ECE2277a: DIGITAL LOGIC SYSTEMS
Electrical and Computer Engineering
Western University

Unit 4: Combinational Logic

Outline

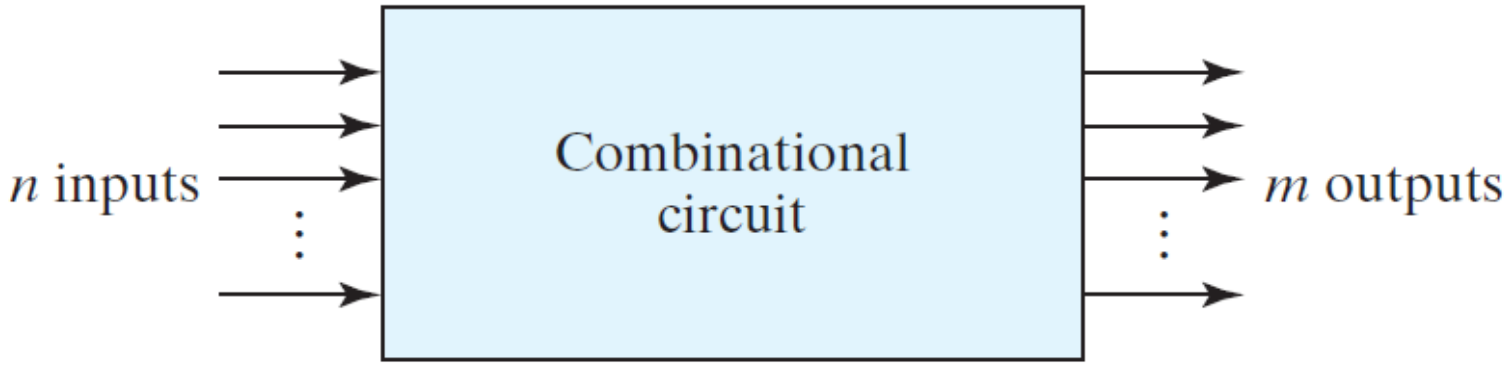
- Combinational Circuits
  - Analysis and Design procedures
  - Arithmetic circuits
    - Binary, Decimal, and Carry-Lookahead Adders/Subtractors, Binary Multiplier
  - Comparators
  - Decoders
  - Encoders
  - Multiplexers
- Sections 4.1 to 4.11

Logic Circuits

- There are two types of logic circuits for digital systems: combinational and sequential.
- Combinational circuits consist of logic gates whose outputs at any time are determined from only the present combination of inputs.
  - They can be specified logically by a set of Boolean functions.
- Sequential circuits contain storage elements in addition to logic gates
  - Their outputs are a function of the inputs and the state of the storage elements.
  - The outputs depend not only on present values of inputs, but also on past inputs.

Combinational Circuit

- A combinational circuit consists of input variables, logic gates, and output variables.



- For *n* input variables, there are 2^n possible binary input combinations.
- For each possible input combination, there is one possible output value.
- It can be specified with a truth table with *n* inputs and *m* outputs
- Also, it can be described by *m* Boolean functions expressed in terms of the *n* input variables.

Combinational Circuit

Analysis

- Analysis is to determine the function of a given circuit. It starts with a given logic diagram and ends with
  - a set of Boolean functions
  - a truth table
  - or, possibly, an explanation of the circuit operation.
- It is the opposite way of design process
- The 1st step is to make sure that the circuit is combinational and not sequential:
  - There is no memory element in the circuit
  - The circuit should not contain feedback paths

Combinational Circuit

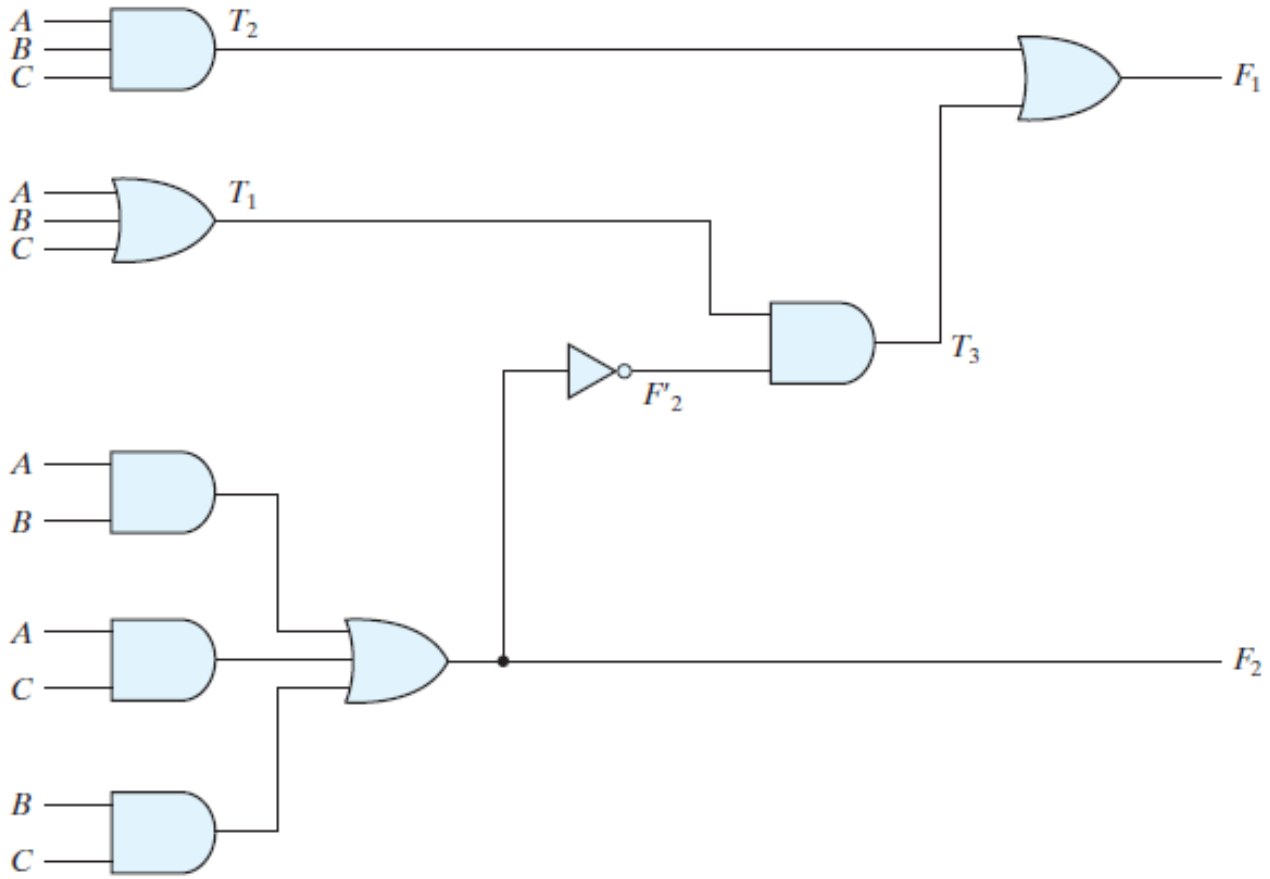
- A feedback path is a connection from the output of one gate to the input of a second gate that forms part of the input to the first gate.

- To obtain the output Boolean functions:
  - Label all nodes with arbitrary symbols
  - Starting from the outputs and using Boolean algebra to determine the Boolean functions for the circuit outputs

Combinational Circuit

Analysis: Example 1

Obtain Boolean functions:

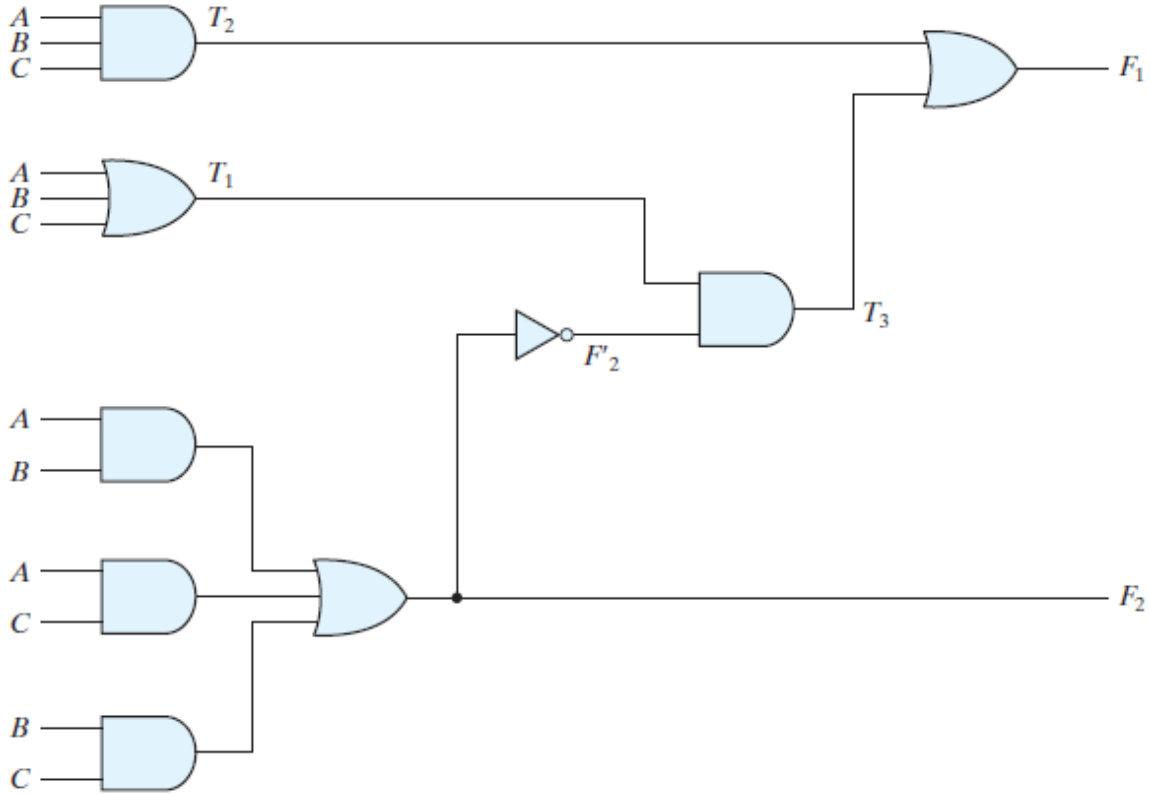


Combinational Circuit

Analysis Procedure

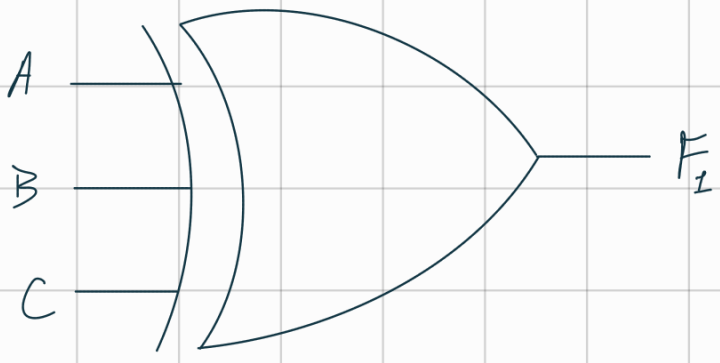
- To obtain the truth table directly from the circuit
  - Determine the number of circuit inputs=n → the truth table has 2^n rows
  - Starting from inputs, obtain the truth table of the required nodes until the truth table of the circuit outputs are found

#	A	B	C	F_2	F'_2	T_1	T_2	T_3	F_1
0	0	0	0	0	1	0	0	0	0
1	0	0	1	0	1	1	0	1	1
2	0	1	0	0	1	1	0	1	1
3	0	1	1	1	0	1	0	0	0
4	1	0	0	0	1	1	0	1	1
5	1	0	1	1	0	1	0	0	0
6	1	1	0	1	0	1	0	0	0
7	1	1	1	1	0	1	1	0	1



A \ BC	00	01	11	10
0	0	1	0	1
1	1	0	1	0

$$F_1 = A'B'C + A'BC' + AB'C' + ABC$$



Combinational Circuit

Design Procedure

- It starts from the specification to obtain a logic circuit diagram or a set of Boolean functions:

  1) Determine the required number of inputs and outputs and assign a symbol to each.

  2) Derive the truth table that defines the required relationship between inputs and outputs.

  3) Obtain the simplified Boolean functions for each output as a function of the input variables.

