CS 225 Switching Theory

Dr. Somanath Tripathy
Indian Institute of Technology Patna

This Class

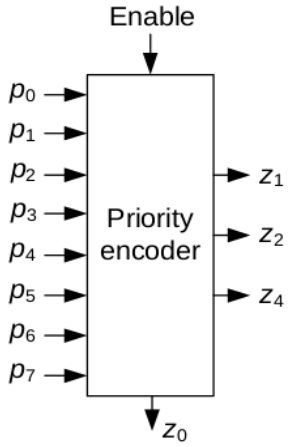
Combinational Circuit logic design

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

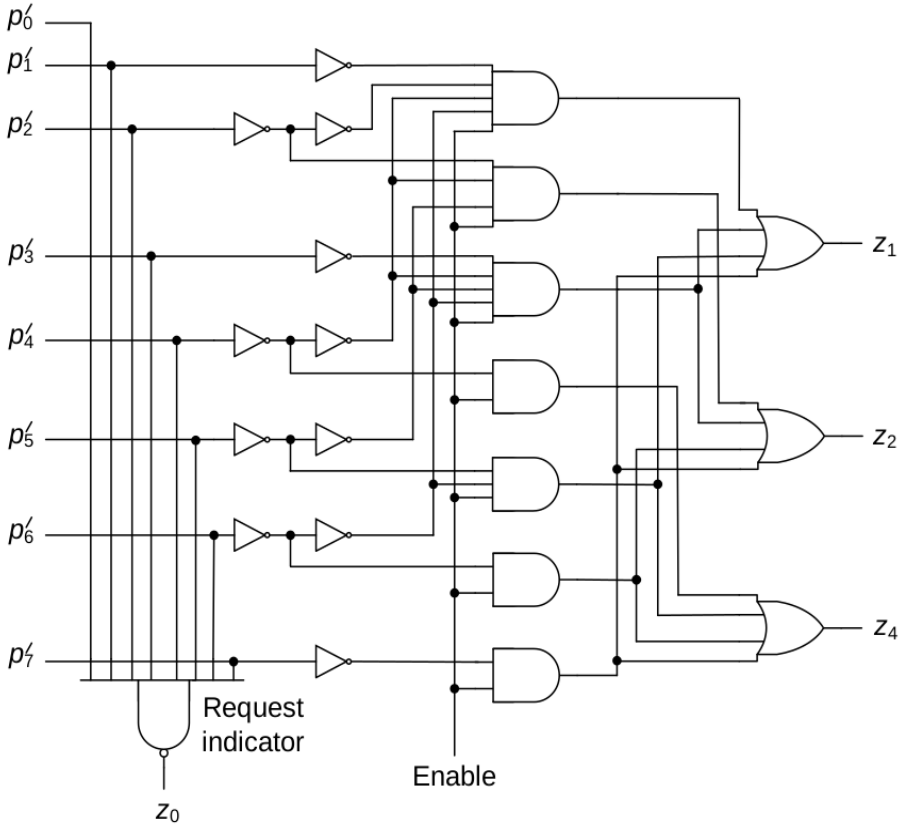
(*b*) Truth table.

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

$z_2 = p_2\, p_3'p_4'p_5'p_6'p_7' + p_3p_4'p_5'p_6'p_7' + p_6p_7' + p_7 = p_2p_4'p_5' + p_3p_4'p_5' + p_6 + p_7$

$z_1 = p_1\, p_2'p_3'p_4'p_5'p_6'p_7' + p_3p_4'p_5'p_6'p_7' + p_5p_6'p_7' + p_7 = p_1p_2'p_4'p_6' + p_3p_4'p_6' + p_5p_6' + p_7$

Priority Encoders (Contd.)



(*c*) Logic diagram.

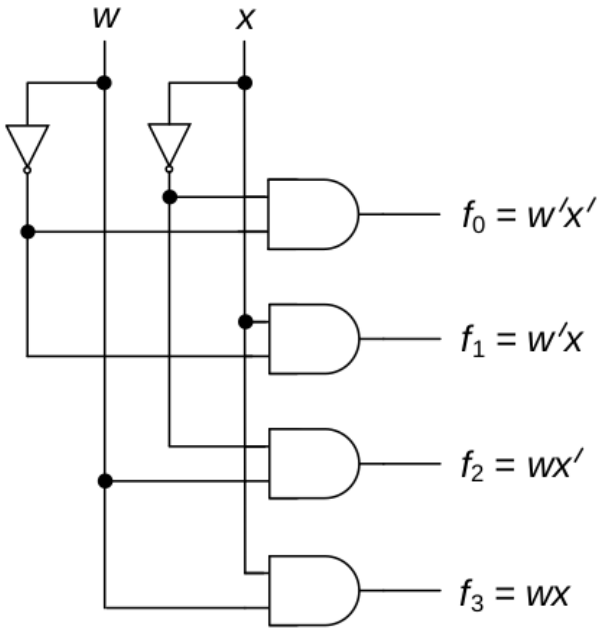
Decoders

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

Useful for:

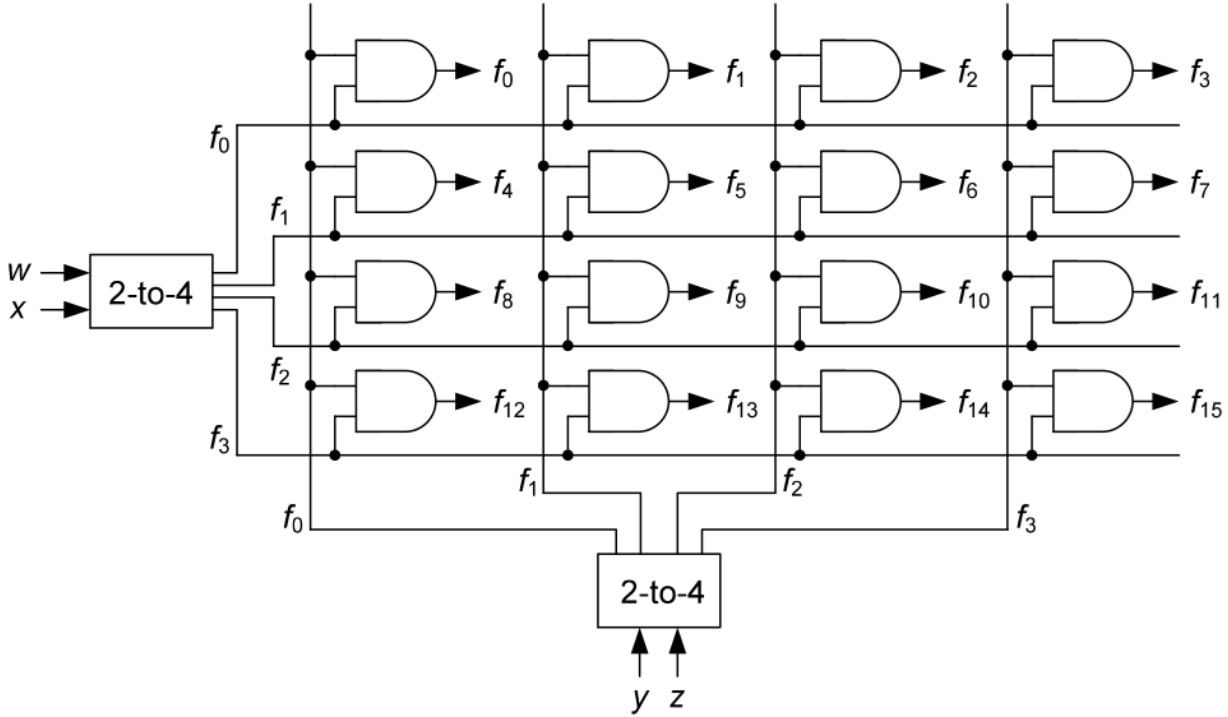
- Routing input data to a specified output line, e.g., in addressing memory
- Basic building blocks for implementing arbitrary switching functions
- Code conversion
- Data distribution

Example: 2-to-4- decoder



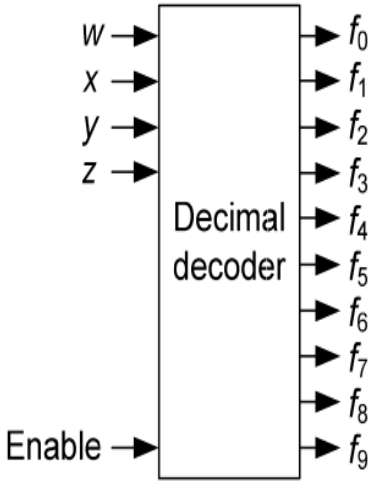
Decoders (Contd.)

