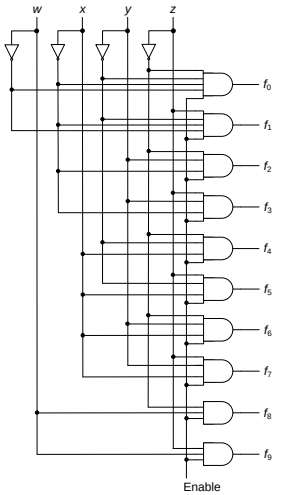
BCD-to-decimal: 4-to-16 decoder made of two 2-to-4 decoders and a gate-switching matrix



(*a*) Block diagram.

yz \ wx	00	01	11	10
00	f_0	f_4	ϕ	f_8
01	f_1	f_5	ϕ	f_9
11	f_3	f_7	ϕ	ϕ
10	f_2	f_6	ϕ	ϕ

(*b*) Map.

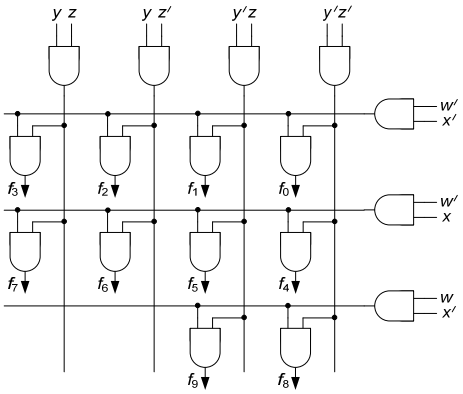


(*c*) Logic diagram.

16

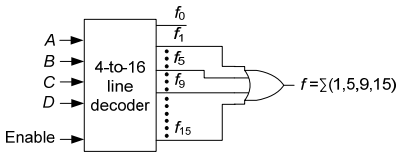
Decimal Decoder (Contd.)

Implementation using a partial-gate matrix:



Implementing Arbitrary Switching Functions

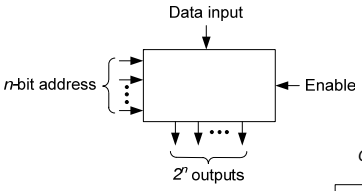
Example: Realize a distinct minterm at each output



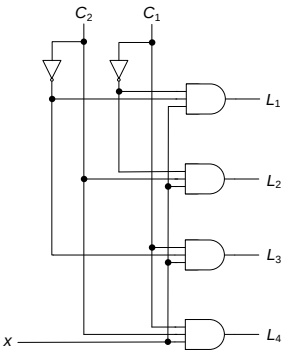
Demultiplexers

Demultiplexers: decoder with1 data input and *n* address inputs

- Directs input to any one of the 2^n outputs



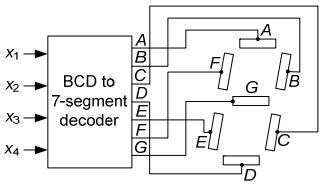
Example: A 4-output demultiplexer



19

Seven-segment Display

Seven-segment display: BCD to seven-segment decoder and seven LEDs



Seven-segment pattern and code:

Decimal digit	*BCD code*				*Seven-segment code*						
	x_1	x_2	x_3	x_4	A	B	C	D	E	F	G
1	0	0	0	1	0	1	1	0	0	0	0
2	0	0	1	0	1	1	0	1	1	0	1
3	0	0	1	1	1	1	1	1	0	0	1
4	0	1	0	0	0	1	1	0	0	1	1
5	0	1	0	1	1	0	1	1	0	1	1
6	0	1	1	0	0	0	1	1	1	1	1
7	0	1	1	1	1	1	1	0	0	0	0
8	1	0	0	0	1	1	1	1	1	1	1
9	1	0	0	1	1	1	1	0	0	1	1
0	0	0	0	0	1	1	1	1	1	1	0

$A = x_1 + x_2'x_4' + x_2x_4 + x_3x_4$

$B = x_2' + x_3'x_4' + x_3x_4$

$C = x_2 + x_3' + x_4$

$D = x_2'x_4' + x_2'x_3 + x_3x_4' + x_2x_3'x_4$

$E = x_2'x_4' + x_3x_4'$

$F = x_1 + x_2x_3' + x_2x_4' + x_3'x_4'$

$G = x_1 + x_2'x_3 + x_2x_3' + x_3x_4'$

20

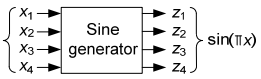
Sine Generators

Combinational sine generators: for fast and repeated evaluation of sine

- Input: angle in radians converted to binary
- Output: sine in binary

Angle x				$\sin(\pi x)$			
x_1	x_2	x_3	x_4	z_1	z_2	z_3	z_4
0	0	0	0	0	0	0	0
0	0	0	1	0	0	1	1
0	0	1	0	0	1	1	0
0	0	1	1	1	0	0	0
0	1	0	0	1	0	1	1
0	1	0	1	1	1	0	1
0	1	1	0	1	1	1	0
0	1	1	1	1	1	1	1
1	0	0	0	1	1	1	1
1	0	0	1	1	1	1	1
1	0	1	0	1	1	1	0
1	0	1	1	1	1	0	1
1	1	0	0	1	0	1	1
1	1	0	1	1	0	0	0
1	1	1	0	0	1	1	0
1	1	1	1	0	0	1	1

(*a*) Truth table.

