CS 225 Switching Theory

Dr. Somanath Tripathy
Indian Institute of Technology Patna

Previous Class

Minimization/ Simplification **of Switching Functions**

- **K-map (SOP)**
- **Quine-McCluskey (Tabular) Minimization**

This Class

Combinational Circuit logic design

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

Logic Design with Integrated Circuits

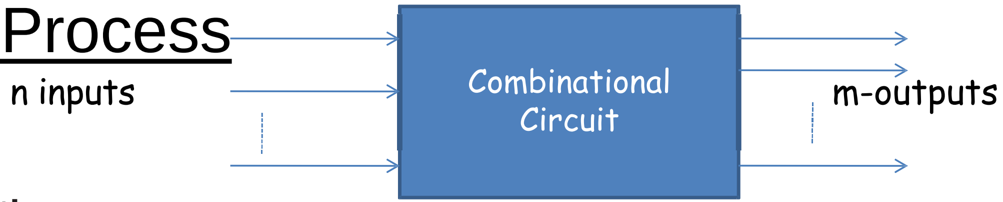
Small scale integration (SSI): integrated circuit packages containing a few gates; e.g., AND, OR, NOT, NAND, NOR, XOR

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

Combinational Logic Design Process



Step		Description
Step 1: Capture behavior	Capture the function	Create a truth table or equations, ***whichever is most natural for the given problem***, to describe the desired behavior of each output of the combinational logic.
Step 2: Convert to circuit	2A: Create equations	This substep is only necessary if you captured the function using a truth table instead of equations. Create an equation for each output by ORing all the minterms for that output. Simplify the equations if desired.
	2B: Implement as a gate-based circuit	For each output, create a circuit corresponding to the output's equation. (Sharing gates among multiple outputs is OK optionally.)

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

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

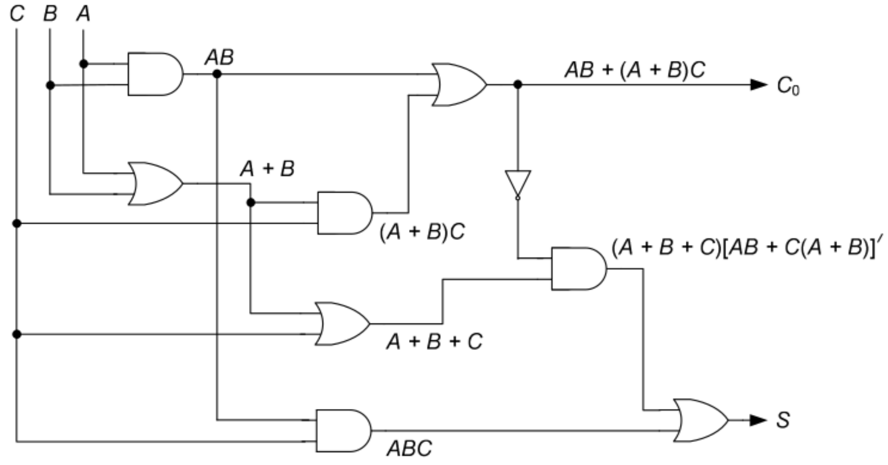
$$= AB + AC + BC$$

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

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

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

$$= A \oplus B \oplus C$$



Simple Design Problems

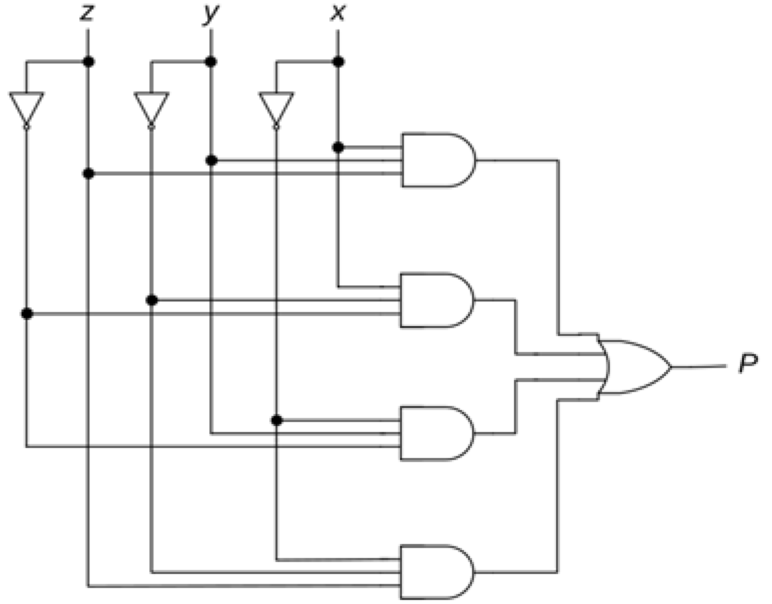
Parallel parity-bit generator: produces output value 1 if and only if an odd number of its inputs have value 1