  4) Draw the logic diagram and verify the correctness of the design (manually or by simulation)

Combinational Circuit

Design Procedure: Example 2

Design a BCD to Excess-3 Code Conversion:

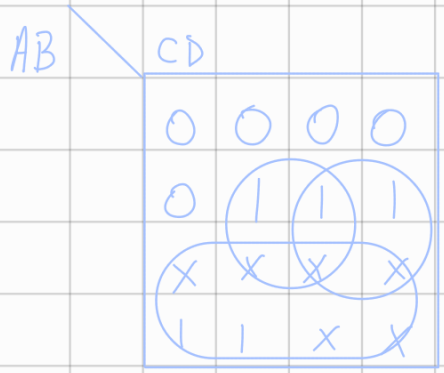
1. Has n=4 inputs (ABCD) and m=4 outputs (wxyz)
2. Truth table →

3. Simplified Boolean functions: all outputs have don't care conditions

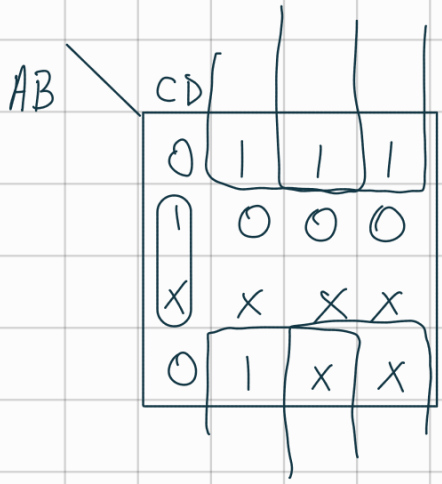
$$d(A,B,C,D)=\Sigma(10,11,12,13,14,15)$$

Input BCD code				Output Excess-3 code			
A	B	C	D	w	x	y	z
0	0	0	0	0	0	1	1
0	0	0	1	0	1	0	0
0	0	1	0	0	1	0	1
0	0	1	1	0	1	1	0
0	1	0	0	0	1	1	1
0	1	0	1	1	0	0	0
0	1	1	0	1	0	0	1
0	1	1	1	1	0	1	0
1	0	0	0	1	0	1	1
1	0	0	1	1	1	0	0

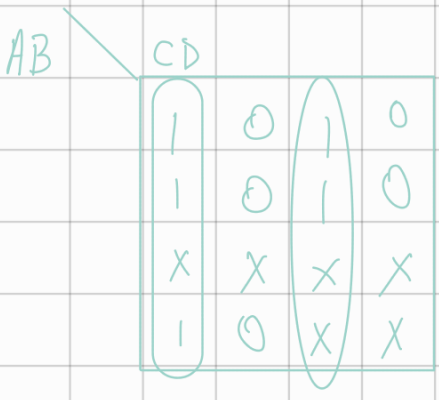
the rest are 'don't cares'



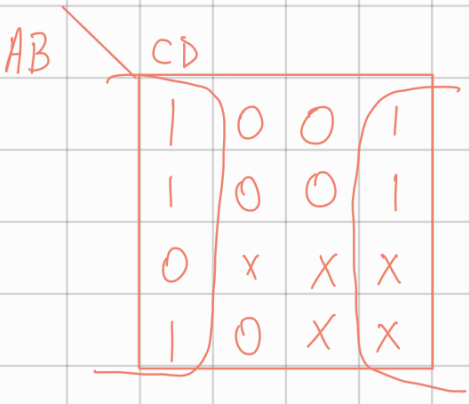
$W = A + BD + BC$



$X = BC'D' + B'D + B'C$



$Y = C'D' + CD$



$Z = D'$

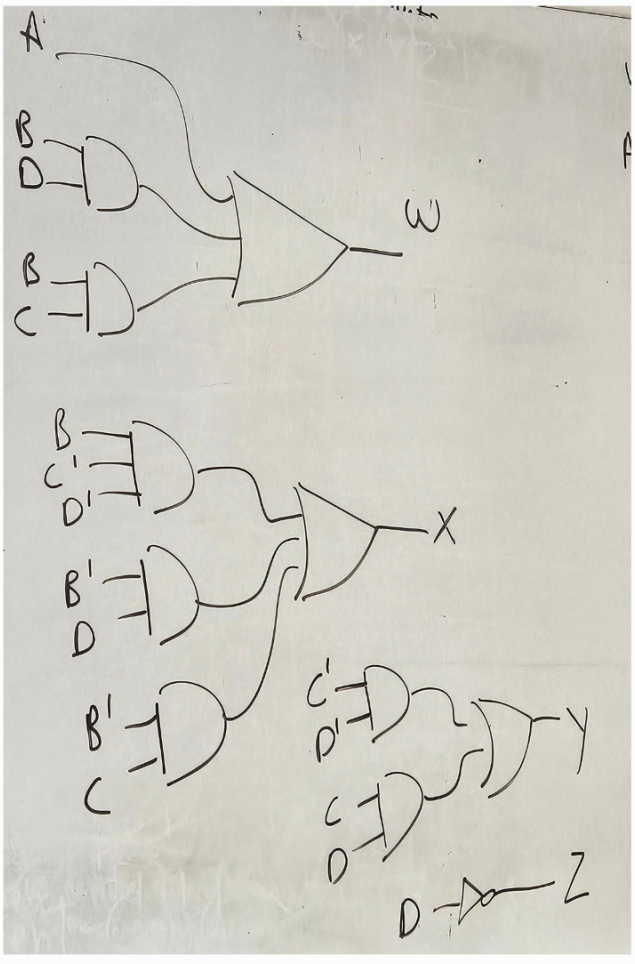
Combinational Circuit

Solution: BCD to Excess-3 Code Conversion:

- A two-level AND-OR logic diagram may be obtained directly from the Boolean expressions

$$z = D'$$
$$y = CD + C'D'$$
$$x = B'C + B'D + BC'D'$$
$$w = A + BC + BD$$

- 7 AND gates (with 2x6+3=15 inputs) and 3 OR gates (with 2+3+3=8 inputs)
  Cost: 10(# of gates)+23 (gate inputs)=33



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

$$X = B'(C+D) + B[C'D']$$

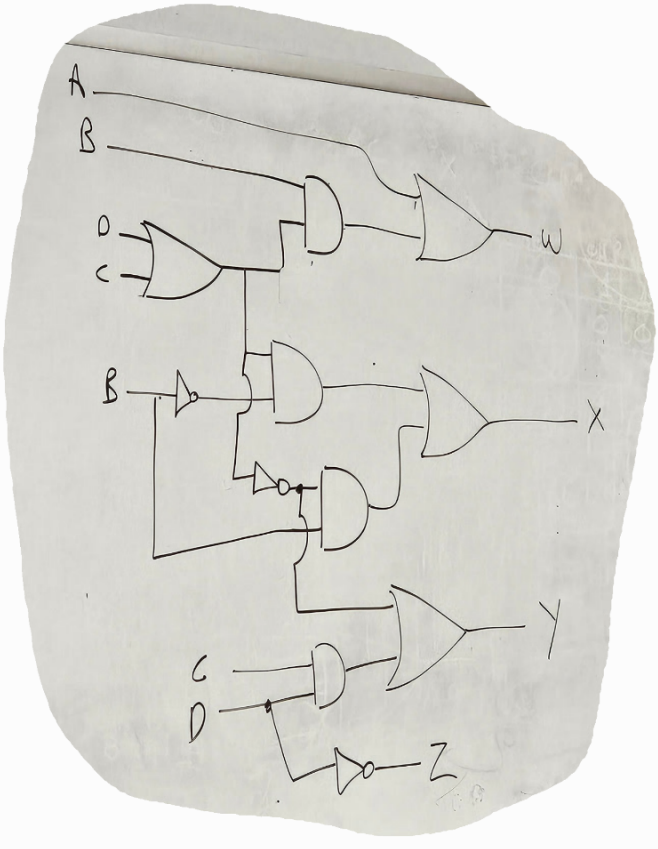
↳ De Morgan

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

$$Y = CD + C'D'$$

$$= CD + (C+D)'$$

$$Z = D'$$



New cost = 24

Previous cost = 33

Combinational Circuit

Solution: BCD to Excess-3 Code Conversion:

- The expressions may be manipulated algebraically for the purpose of using common gates for two or more outputs.

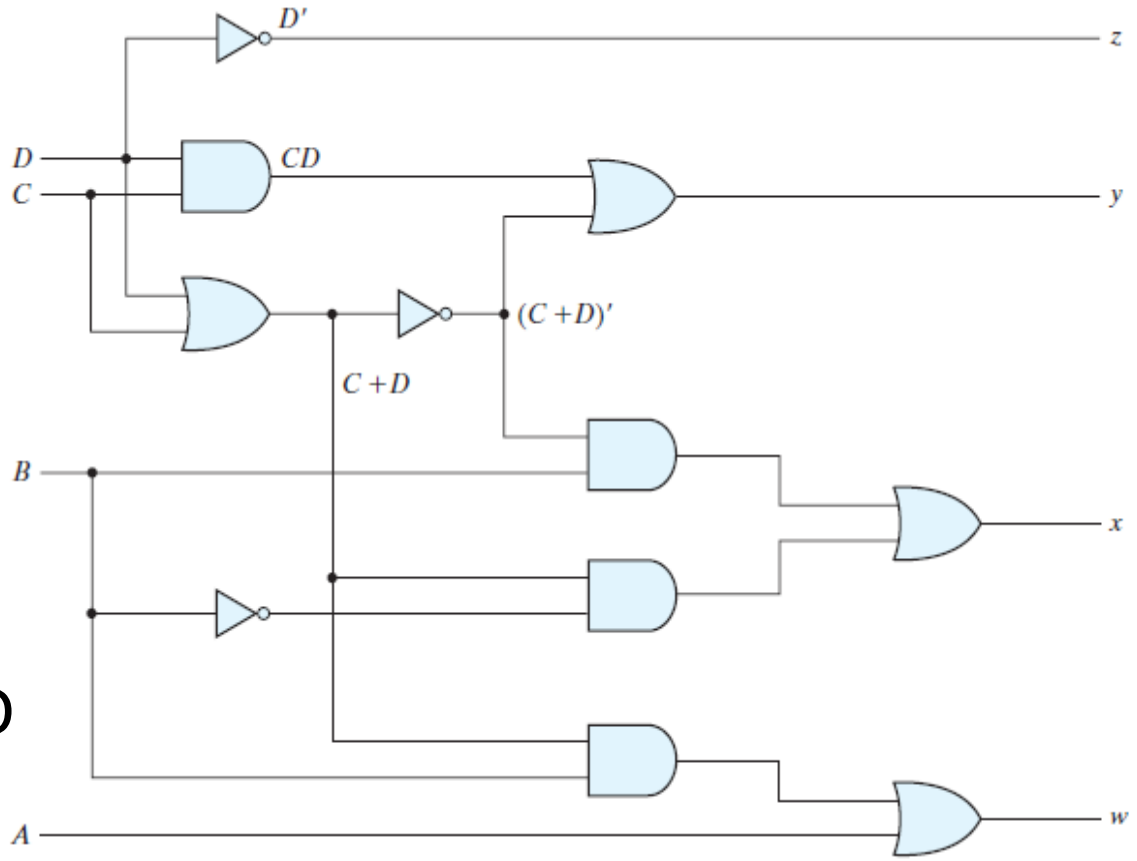
  $z=D'$

  $y = CD + C'D' = CD + (C+D)'$

  $x = B'C + B'D + BC'D' = B'(C+D) + BC'D'$

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

  $w = A + BC + BD = A + B(C+D)$



- It results in multiple-level implementation
- Assume that B' and D' are available: 4 AND gates, 4 OR gates and 1 NOT are required.
- What is the cost?

Decoder

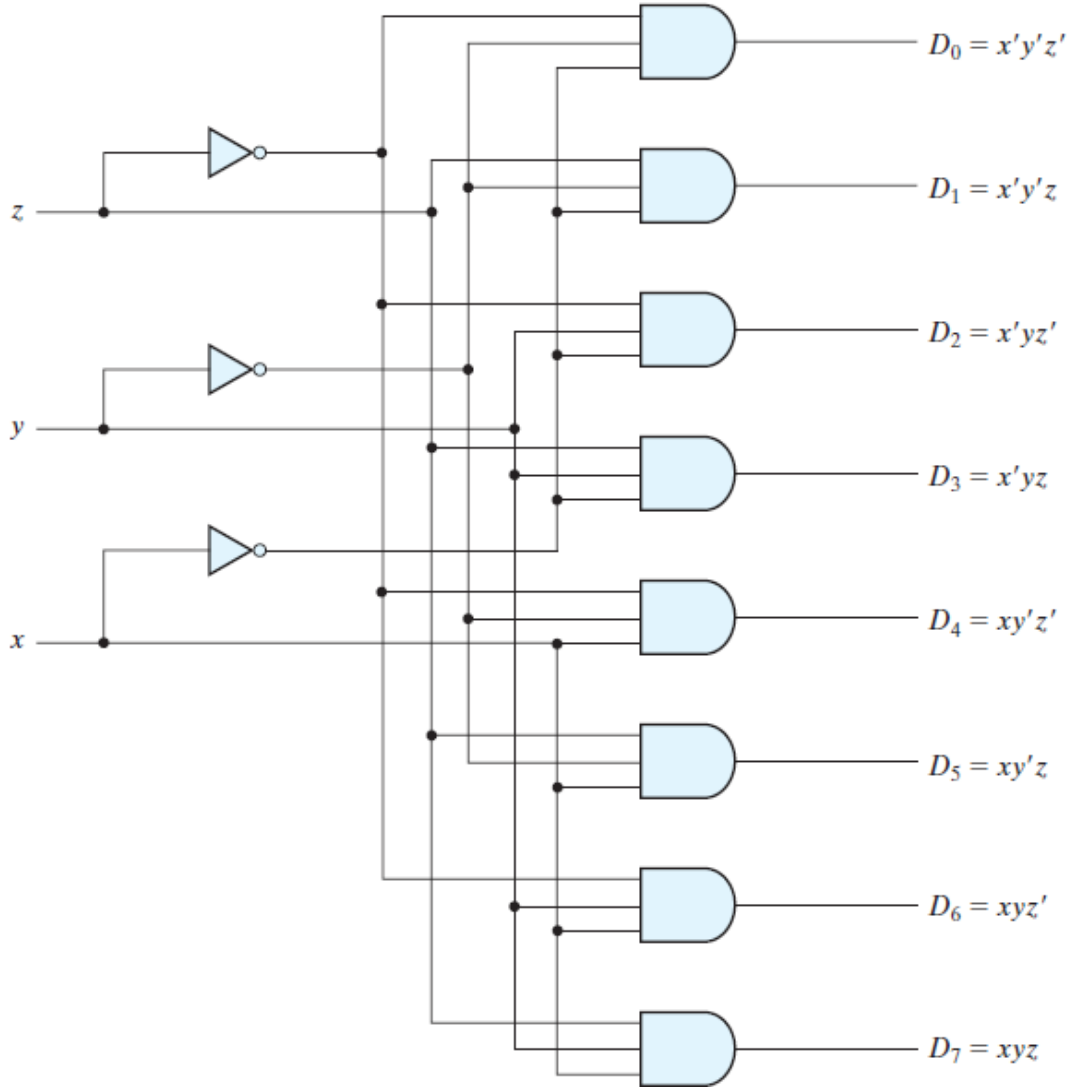
- A *decoder* is a combinational circuit that converts binary information from *n* input lines to $m \leq 2^n$ unique output lines.
- In active-high outputs, each output is a minterm. An example: 3-to-8 line decoder