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



Decimal Decoder

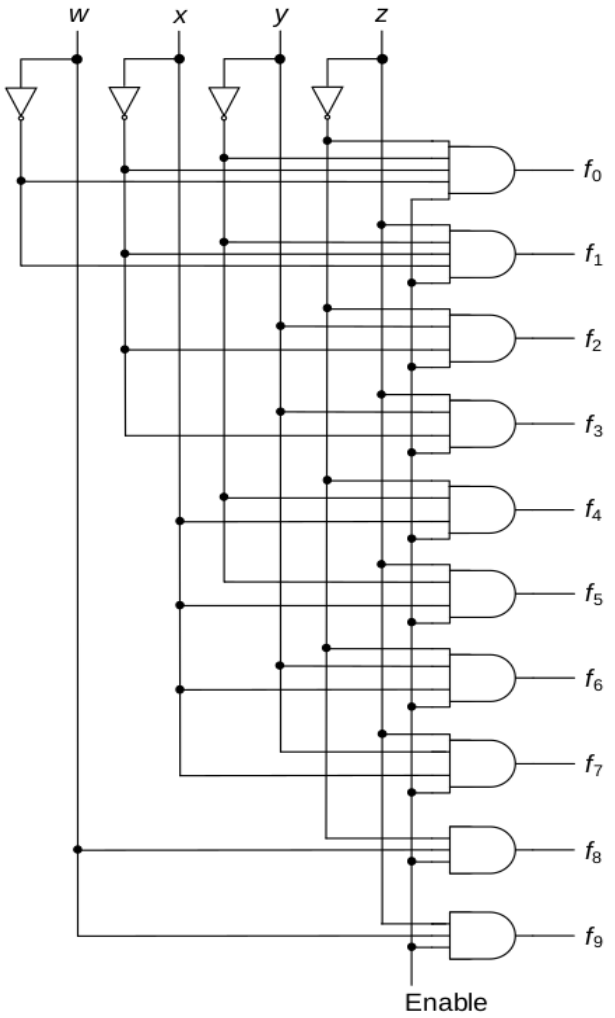
BCD-to-decimal: 4-to-16 decoder made of two 2-to-4 decoders and a gate-switching matrix



(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.

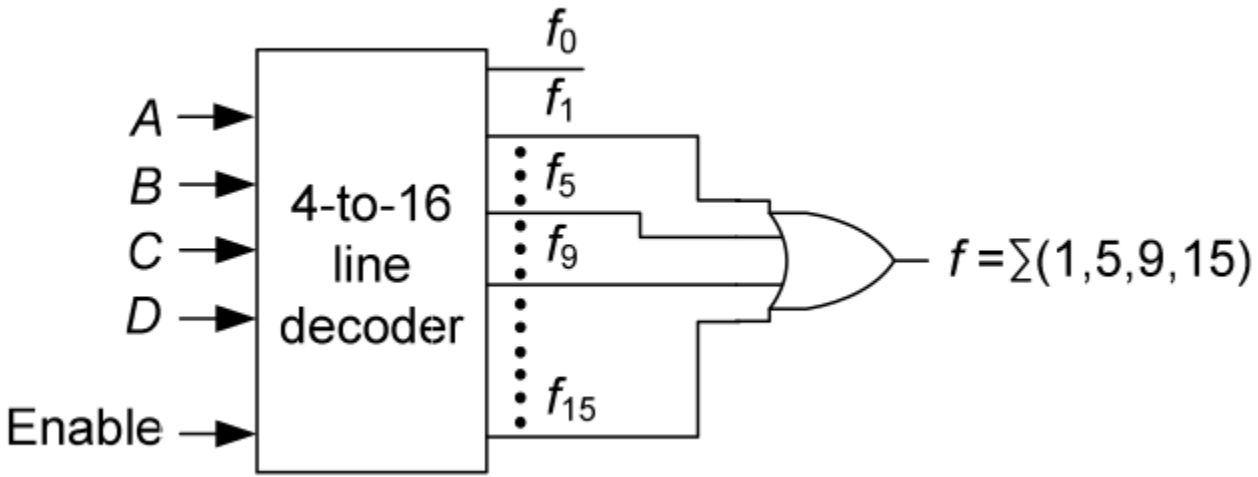


(c) Logic diagram.