K-Map

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

$$P = X'Y'Z + X'YZ' + XY'Z' + XYZ$$

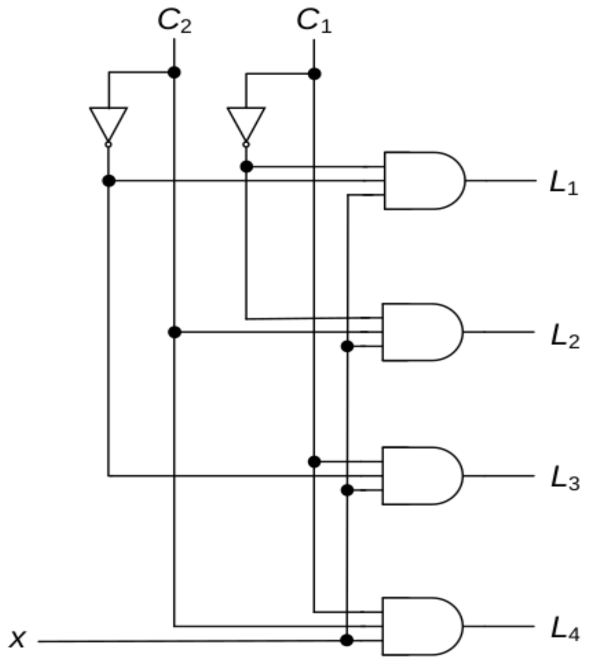
Implementation



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
C1	C2	L1	L2	L3	L4	
0	0	x	0	0	0	$L_1 = xC1'C2'$
0	1	0	x	0	0	$L_2 = xC1'C2$
1	0	0	0	x	0	$L_3 = xC1C2'$
1	1	0	0	0	x	$L_4 = xC1C2$



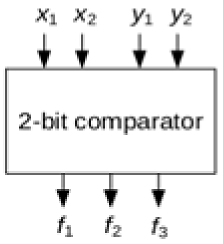
Comparators

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 $X < Y$

2-Bit Comparator

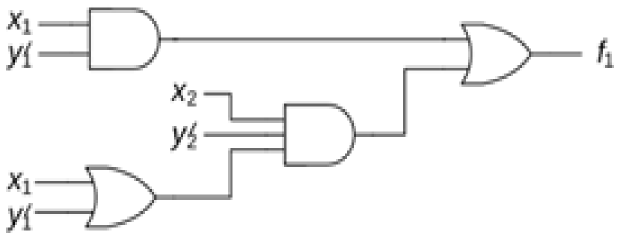
Block diagram



K-Map

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

Logic circuit diagram



Logic Expression

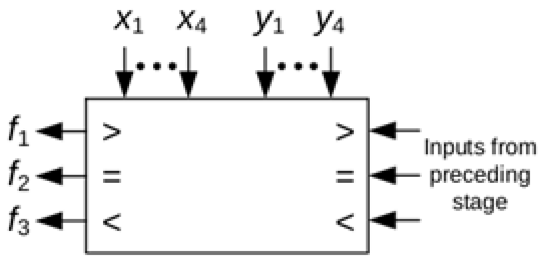
$$f_1 = X_1X_2Y_2' + X_2Y_1'Y_2' + X_1Y_1'$$
$$= (X_1 + Y_1')X_2Y_2' + X_1Y_1'$$

$$f_2 = X_1'X_2'Y_1'Y_2' + X_1'X_2Y_1'Y_2$$
$$+ X_1X_2'Y_1Y_2' + X_1X_2Y_1Y_2$$
$$= X_1'Y_1'(X_2'Y_2' + X_2Y_2)$$
$$+ X_1Y_1(X_2'Y_2' + X_2Y_2)$$
$$= (X_1'Y_1' + X_1Y_1)(X_2'Y_2' + X_2Y_2)$$