Table 4.6
Truth Table of a Three-to-Eight-Line Decoder

Inputs			Outputs							
x	*y*	*z*	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1

Decoder

- It can be used for binary-to-octal conversion
- Each output is a minterm for a 3-variable function

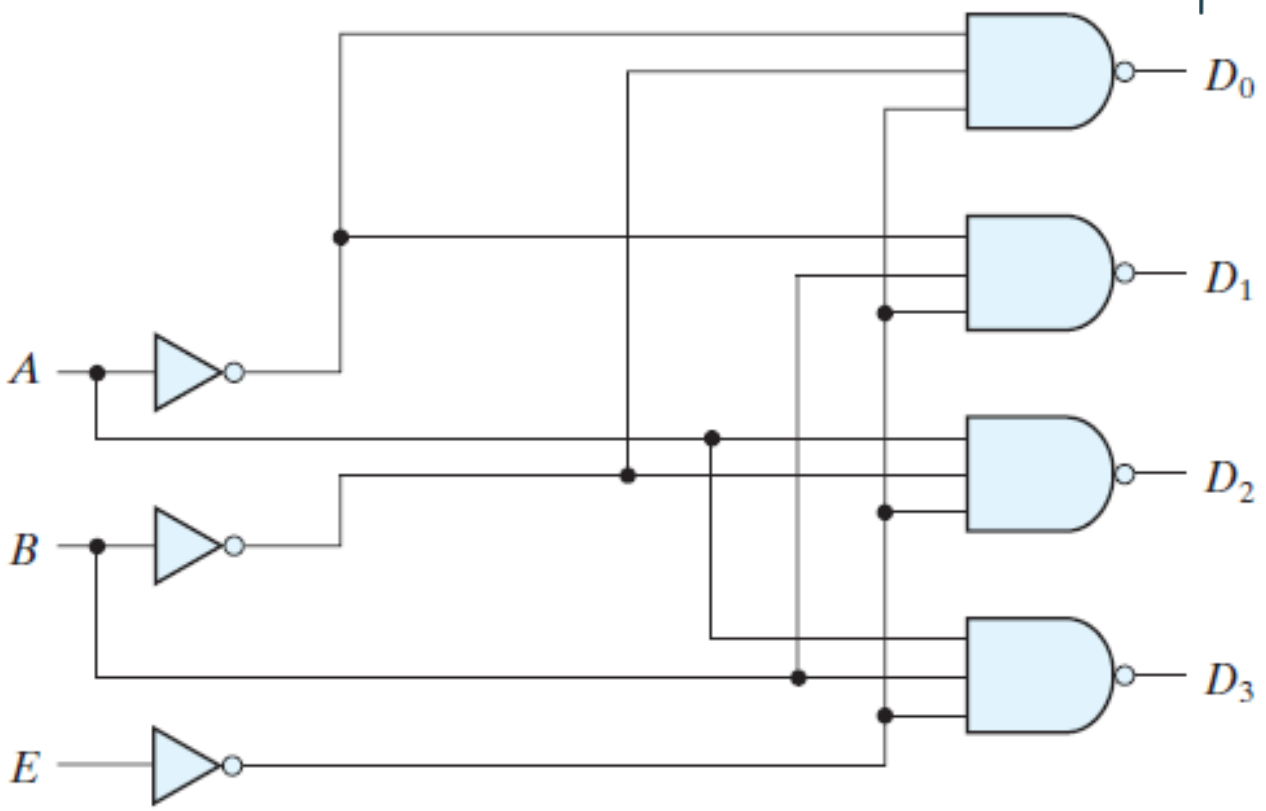


Decoder with Enable

- Design a 2-to-4 decoder with the following truth table
  - Active-low outputs
  - Active-low enable (E)

E	A	B	D_0	D_1	D_2	D_3
1	X	X	1	1	1	1
0	0	0	0	1	1	1
0	0	1	1	0	1	1
0	1	0	1	1	0	1
0	1	1	1	1	1	0

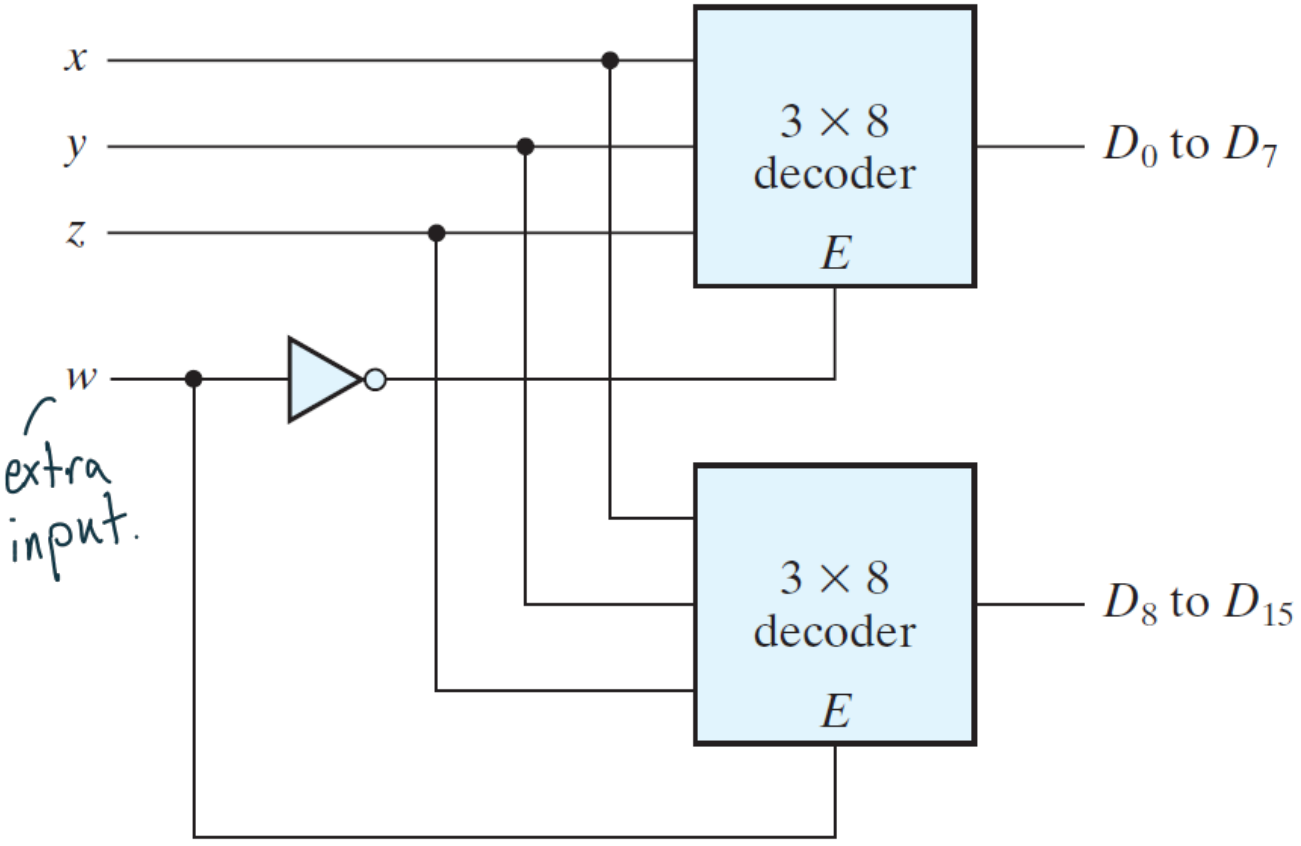
limits to 5 inputs



Same process, use k-maps

Decoder with Enable

- Decoders with enable inputs can be connected together to form a larger decoder circuit
- Example 3: Form a 4x16 decoder using 2 3x8 decoders with enable input



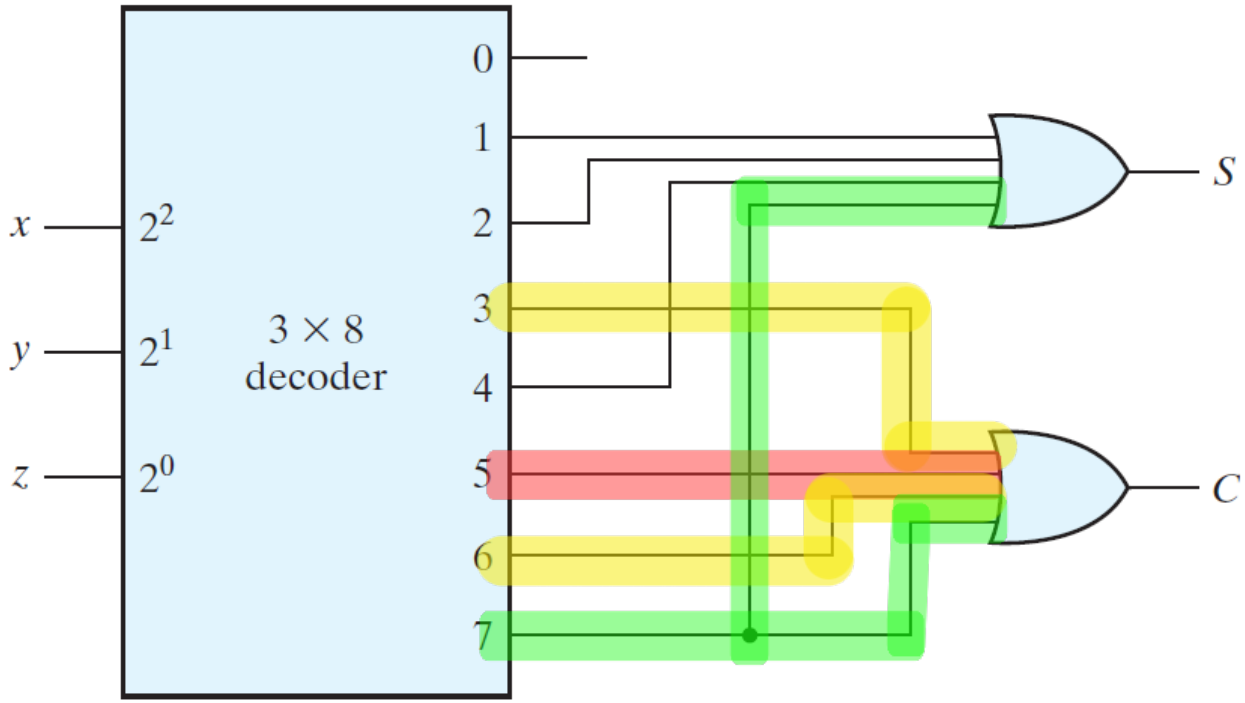
w x y z	D_0 D_1 … D_7	D_8 D_9 … D_{15}

Complete the truth table

Decoder

Combinational Logic Implementation using Decoders

- A decoder with active-high outputs provides the 2^n minterms of n inputs
- Example 4: Implementing a full adder (FA) using a decoder with active-high outputs