Implementing Arbitrary Switching Functions

Example: Realize a distinct minterm at each output

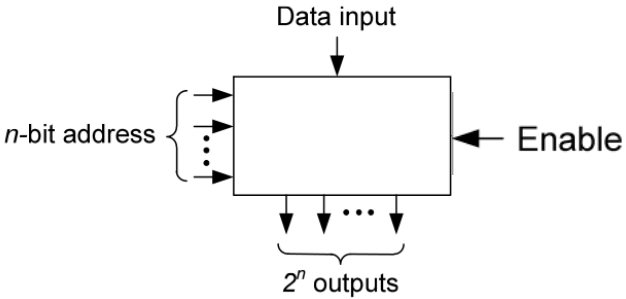
$f = \Sigma(1,5,9,15)$



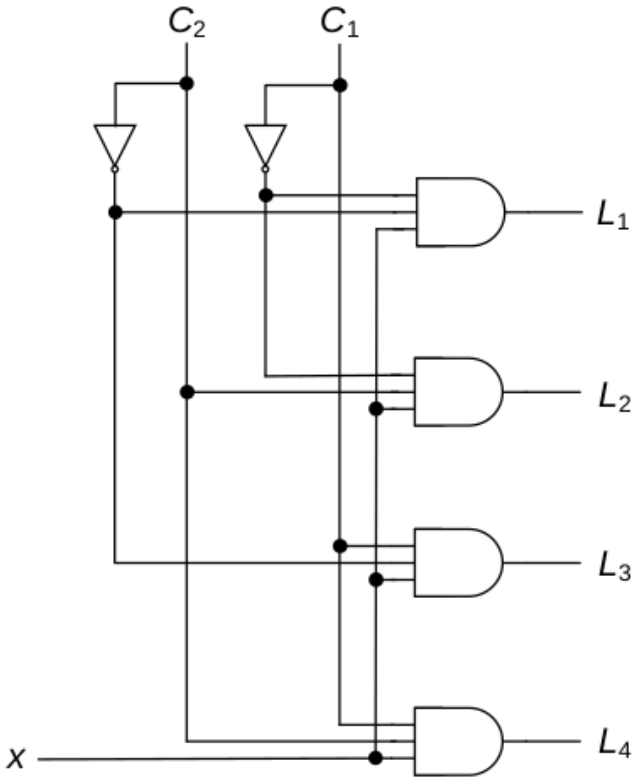
Demultiplexers

Demultiplexers: decoder with 1 data input and n address inputs

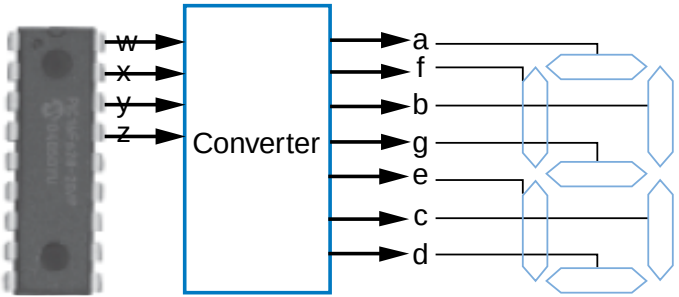
- Directs input to any one of the 2^n outputs