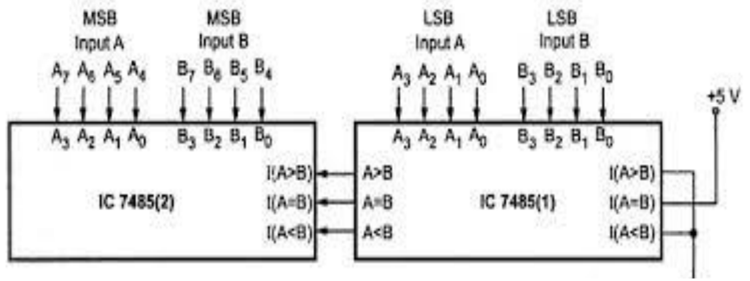
$$f_3 = X_2'Y_1Y_2 + X_1'X_2'Y_2 + X_1'Y_1$$
$$= X_2'Y_2(Y_1 + X_1') + X_1'Y_1$$

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)

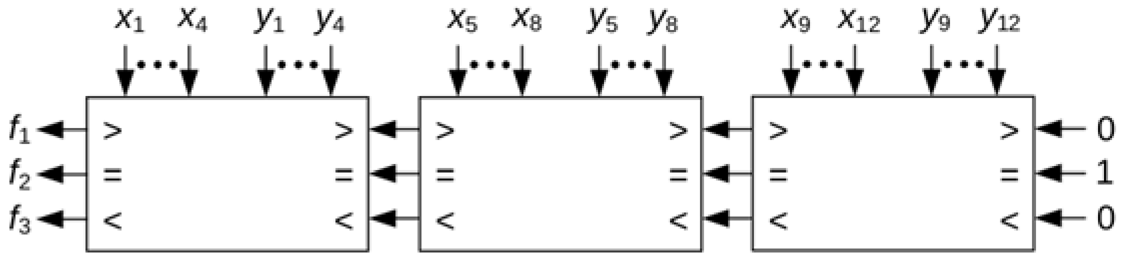


(a) A 4-bit comparator.



:

12-bit Comparator:



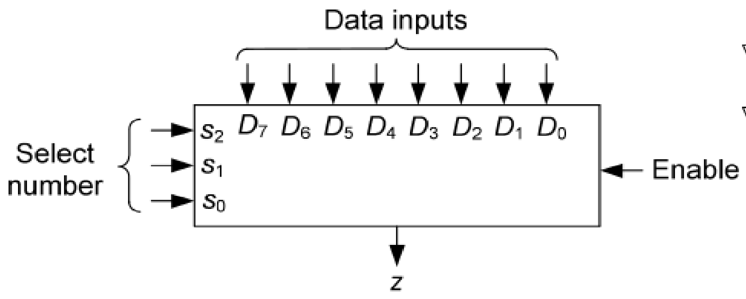
(b) A 12-bit comparator.

Data Selectors

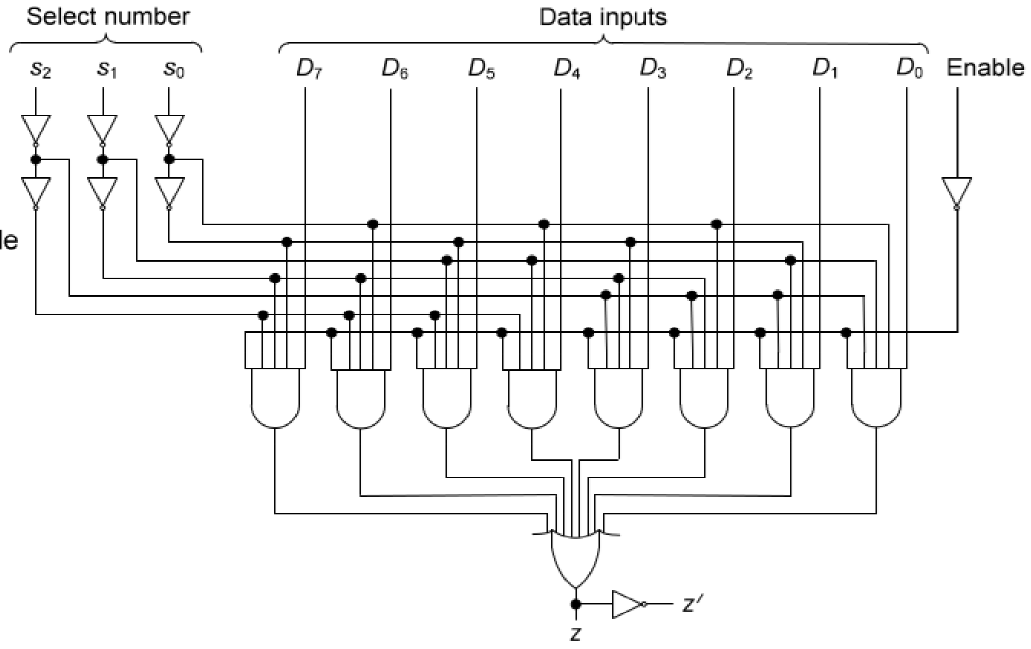
Multiplexer: electronic switch that connects one of n inputs to the output