x	y	z	C	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

7-Segment Display

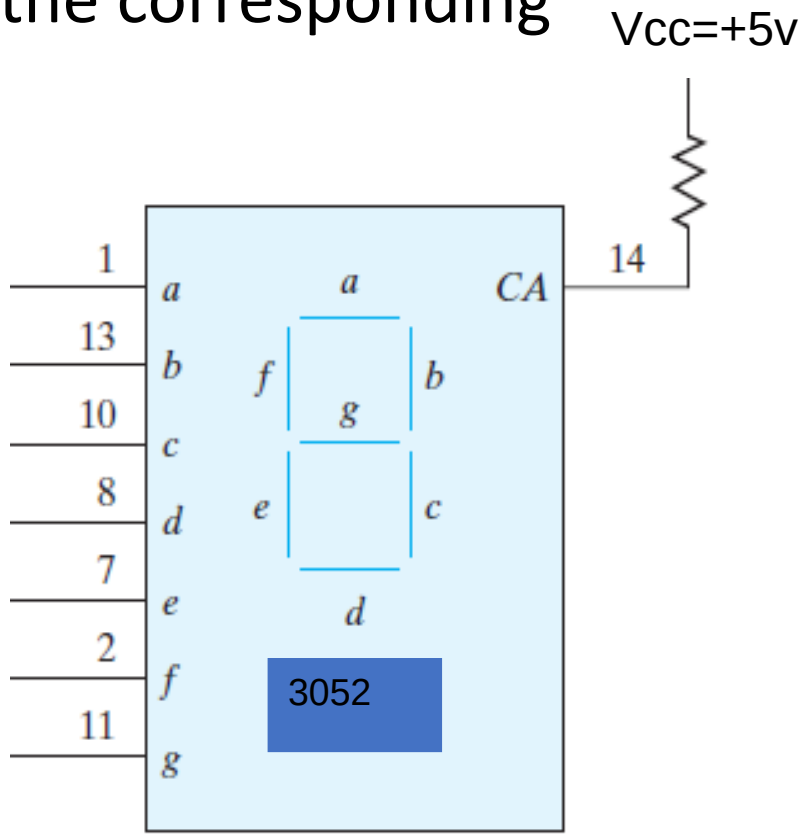
- A 7-segment display is used for displaying any one of the decimal digits **0** through 9.
- It contains the 7 LED (light-emitting diode) segments on top of the package controlled by inputs a, b, c, d, e, f, and g.
- The input 0 to the input a to g of display will turn the corresponding segment on
- To display the number 0 to the input a to g of display will turn the corresponding segment on

  a=b=f=c=e=d=0

  g=1
- To display a small o shown below the inputs should be

  a=b=f=1

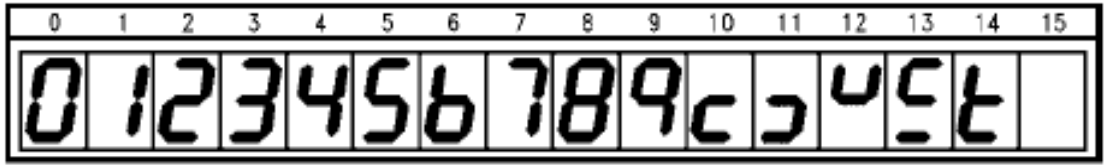
  c=d=e=g=0



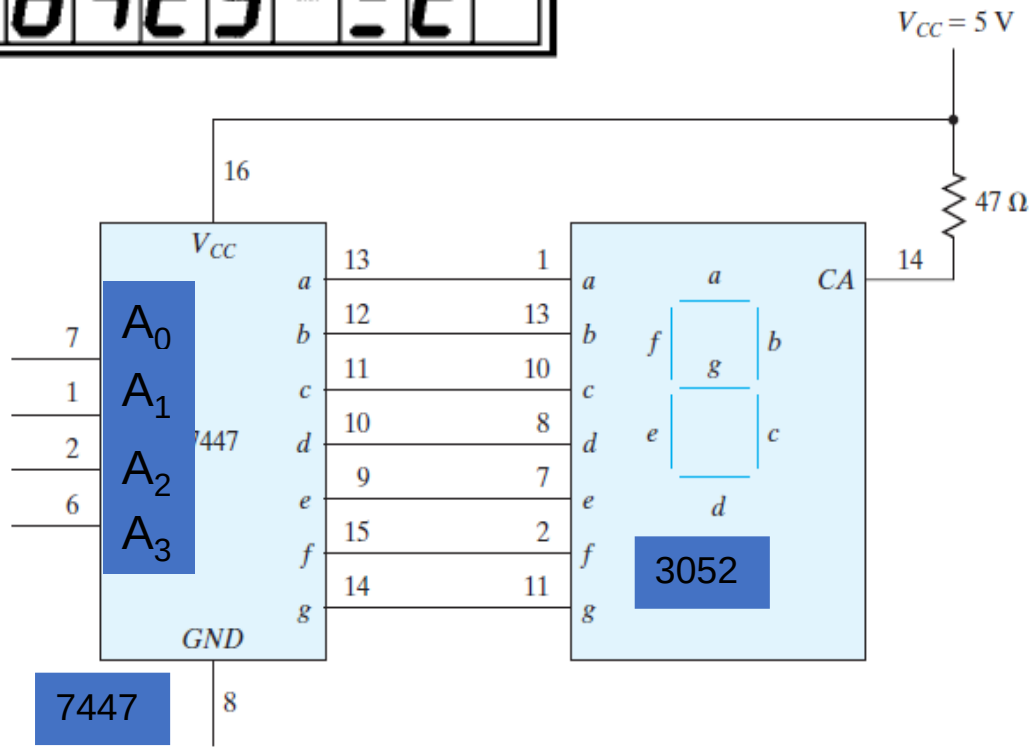
7-Segment Display

- A BCD-to-7-segment decoder (7447) is a combinational circuit that accepts a decimal digit in BCD ($A_3A_2A_1A_0$) and generates the corresponding 7-segment code (a to g).

Numerical Designations—Resultant Displays



Example 5: Design the 7447 component

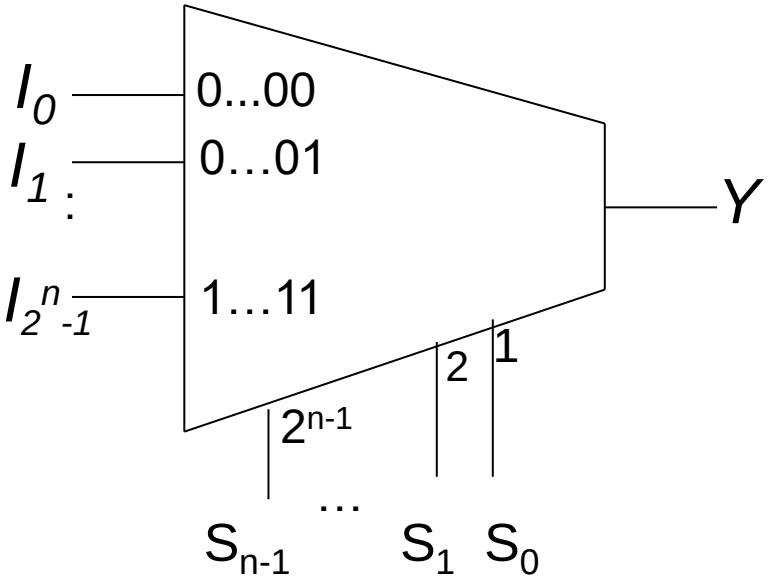


Multiplexer (MUX)

- A multiplexer is a combinational circuit that selects binary information from one of 2^n lines (I_j's $0 \le j \le 2^n-1$) and directs it to a single output line (Y).

- The selection of a particular input line is controlled by a set of n selection lines ($S_{n-1}\ldots S_1S_0$).

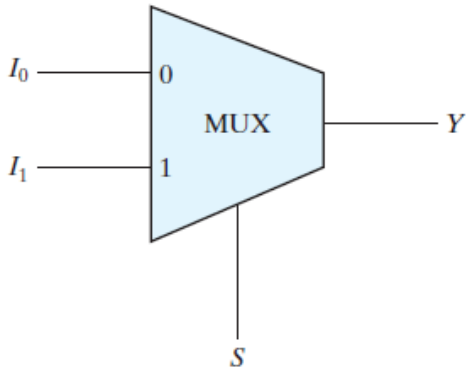
- The MUX acts like an electronic switch that selects one of 2^n inputs



$S_{n-1}\ldots S_1S_0$	Y
0 … 0 0	I_0
0 … 0 1	I_1
⋮	
1 … 1 1	I_{2^n-1}

Multiplexer (MUX)

Example: A 2-to-1 (2x1) MUX (n=1)

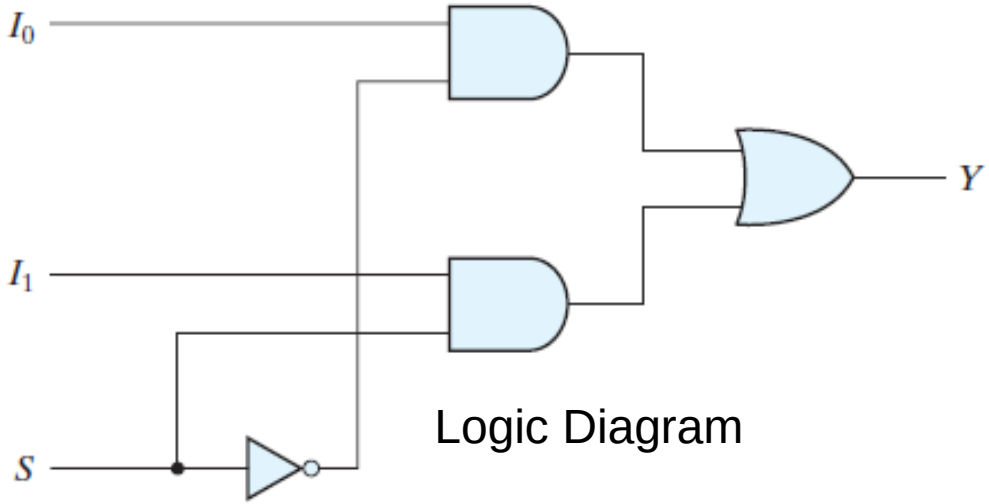


Block diagram

S	Y
0	I_0
1	I_1

Truth (Function) table

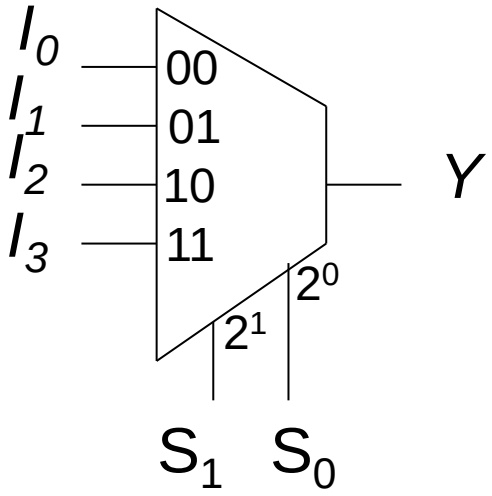
$$Y=I_0S'+I_1S$$



Logic Diagram

Multiplexer (MUX)

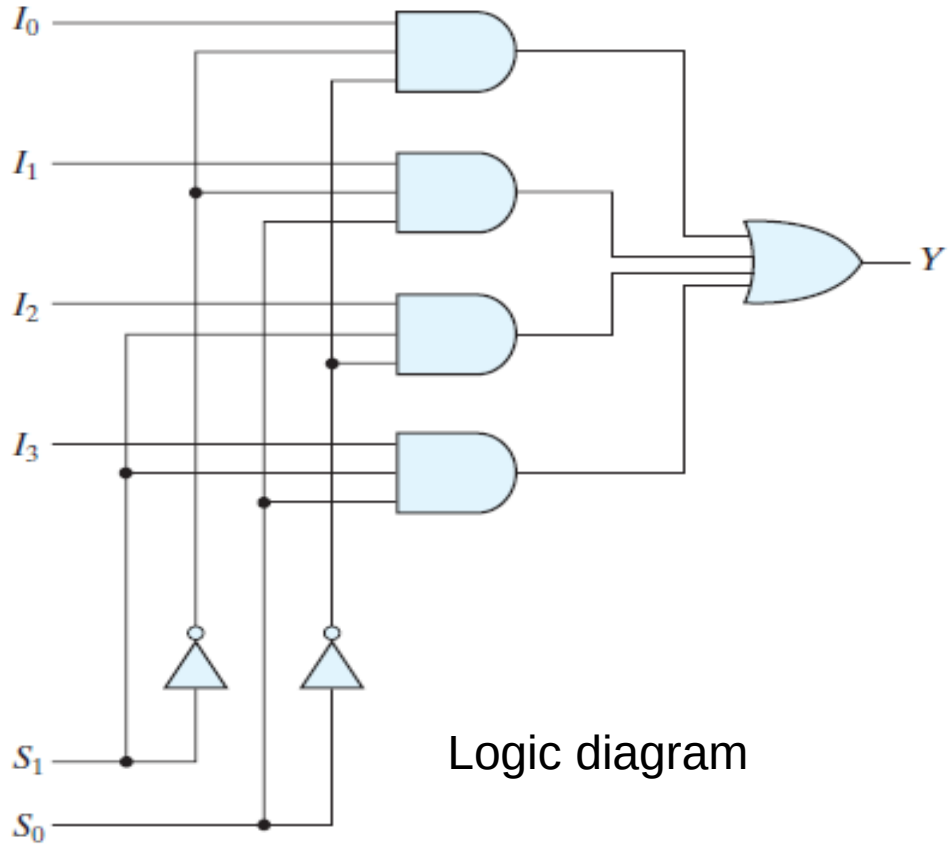
Example: A 4-to-1 (4 x 1) MUX



Block diagram

S_1	S_0	Y
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3

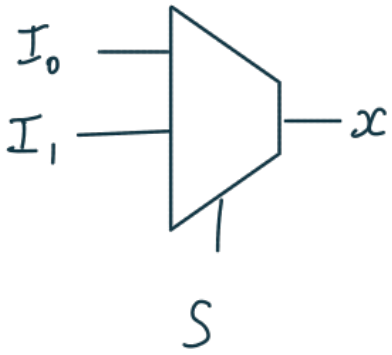
Function table



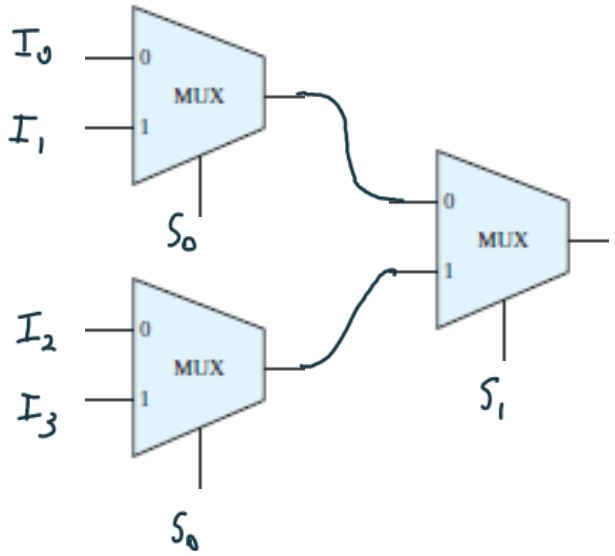
Logic diagram

Multiplexer (MUX)