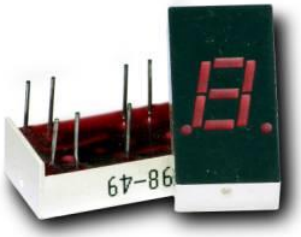


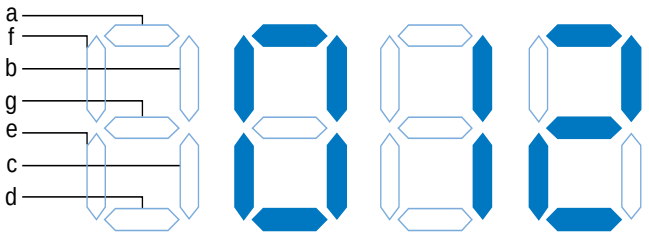
Example: A 4-output demultiplexer



BCD to 7-Segment Converter

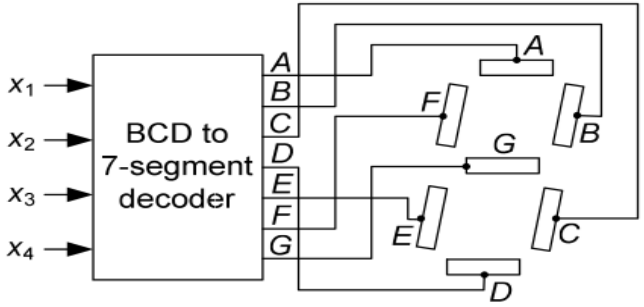






Seven-segment Display

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



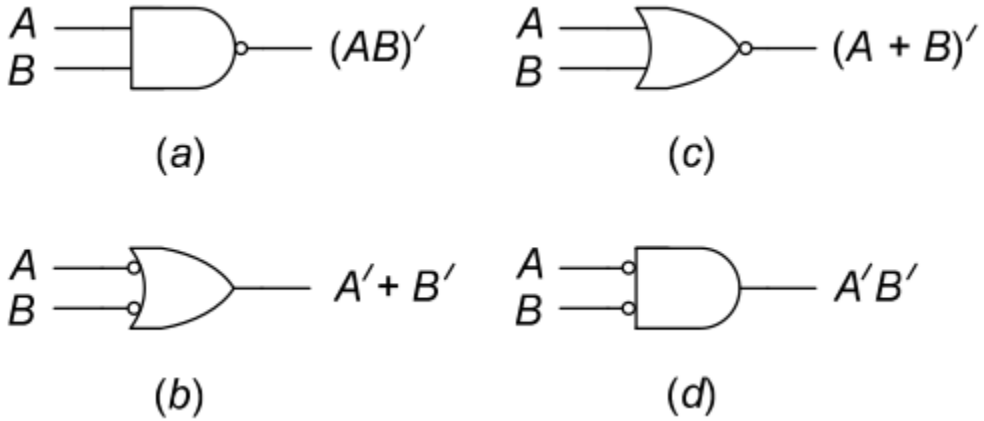
Seven-segment pattern and code:

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

$$A = x_1 + x_2'x_4' + x_2x_4 + x_3x_4$$
$$B = x_2' + x_3'x_4' + x_3x_4$$
$$C = x_2 + x_3' + x_4$$
$$D = x_2'x_4' + x_2'x_3 + x_3x_4' + x_2x_3'x_4$$
$$E = x_2'x_4' + x_3x_4'$$
$$F = x_1 + x_2x_3' + x_2x_4' + x_3'x_4'$$
$$G = x_1 + x_2'x_3 + x_2x_3' + x_3x_4'$$

NAND/NOR Circuits

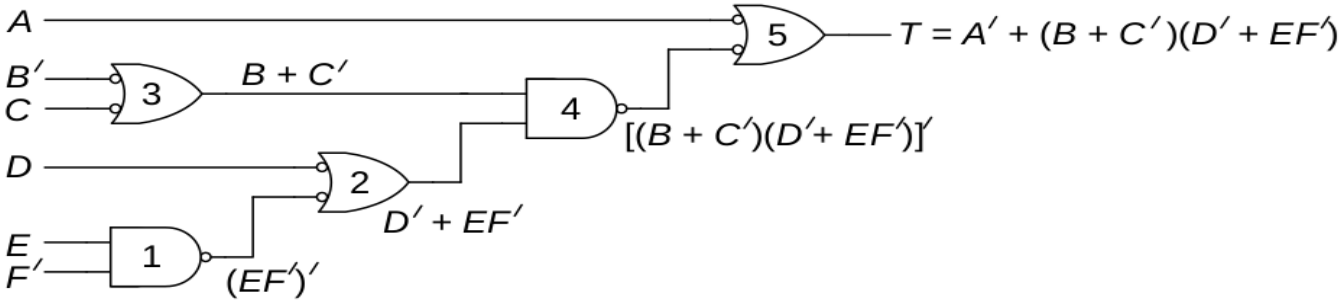
Switching algebra: not directly applicable to NAND/NOR logic