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

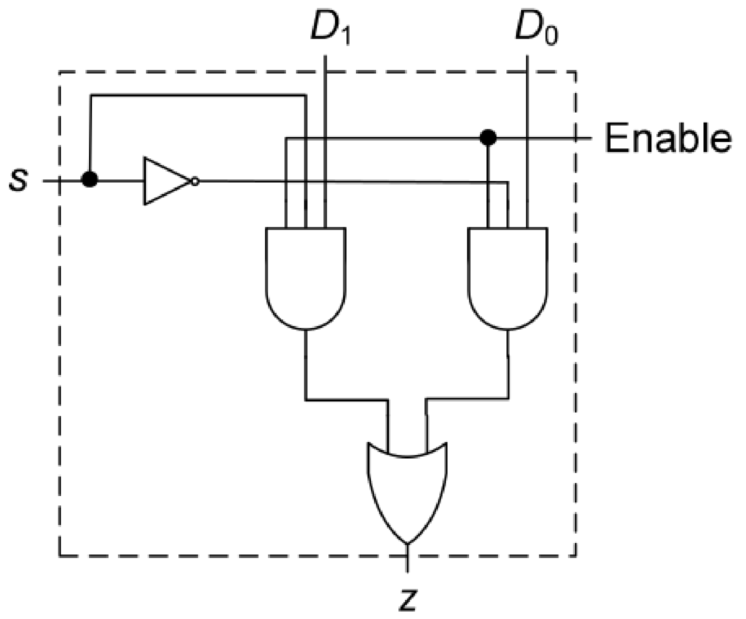
$z = sD_1 + s'D_0$

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

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

As $AB' + A' = A' + B'$

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

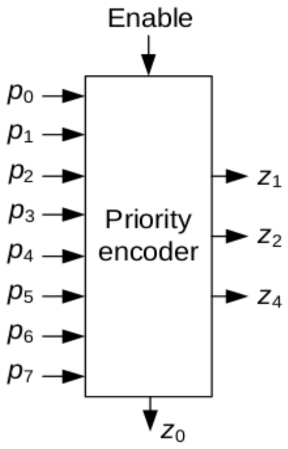


Priority Encoders

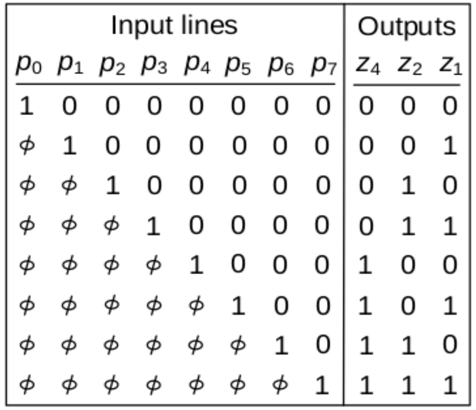
Priority encoder: n input lines and $\log_2 n$ output lines