- Example 6: Using only 2-to-1 multiplexers to build a 4-to-1 multiplexer



S	x
0	I_0
1	I_1

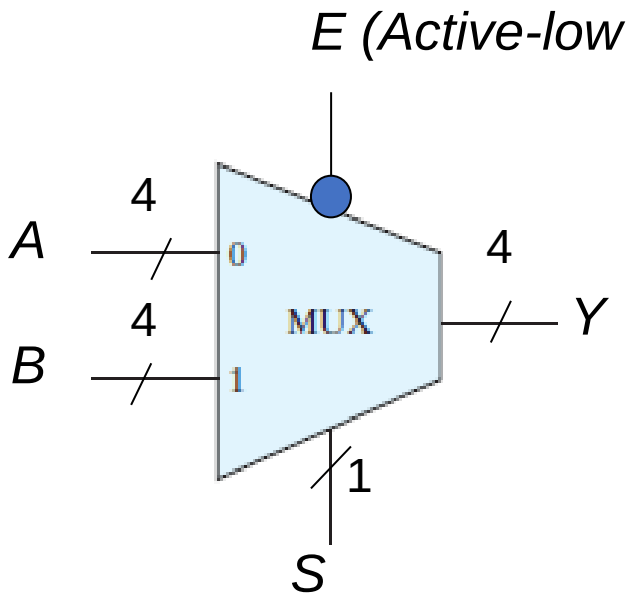


S_1	S_0	Y
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3

Function table

Multiplexer (MUX)

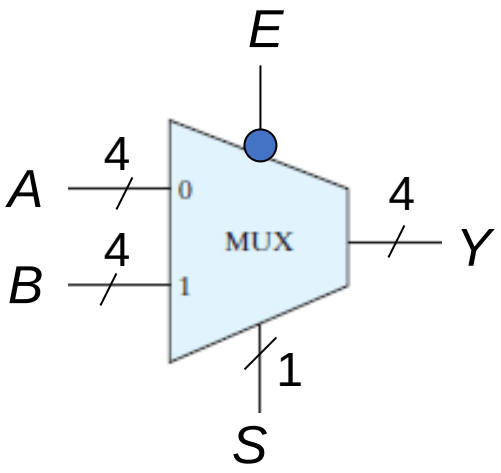
- Some MUXs provide multiple-bit selection with common selection inputs
- Some Muxs have an enable signal. The enable signal must be active for normal operation
- Example: A quadruple 2x1 Mux

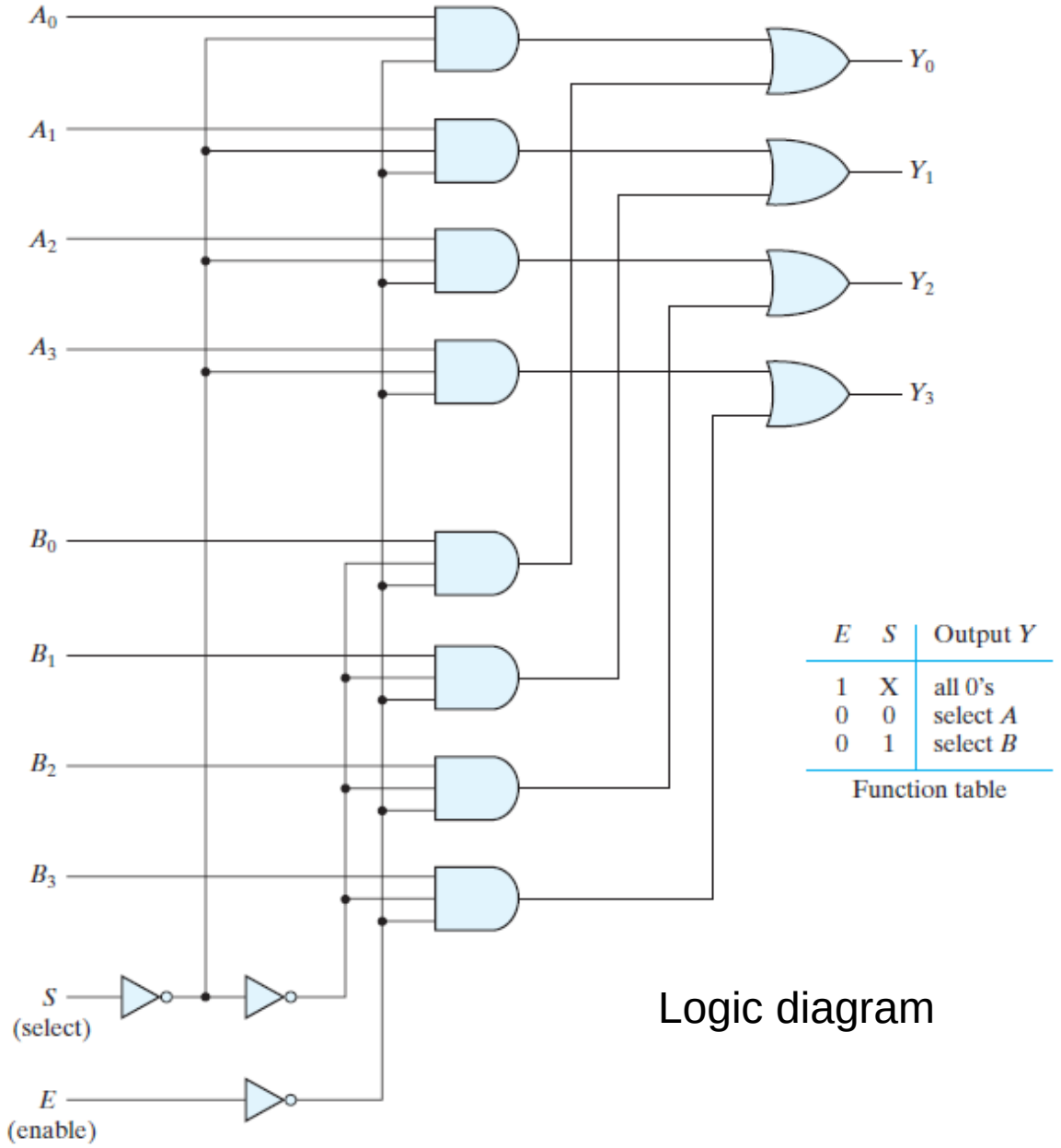


E	S	Output Y
1	X	all 0's
0	0	select A
0	1	select B

Function table

Multiplexer (MUX)





E	S	Output Y
1	X	all 0's
0	0	select A
0	1	select B

Function table

Logic diagram

Three-state Gates

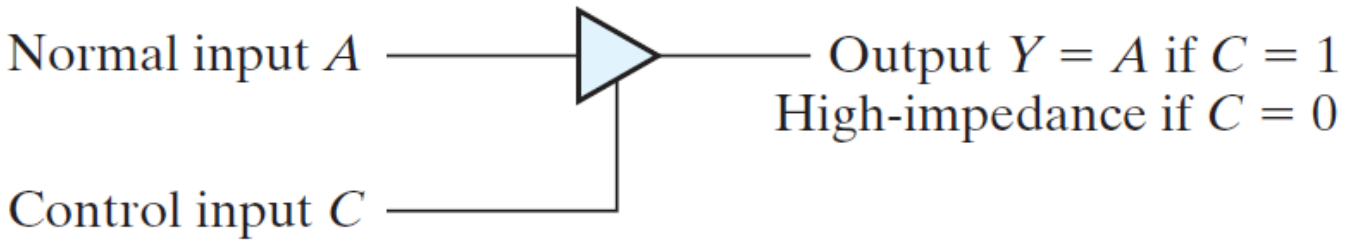
- A 3-state gate has an additional control input to generate 3 states at its output: {0,1, Z}

- When the control input is active, the gate output is enabled based on type of the gate
  - The 2 conventional states of 0 and 1 appear at the gate output

- The 3rd state is a high impedance (Z) state when the control input is not active
  - The gate output behaves like an open circuit

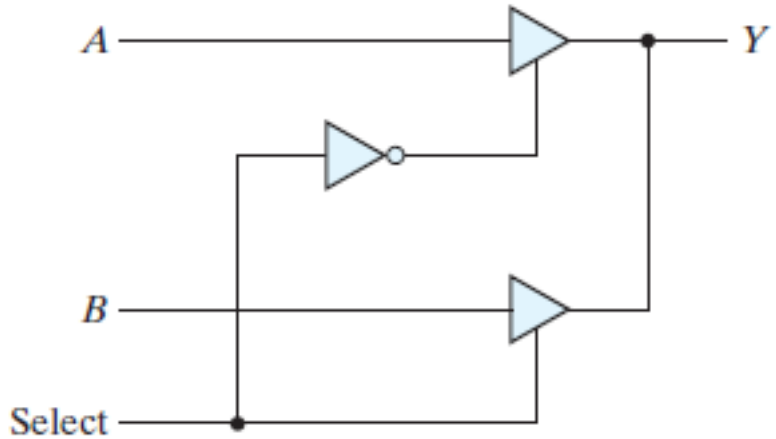
- A 3-state buffer:

tri-state.

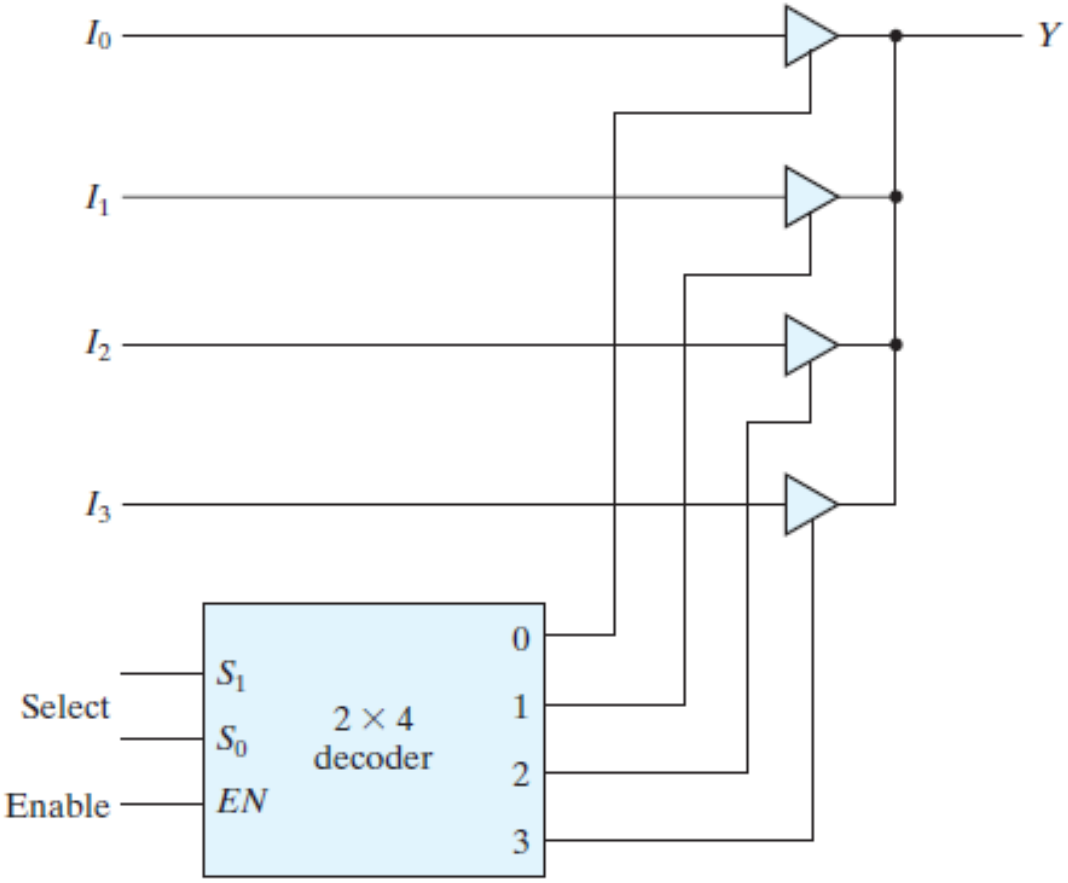


Three-state Gates

- The outputs of three-state gates can be connected together (not in regular gates) if the control input of at most one gate is active at any time.
- A MUX can be constructed with 3-state gates:



(a) 2-to-1-line mux

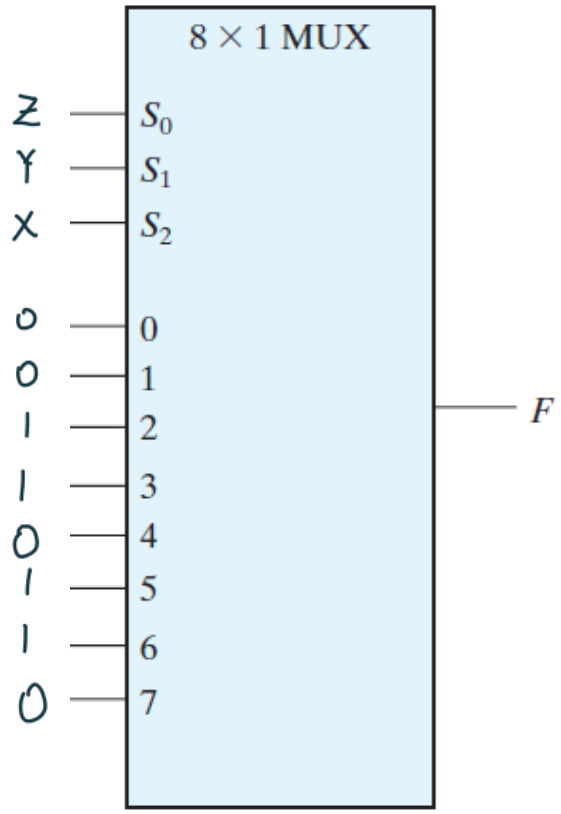


(b) 4-to-1-line mux

Boolean Function Implementation with MUXs

- A Boolean function with n variables can be easily implemented with a 2^nx1 MUX.
- Example 7: Design $F(x, y, z)=\sum(2, 3, 5, 6)$ with a 8x1 MUX (n=3)

x	y	z	F
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0

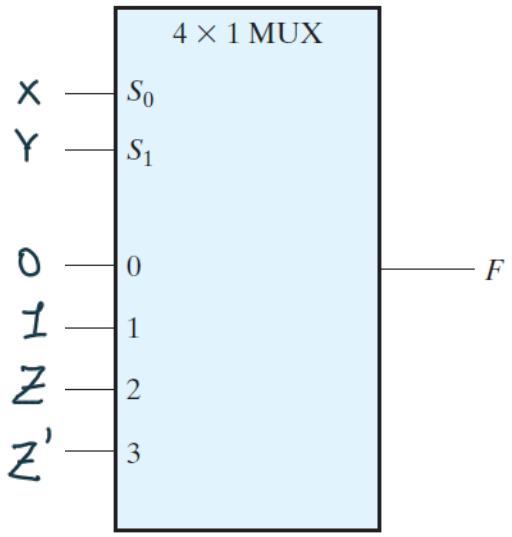


Boolean Function Implementation with MUXs

- A more efficient method is to use a 2^{n-1}x1 MUX:
  - The first *n-1* variables are connected to the selection inputs
  - Let the remaining variable be z. Then each input of MUX will be either 0, 1, z or z' based on the function.

- Example 8: Design $F(x, y, z)=\sum(2, 3, 5, 6)$ with a 4x1 MUX

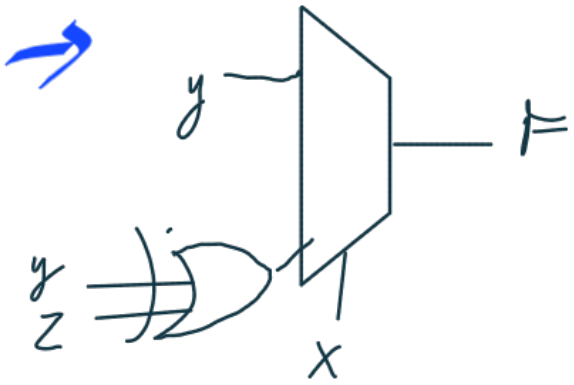
x	y	z	F
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0



Pay Attention

Boolean Function Implementation with MUXs

Example9: Design $F(x, y, z)=\sum(2, 3, 5, 6)$ using 2x1 MUX



Example 10: Implement $F(A,B,C)=A'C'+AB+AC$ using

(a) a 2-to-1 multiplexer and a few gates

(b) a 4-to-1 multiplexer

Always use the outer variable as the control variable.

Encoder

- An encoder is a digital circuit that performs the inverse operation of a decoder.
- An encoder has 2^n (or fewer) inputs and ***n*** outputs.
  - The output generates the binary code corresponding to the input value.

Table 4.7
Truth Table of an Octal-to-Binary Encoder

Inputs								Outputs		
D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7	x	y	z
1	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	0	0	1	0	0	0	0	0	1	1
0	0	0	0	1	0	0	0	1	0	0
0	0	0	0	0	1	0	0	1	0	1
0	0	0	0	0	0	1	0	1	1	0
0	0	0	0	0	0	0	1	1	1	1

Encoder

- This encoder can be implemented with three 4-input OR gates

$$X = D_1 + D_3 + D_5 + D_7$$
$$Y = D_2 + D_3 + D_6 + D_7$$
$$Z = D_4 + D_5 + D_6 + D_7$$

Table 4.7
Truth Table of an Octal-to-Binary Encoder

Inputs								Outputs		
D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7	x	y	z
1	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	0	0	1	0	0	0	0	0	1	1
0	0	0	0	1	0	0	0	1	0	0
0	0	0	0	0	1	0	0	1	0	1
0	0	0	0	0	0	1	0	1	1	0
0	0	0	0	0	0	0	1	1	1	1

- It has two limitations
  - Only one input can be active at any given time. If D_3 and D_6 are 1 simultaneously →What are xyz=?
  - All 0 outputs are generated if all D_is are zero. So, an additional output is required to distinguish it from D_0=1?

Priority Encoder

- A priority encoder is an encoder circuit that includes the priority function
- If two or more inputs are equal to 1 at the same time, the input code having the highest priority will be generated.
  - Determine the highest and lowest priority inputs?

Table 4.8
Truth Table of a Priority Encoder

Inputs				Outputs		
D_0	D_1	D_2	D_3	x	y	V
0	0	0	0	X	X	0
1	0	0	0	0	0	1
X	1	0	0	0	1	1
X	X	1	0	1	0	1
X	X	X	1	1	1	1

Priority Encoder

Table 4.8
Truth Table of a Priority Encoder

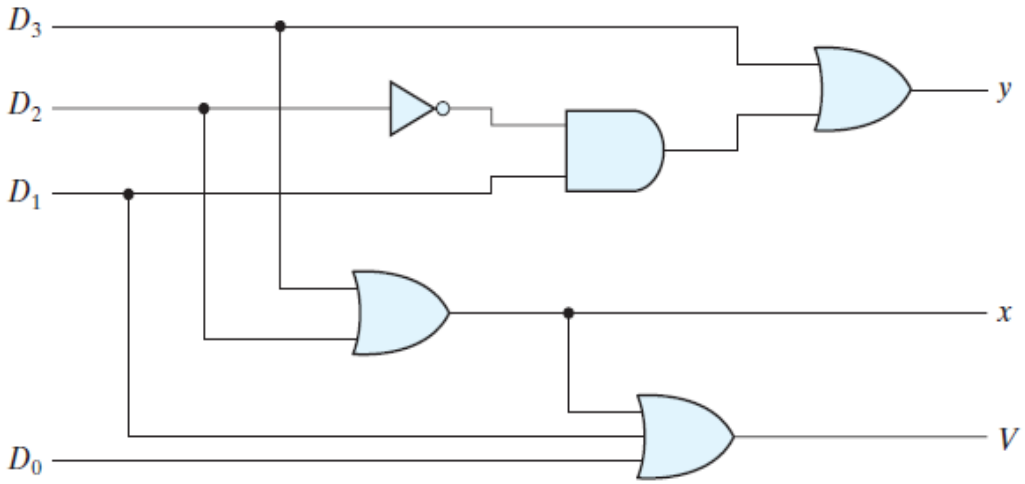
Inputs				Outputs		
D_0	D_1	D_2	D_3	x	y	V
0	0	0	0	X	X	0
1	0	0	0	0	0	1
X	1	0	0	0	1	1
X	X	1	0	1	0	1
X	X	X	1	1	1	1

Example 11: Design of Priority Encoder

$x=\Pi(8,4,12)$, $d=m_0$

$y=\Pi(2,6,8,10,14)$, $d=m_0$

- Logic Circuit Design of Priority Encoder
  - $y=D_3+D_1D'_2$
  - $x=D_2+D_3$
  - $V=D_o+D_1+D_2+D_3$

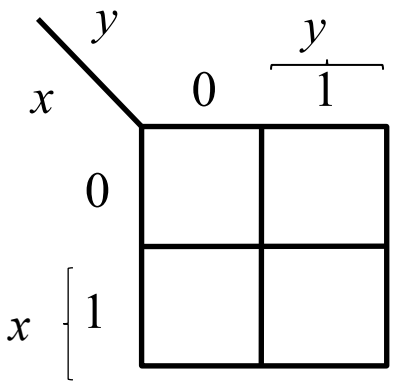


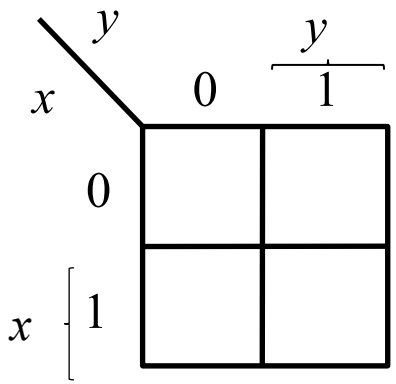
Half Adder (HA)

- A ***half adder*** is a combinational circuit that adds two bits. It has n=2 inputs (x and y) and m=2 outputs denoted by C (carry) and S (sum)

Example 12: Create a half adder

x	y	C	S
0	0		
0	1		
1	0		
1	1		

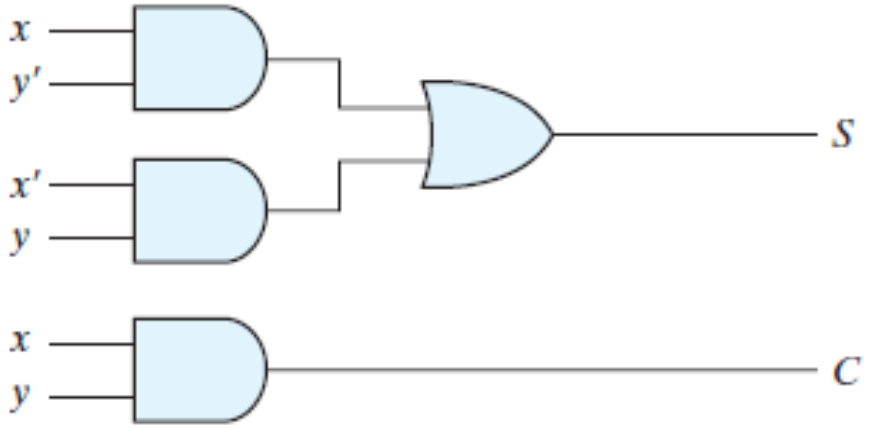




Half Adder (HA)

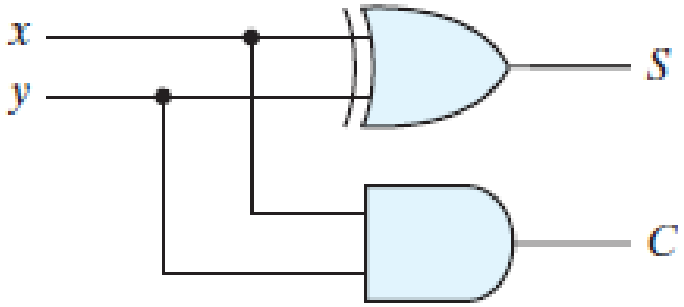
Half Adder (Implementations)

(a) Two-level of AND-OR



(a) $S = xy' + x'y$
$C = xy$

(b) Using an XOR gate



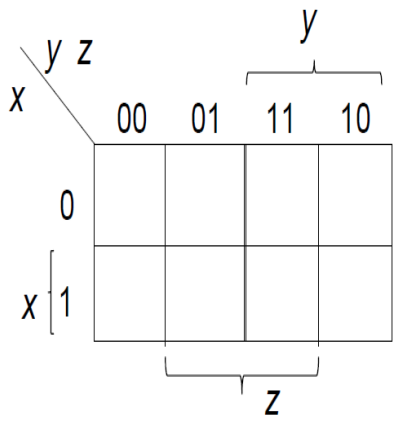
(b) $S = x \oplus y$
$C = xy$

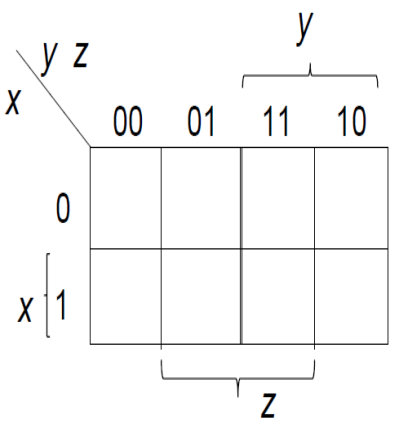
Full Adder (FA)

- A full-adder is a combinational circuit that adds three input bits. It consists of n=3 inputs (x, y and z) and m=2 outputs (C and S).