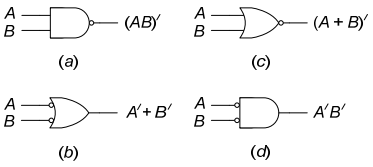


(*b*) Block diagram.

$$z_1 = x_1'x_2 + x_1x_2' + x_2x_3' + x_1'x_3x_4$$
$$z_2 = x_1x_2' + x_3x_4' + x_1'x_2x_4$$
$$z_3 = x_3x_4' + x_2x_3 + x_2x_4' + x_2'x_3'x_4 + x_1x_4'$$
$$z_4 = x_2'x_3'x_4 + x_2x_3'x_4' + x_1x_2'x_3' + x_1x_3x_4 + x_1'x_2x_4$$

NAND/NOR Circuits

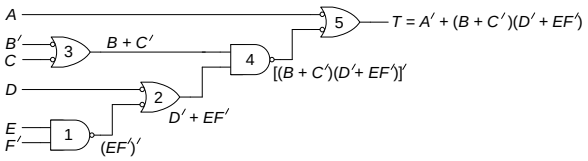
Switching algebra: not directly applicable to NAND/NOR logic



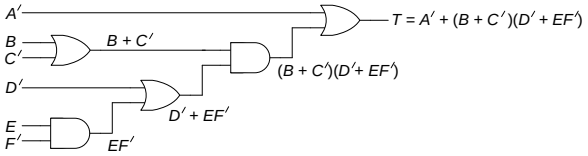
NAND and NOR gate symbols

Analysis of NAND/NOR Networks

Example: circles (inversions) at both ends of a line cancel each other



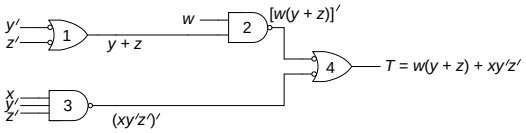
(*a*) NAND-logic circuit.



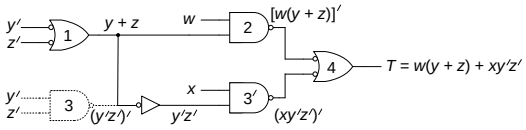
(*b*) Logically equivalent AND-OR circuit.

Synthesis of NAND/NOR Networks

Example: Realize $T = w(y+z) + xy'z'$



(*a*) First realization.



(*b*) Realization with two-input gates.

Design of High-speed Adders

Full adder: performs binary addition of three binary digits

- Inputs: arguments *A* and *B* and carry-in *C*
- Outputs: sum *S* and carry-out C_0

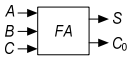
Example:

$$\begin{array}{rcl} 0\,1\,1 & = & \text{carry-in} \\ 1\,0\,1\,1 & = & \text{augend} \\ 0\,0\,1\,1 & = & \text{addend} \\ \hline 1\,1\,1\,0 & = & \text{sum} \end{array}$$

Truth table, block diagram and expressions:

A	B	C	S	C_0
0	0	0	0	0
0	0	1	1	0
0	1	1	0	1
0	1	0	1	0
1	1	0	0	1
1	1	1	1	1
1	0	1	0	1
1	0	0	1	0

(*a*) Truth table for *S* and C_0.



(*b*) Block diagram.

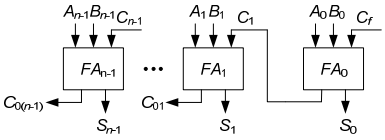
$S = A'B'C + A'BC' + AB'C' + ABC$
$= A \oplus B \oplus C$
$C_0 = A'BC + ABC' + AB'C + ABC$
$= AB + AC + BC$

25

Ripple-carry Adder

Ripple-carry adder: Stages of full adders

- C_f: forced carry
- $C_{0(n-1)}$: overflow carry



$S_i = A_i \oplus B_i \oplus C_i$
$C_{0i} = A_iB_i + A_iC_i + B_iC_i$

Time required:

- Time per full adder: 2 units
- Time for ripple-carry adder: 2*n* units

26

Carry-lookahead Adder

Carry-lookahead adder: several stages simultaneously examined and their carries generated in parallel