- 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

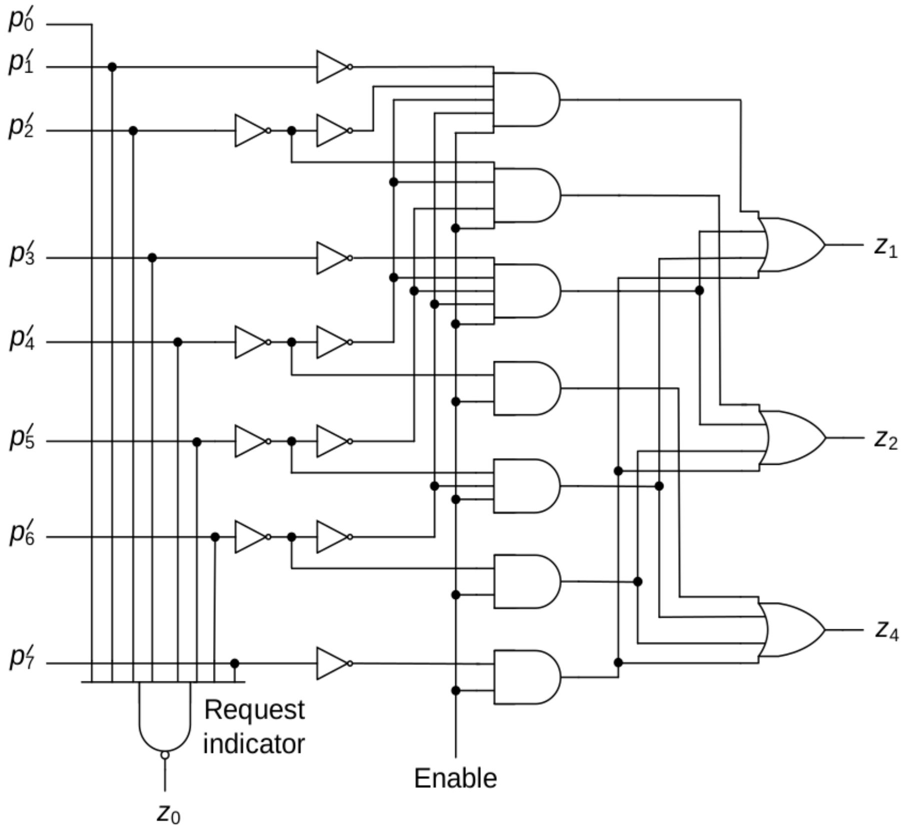
(*b*) Truth table.

$$z_4 = p_4\, p_5' p_6' p_7' + p_5\, p_6' p_7' + p_6\, p_7' + p_7 = p_4 + p_5 + p_6 + p_7$$

$$z_2 = p_2\, p_3' p_4' p_5' p_6' p_7' + p_3 p_4' p_5' p_6' p_7' + p_6 p_7' + p_7 = p_2 p_4' p_5' + p_3 p_4' p_5' + p_6 + p_7$$

$$z_1 = p_1\, p_2' p_3' p_4' p_5' p_6' p_7' + p_3 p_4' p_5' p_6' p_7' + p_5 p_6' p_7' + p_7 = p_1 p_2' p_4' p_6' + p_3 p_4' p_6' + p_5 p_6' + p_7$$

Priority Encoders (Contd.)



(*c*) Logic diagram.

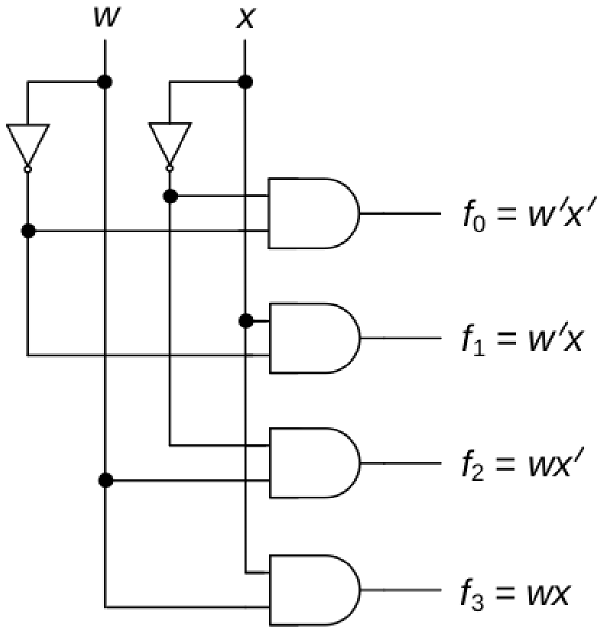
Decoders

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

Useful for:

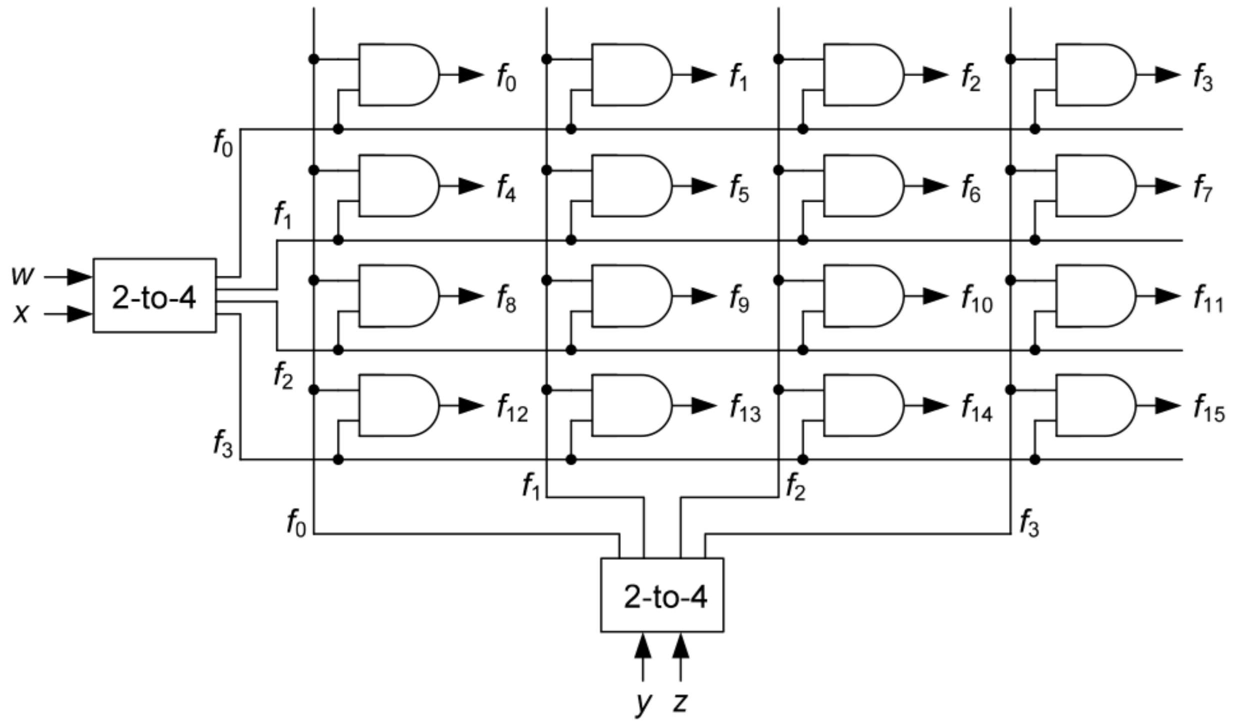
- Routing input data to a specified output line, e.g., in addressing memory
- Basic building blocks for implementing arbitrary switching functions
- Code conversion
- Data distribution

Example: 2-to-4- decoder



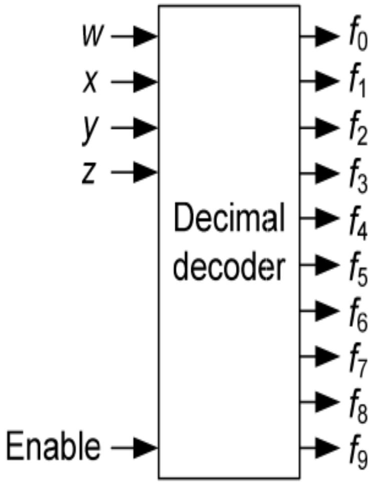
Decoders (Contd.)

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



Decimal Decoder

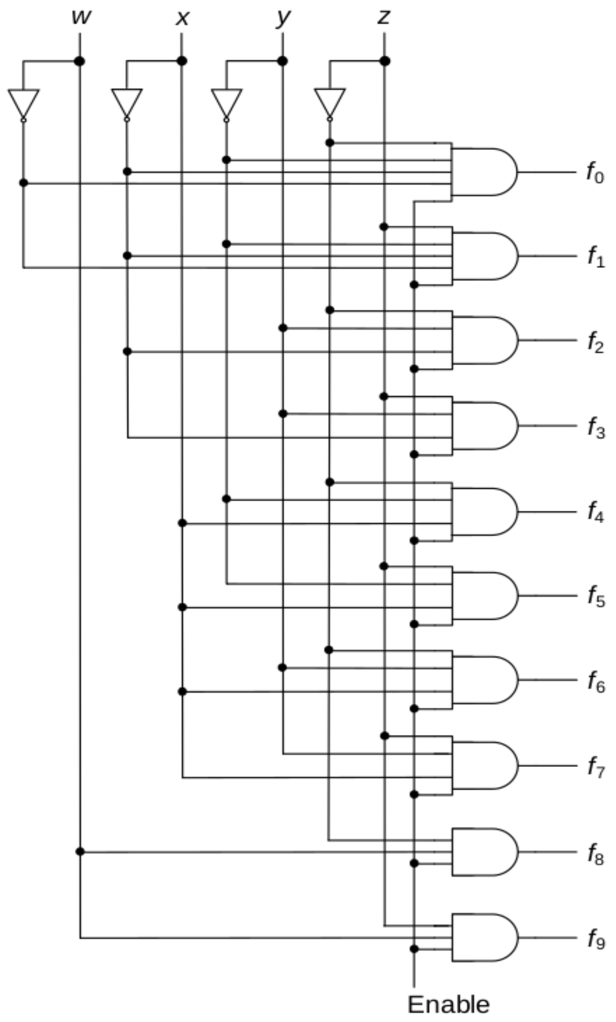
BCD-to-decimal: 4-to-16 decoder made of two 2-to-4 decoders and a gate-switching matrix



(*a*) Block diagram.