Example 13: Create a full adder

x	y	z	C	S
0	0	0		
0	0	1		
0	1	0		
0	1	1		
1	0	0		
1	0	1		
1	1	0		
1	1	1		



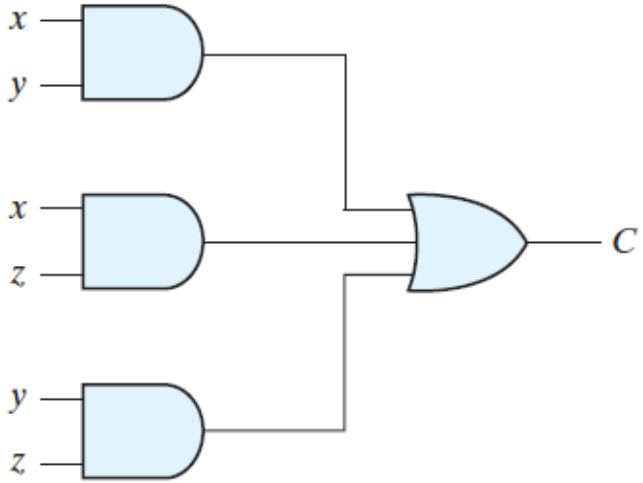


Full Adder (FA)

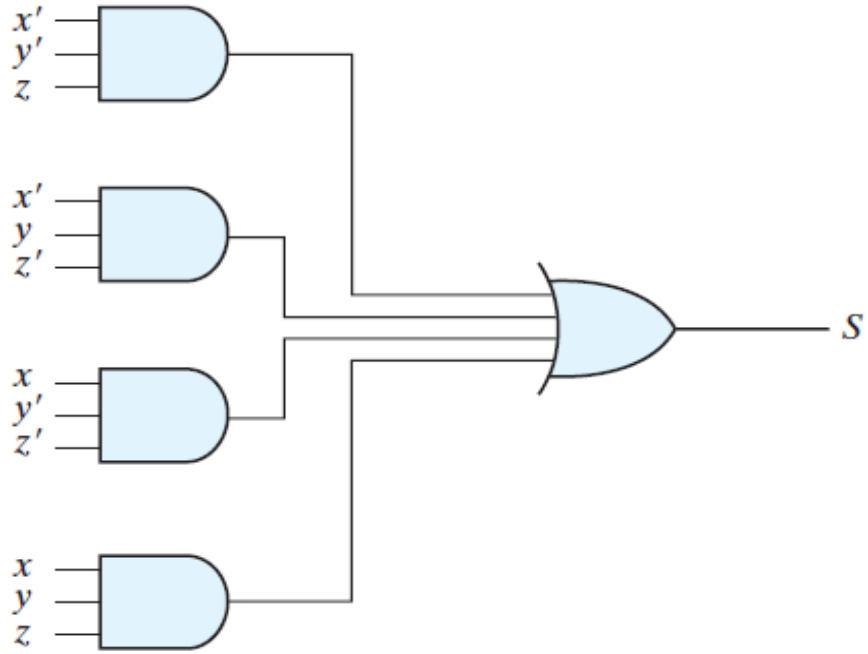
Full Adder (Implementations)

- Two-level of AND-OR implementation

$C = xy + xz + yz$

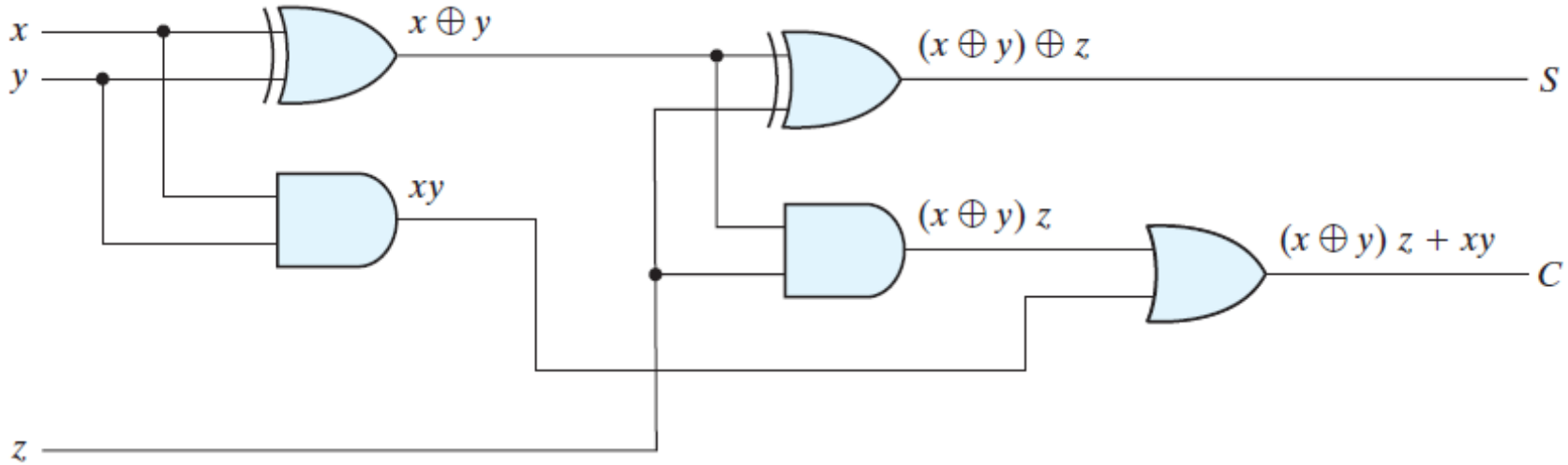


$S = x'y'z + x'yz' + xy'z' + xyz$



Full Adder (FA)

- Implementation using 2 HA and an OR gate



- Prove:

$(x \oplus y)\, z + xy$ =xy+xz+yz

$(x \oplus y) \oplus z$ =x'y'z+x'yz'+xy'z'+xyz

Binary Adder

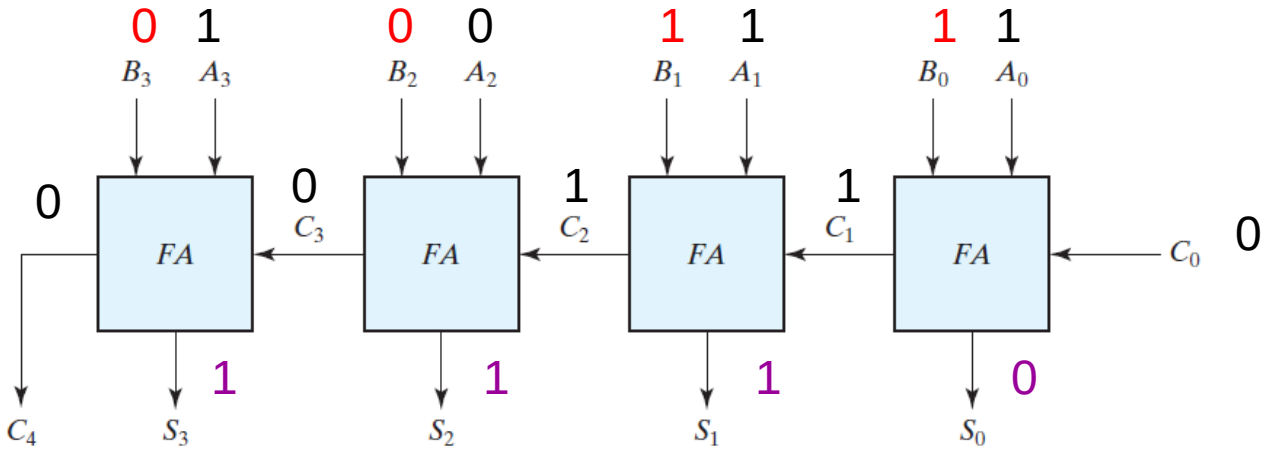
- A binary adder is a digital circuit that produces the arithmetic sum of two binary numbers
- It adds two *n*-bit binary numbers $A=(A_{n-1}...A_1A_0)_2$ and $B=(B_{n-1}...B_1B_0)_2$ using *n* full adders (FA)s.
  - *In a FA:* $x=A_i$, $y=B_i$, $z=C_i$, $S=S_i$, $C=C_{i+1}$, $0 \le i \le n-1$

at 0 nano sec.

lets assume each adder takes 10 nano sec.

Example 14: Adding A=1011, B=0011, Co = 0

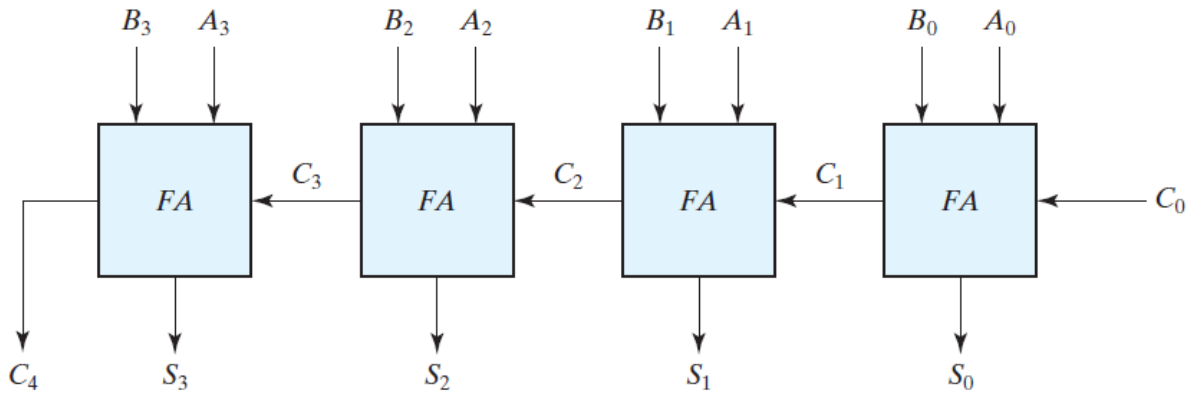
→ $C_4=0$, S=1110



The idea is that @ 0 n.s, all the adders are still adding something. You'd have to wait until 40 ns have elapsed to get the right answer.

Binary Adder

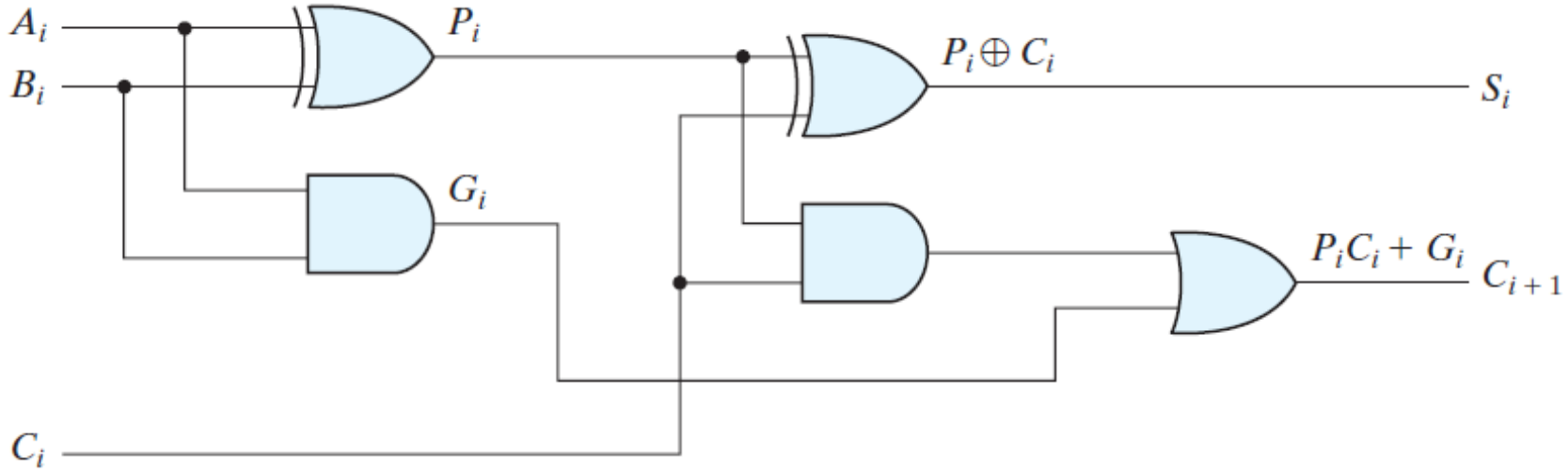
- Assume that FAs are implemented in 2 levels, what is the number of gate levels of an n-bit adder?
- The total propagation time is equal to the propagation delay of a typical gate times the number of gate levels in the circuit.