- Generate signal $D_i = A_iB_i$
- Propagate signal $T_i = A_i \oplus B_i$
- Thus, $C_{0i} = D_i + T_iC_i$

To generate carries in parallel: convert recursive form to nonrecursive

$$C_{0i} = D_i + T_iC_i$$
$$C_i = C_{0(i-1)}$$
$$C_{0i} = D_i + T_i(D_{i-1} + T_{i-1}C_{i-1})$$
$$= D_i + T_iD_{i-1} + T_iT_{i-1}(D_{i-2} + T_{i-2}C_{i-2})$$
$$= D_i + T_iD_{i-1} + T_iT_{i-1}D_{i-2} + T_iT_{i-1}T_{i-2}C_{i-2}$$
$$\ldots \quad \ldots\ldots\ldots$$
$$C_{0i} = D_i + T_iD_{i-1} + T_iT_{i-1}D_{i-2} + \ldots + T_iT_{i-1}T_{i-2}\ldots T_0C_f$$

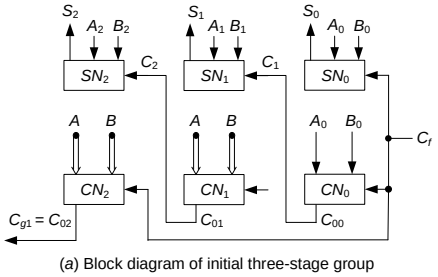
Thus, $C_{0i} = 1$ if it has been generated in the i^{th} stage or originated in a preceding stage and propagated to all subsequent stages

Carry-lookahead Adder (Contd.)

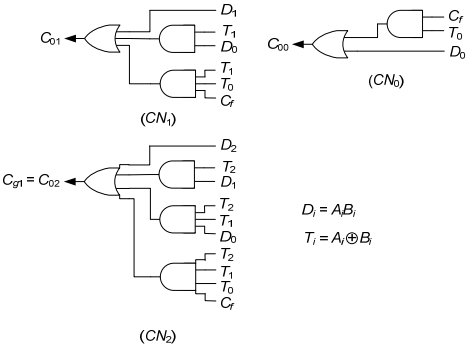
Implementation of lookahead for the complete adder impractical:

- Divide the *n* stages into groups
- Full carry lookahead within group
- Ripple carry between groups

Example: Three-digit adder group with full carry lookahead



(*a*) Block diagram of initial three-stage group



(*b*) The carry networks

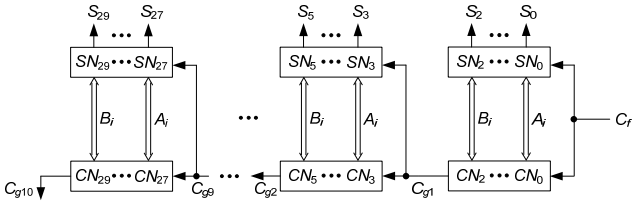
Time taken:

- 4 time units for C_{g1}
- Only 2 time units for C_{g2} and other group carries

30-bit Adder

Example: divide *n* stages into groups of three stages

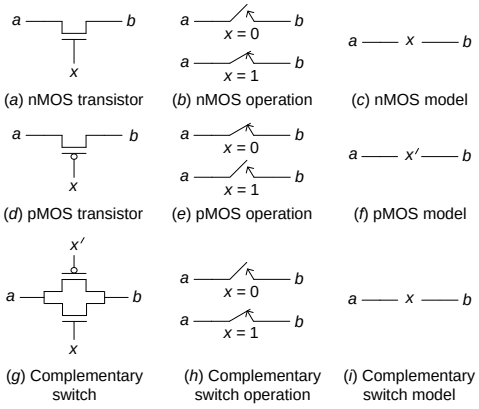
- Time taken: $4 + 2n/3$ time units
- 50% additional hardware for a threefold speedup



Metal-oxide Semiconductor (MOS) Transistors and Gates

Complementary metal-oxide semiconductor (CMOS): currently the dominant technology

- Two types of transistors: nMOS and pMOS

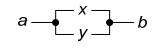
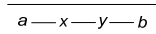
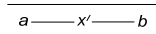


(*a*) nMOS transistor (*b*) nMOS operation (*c*) nMOS model

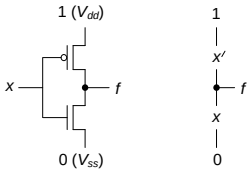
(*d*) pMOS transistor (*e*) pMOS operation (*f*) pMOS model

(*g*) Complementary switch (*h*) Complementary switch operation (*i*) Complementary switch model