$yz \backslash wx$	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

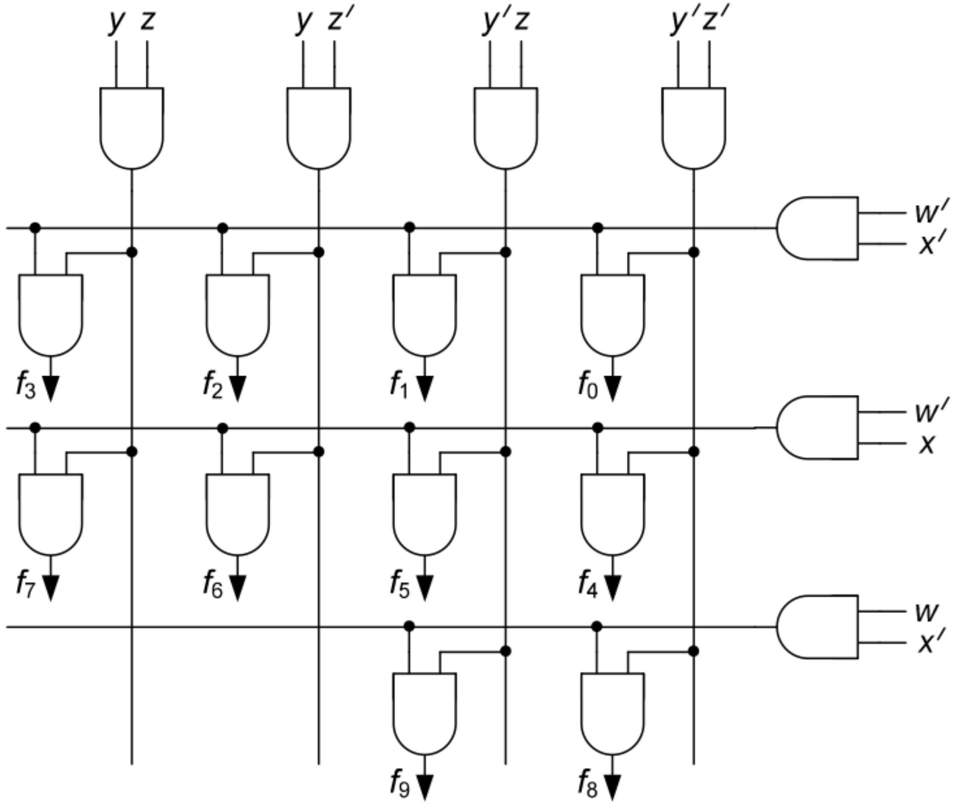
(*b*) Map.



(*c*) Logic diagram.

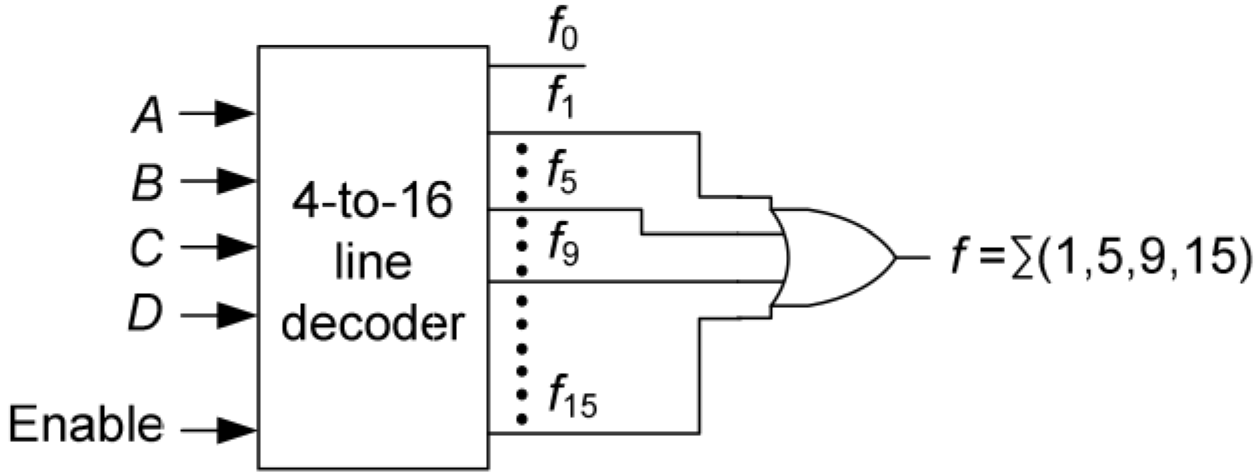
Decimal Decoder (Contd.)