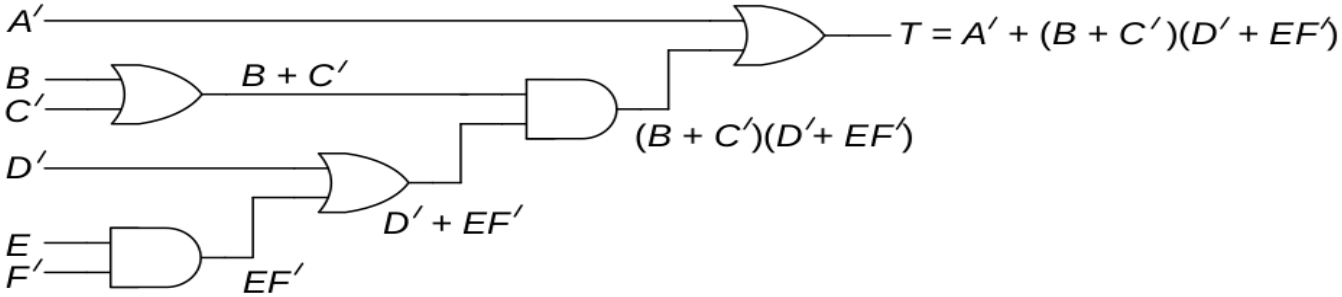
NAND and NOR gate symbols

Analysis of NAND/NOR Networks

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



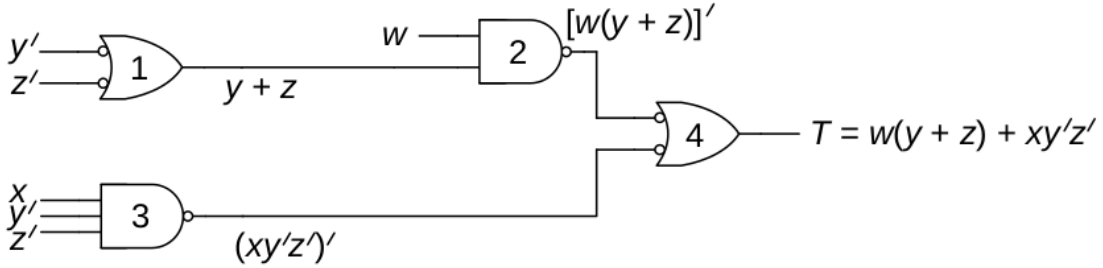
(*a*) NAND-logic circuit.



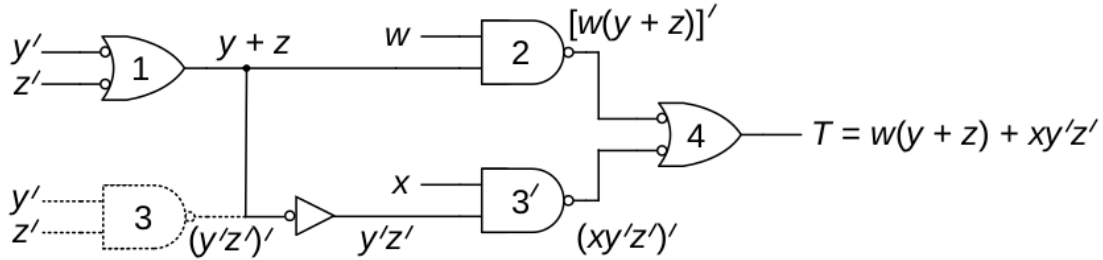
(*b*) Logically equivalent AND-OR circuit.

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

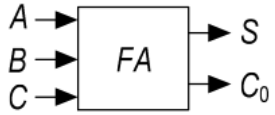
Example:

0	1	1		=	carry-in
1	0	0	1	=	augend
0	0	1	1	=	addend
1	1	0	0	=	sum

Truth table, block diagram and expressions:

A	B	C	S	C0
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	1	1	0	1
1	0	0	1	0

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

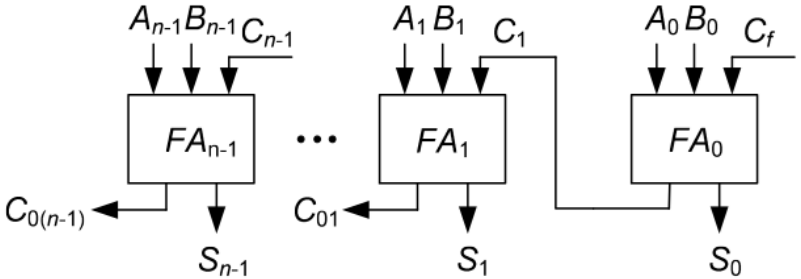


(*b*) Block diagram.

Ripple-carry Adder

Ripple-carry adder: Stages of full adders

- C_f : forced carry
- $C_0(n-1)$: overflow carry



$$S_i = A_i \oplus B_i \oplus C_i$$

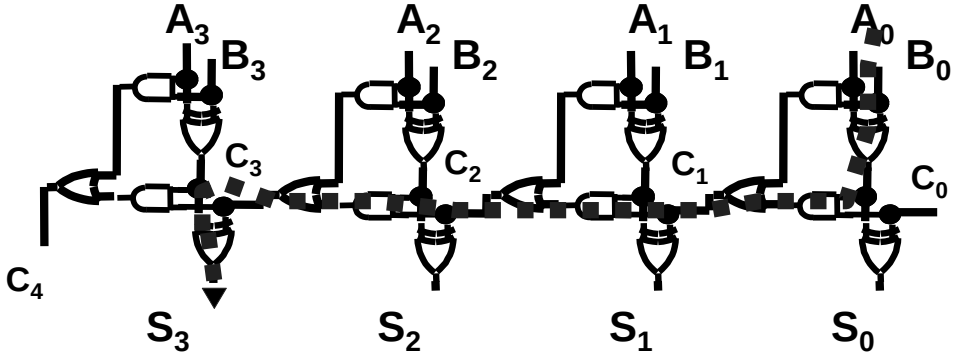
$$C_{0i} = A_iB_i + A_iC_i + B_iC_i$$

Time required:

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

Carry Propagation & Delay

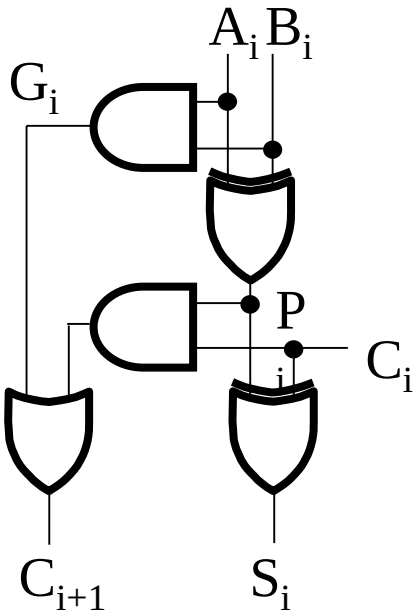
- One problem with the addition of binary numbers is the length of time to propagate the ripple carry from the least significant bit to the most significant bit.
- The gate-level propagation path for a 4-bit ripple carry adder of the last example:



- Note: The "long path" is from A_0 or B_0 though the circuit to S_3.

Carry Lookahead

- Given Stage i from a Full Adder, we know that there will be a carry generated when $A_i = B_i =$ "1", whether or not there is a carry-in.
- Alternately, there will be a carry propagated if the "half-sum" is "1" and a carry-in, C_i occurs.
- These two signal conditions are called *generate*, denoted as G_i, and *propagate*, denoted as P_i respectively and are identified in the circuit:



Carry Lookahead **(continued)**

- In the ripple carry adder:
  - Gi, Pi, and Si are local to each cell of the adder
  - Ci is also local each cell
- In the carry lookahead adder, in order to reduce the length of the carry chain, C_i is changed to a more global function spanning multiple cells
- Defining the equations for the Full Adder in term of the P_i and G_i:

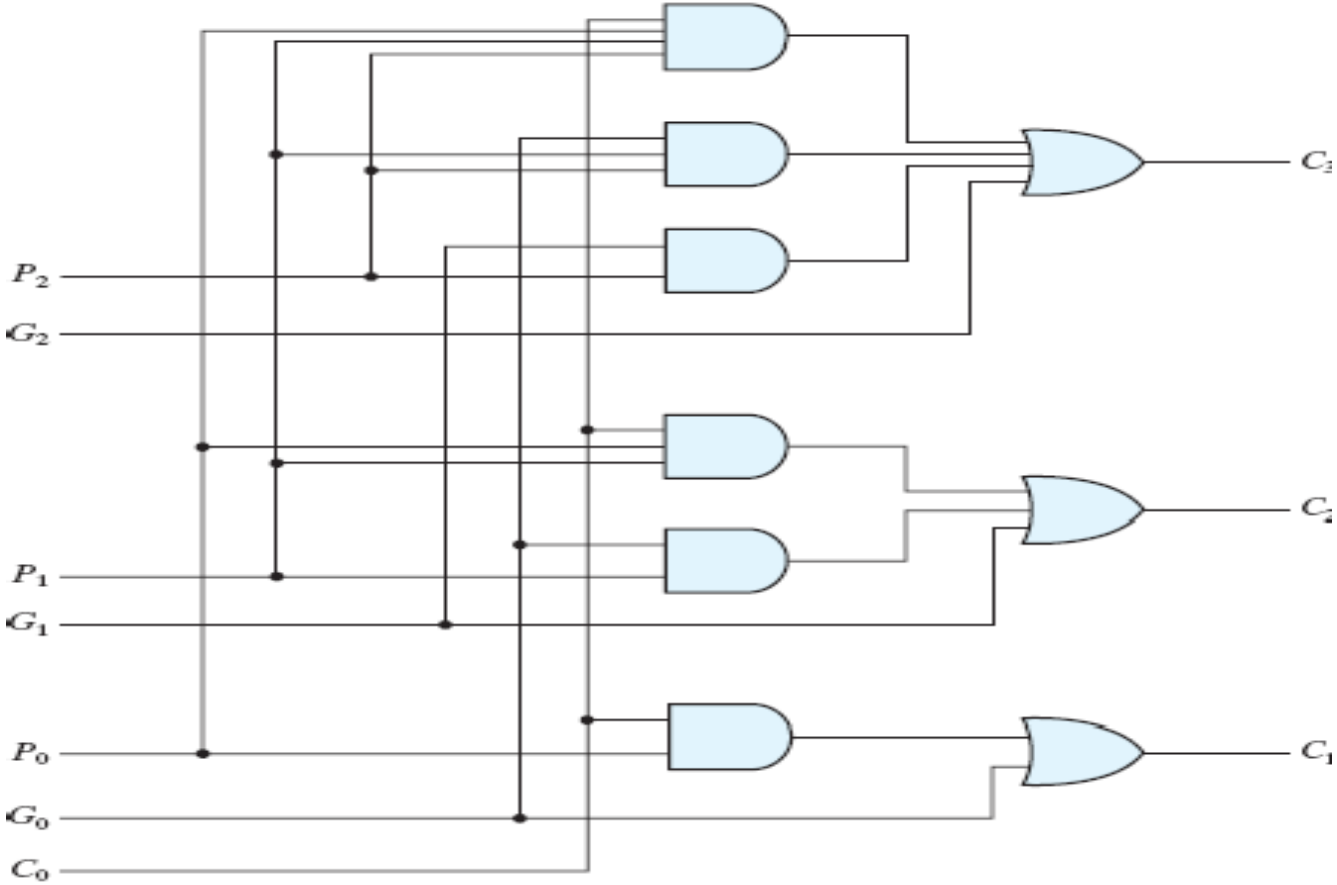
$$S_i = P_i \oplus C_i \qquad C_{i+1} = G_i + P_i C_i$$