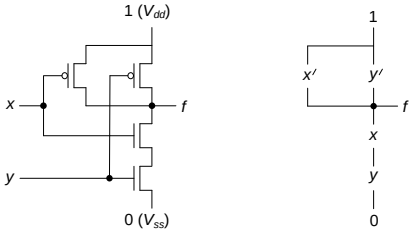
Transmission Function of a Network

Network	Transmission function
	$T_{ab} = x + y$
	$T_{ab} = xy$
	$T_{ab} = x'$

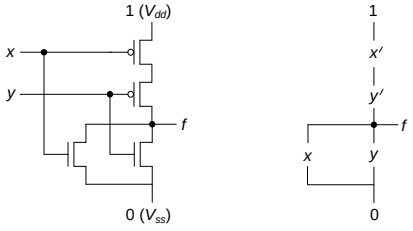
CMOS inverter and its transmission functions:



CMOS NAND/NOR Gates



(*a*) CMOS NAND gate and its transmission functions.

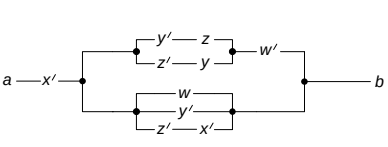


(*b*) CMOS NOR gate and its transmission functions.

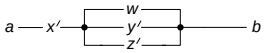
Analysis of Series-parallel Networks

Algebra of MOS networks: isomorphic to switching algebra

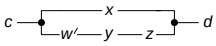
Example: Find the transmission function of the network and its complementary switch based and complex gate CMOS implementations



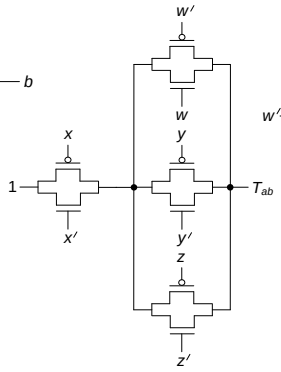
(*a*) $T_{ab} = x'[(y'z + z'y)w' + w + y' + x'z']$.



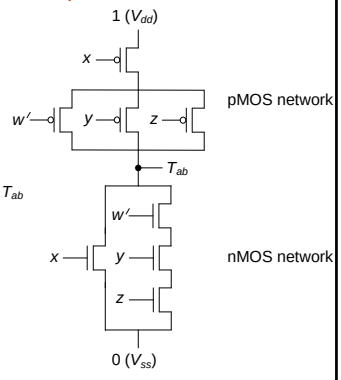
(*b*) $T_{ab} = x'(w + y' + z')$.



(*c*) $T_{cd} = T'_{ab} = x + w'yz$.



Complementary switch based



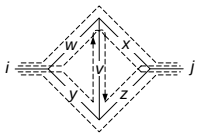
Complex gate

33

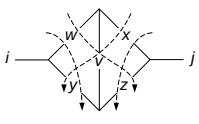
Analysis of Non-series-parallel Networks

Obtaining the transmission function:

- Tie sets: minimal paths between two terminals
- Cut sets: minimal sets of branches, when open, ensure no transmission between the two terminals



(*a*) Tie sets. $T_{ij} = wx + wvz + yvx + yz$.



(*b*) *Cut* sets. $T_{ij} = (w + y)(w + v + z)(x + v + y)(x + z)$.

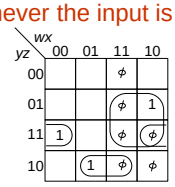
34

Synthesis of MOS Networks

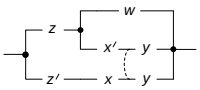
Sneak paths in non-series-parallel networks: undesired paths that may change the transmission function

- Occur because of bilateral nature of MOS transistors

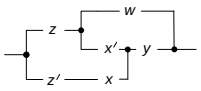
Example: Design a minimal network with BCD inputs that produces a 1 whenever the input is 3 or a multiple of 3



(*a*) Map for $T = wz + xyz' + x'yz$.



(*b*) Series-parallel realization of *T*.

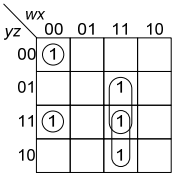


(*c*) Minimal realization of *T*.

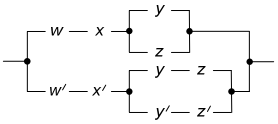
Sneak path: *z'xx'w* – OK since it has no effect on the transmission function

Synthesis of MOS Networks (Contd.)

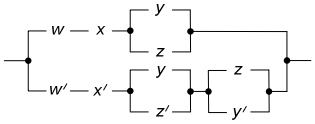
Example: Design a minimal network to realize $T(w,x,y,z) = \sum(0,3,13,14,15)$



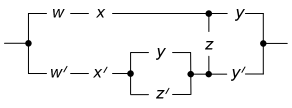
(*a*) Map for $T = wxy + wxz + w'x'y'z' + w'x'yz$.



(*b*) Series-parallel realization of *T*.



(*c*) An alternative series-parallel realization of *T*.



(*d*) A minimum realization of *T*.