Implementation using a partial-gate matrix:



Implementing Arbitrary Switching Functions

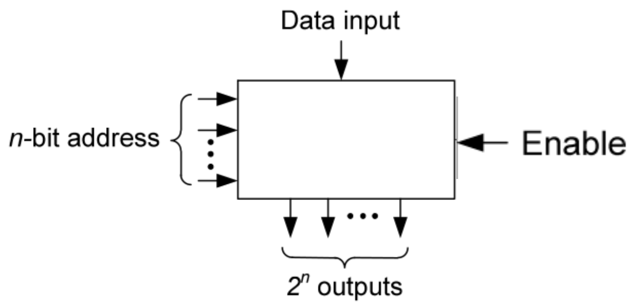
Example: Realize a distinct minterm at each output



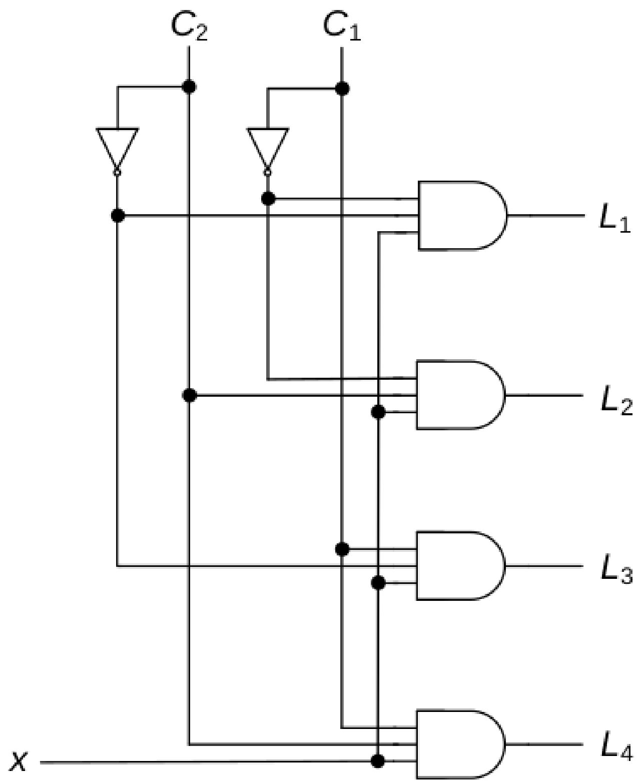
Demultiplexers

Demultiplexers: decoder with 1 data input and n address inputs

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

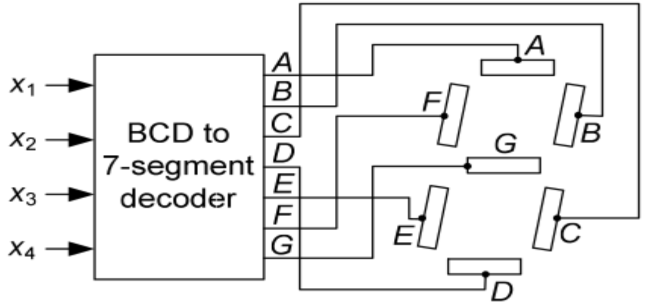


Example: A 4-output demultiplexer



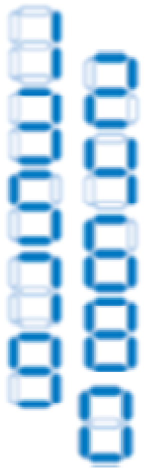
Seven-segment Display

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



Seven-segment pattern and code:

Decimal Digit	BCD code				Seven-segment code						
	x1	x2	x3	x4	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



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

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

$C = x_2 + x_3' + x_4$

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

$E = x_2'x_4' + x_3x_4'$

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

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