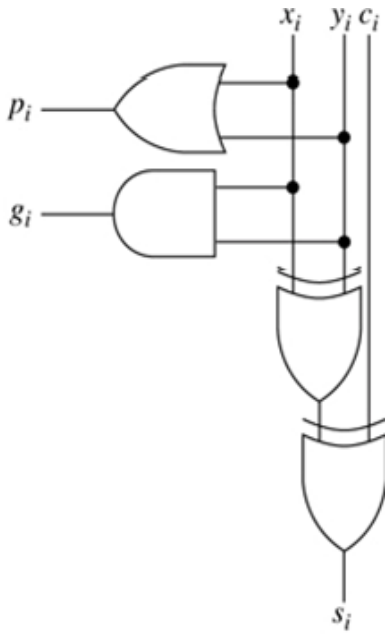
$$P_i = A_i \oplus B_i \qquad G_i = A_i B_i$$

Example - 4-bit Adder

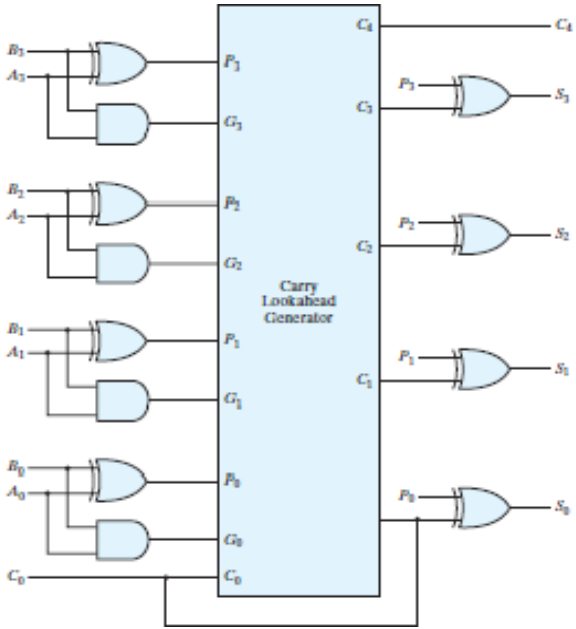
$$
\begin{aligned}
c_1 &= G_0 + c_0 P_0, \\
c_2 &= G_1 + G_0 P_1 + c_0 P_0 P_1, \\
c_3 &= G_2 + G_1 P_2 + G_0 P_1 P_2 + c_0 P_0 P_1 P_2, \\
c_4 &= G_3 + G_2 P_3 + G_1 P_2 P_3 + G_0 P_1 P_2 P_3 + c_0 P_0 P_1 P_2 P_3
\end{aligned}
$$

A carry lookahead generator.





A carry lookahead Adder



Thanks

Q3 on next week
Mid Sem is as schedule

Wish you all have a Good Exam