- The longest propagation delay time in an adder is the time it takes the carry to propagate through the FAs.
- It limits the speed of the adder

How to Design a Fast Adder → uses more gates to avoid delays.

- Let us define $P_i = A_i \oplus B_i$ and $G_i = A_iB_i$ signals as the outputs of the first HA of the i^{th} FA
- Then, $C_{i+1} = G_i + P_iC_i$, $S_i = P_i \oplus C_i$ $\quad 0 \le i \le n-1$



- To reduce the carry propagation time in the adder, a carry lookahead generator is used

Carry Lookahead Generator

- We derive the Boolean function for C_{i+1} as a function of only C_0 (not C_i) and other signals

$C_{i+1}=G_i+P_iC_i$ $(i=0) \rightarrow C_1=G_0+P_0C_0$

$(i=1) \rightarrow C_2=G_1+P_1C_1$

$=G_1+P_1(G_0+P_0C_0)$

$=G_1+P_1G_0+P_1P_0C_0$

$(i=2) \rightarrow C_3=G_2+P_2C_2$

$=G_2+P_2(G_1+P_1G_0+P_1P_0C_0)$

$=G_2+P_2G_1+P_2P_1G_0+P_2P_1P_0C_0$

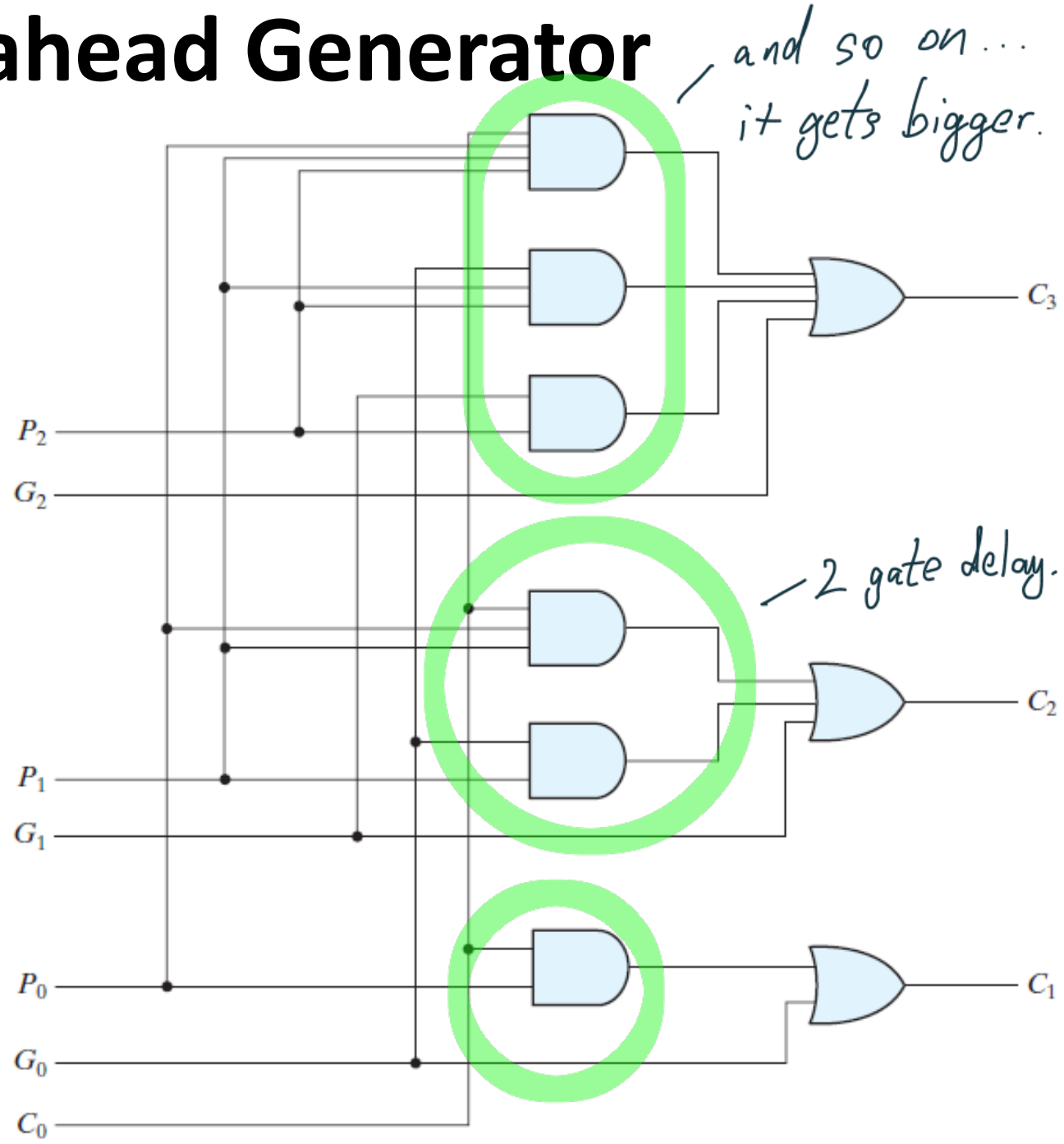
- The three carry functions are expressed in SOP.
- Then, we can implement them in 2-level AND-OR implementation

Carry Lookahead Generator

$C_3=G_2+P_2G_1+P_2P_1G_0+P_2P_1P_0C_0$

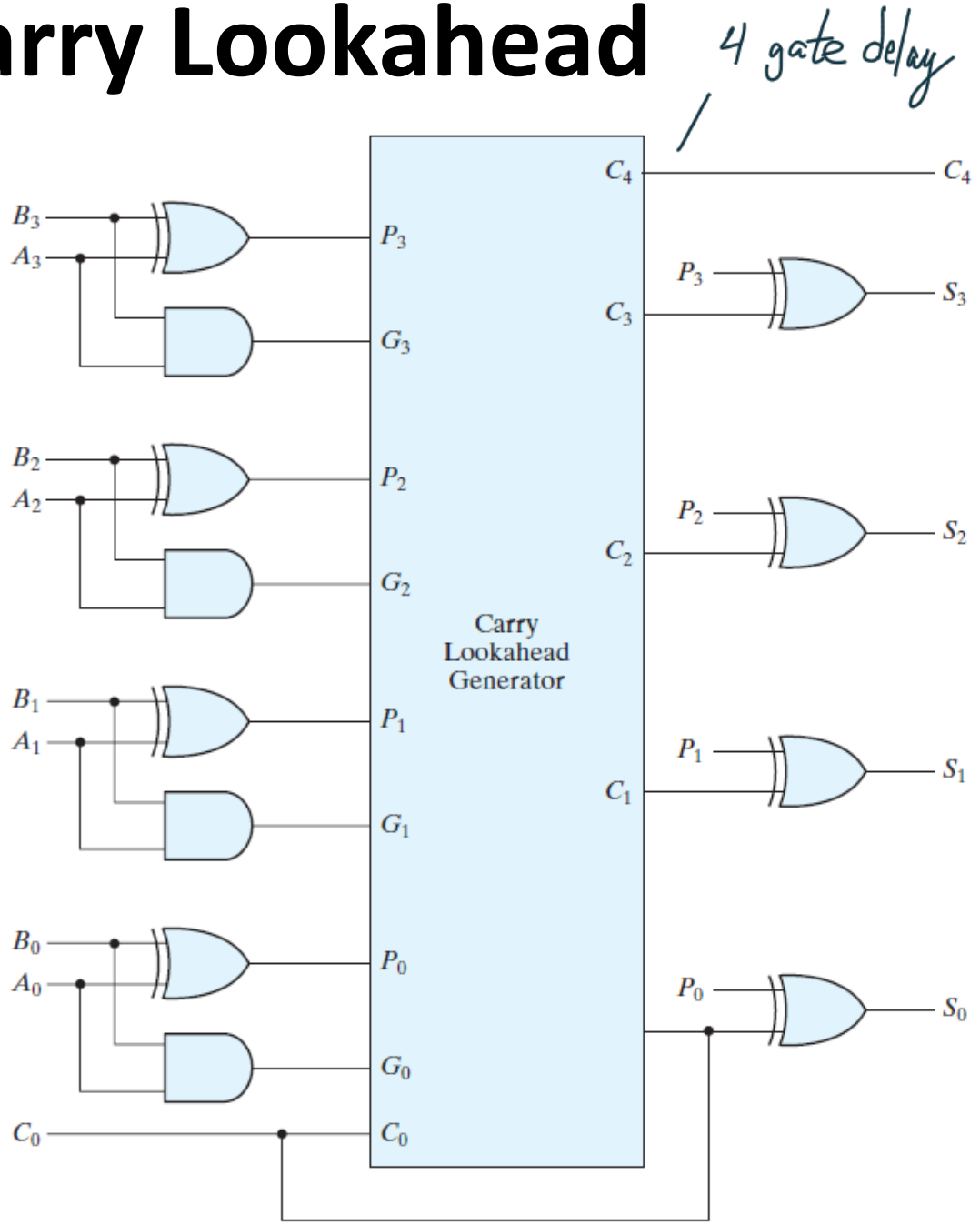
$C_2=G_1+P_1G_0+P_1P_0C_0$

$C_1=G_0+P_0C_0$



Binary Adder using Carry Lookahead

- Recall: $P_i = A_i \quad B_i$, $G_i = A_i B_i$ and $S_i = P_i \oplus C_i$
- *How many levels does it have?*
- The propagation delay is independent of *n* (the number of bits)

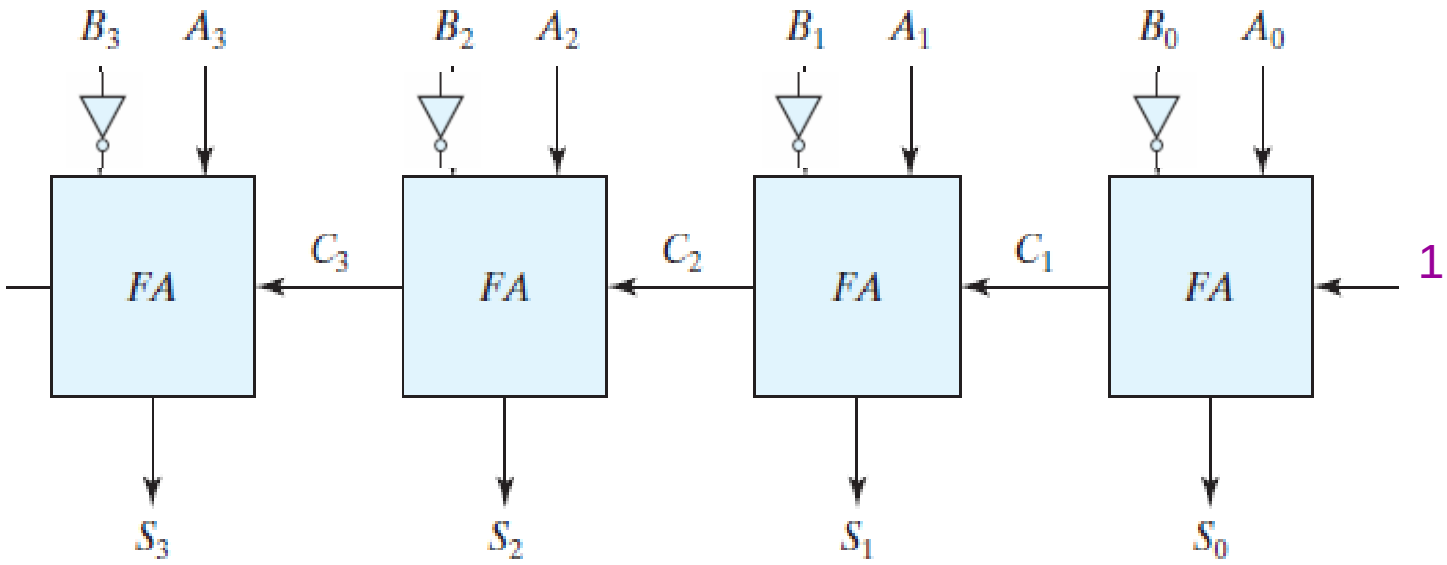


Unsigned Binary Subtractor

- The subtraction of unsigned binary numbers can be done by means of complements
- Recall: the subtraction A - B can be done by adding A to the 2's complement of B
- 2's complement of B = 1's complement of B + 1

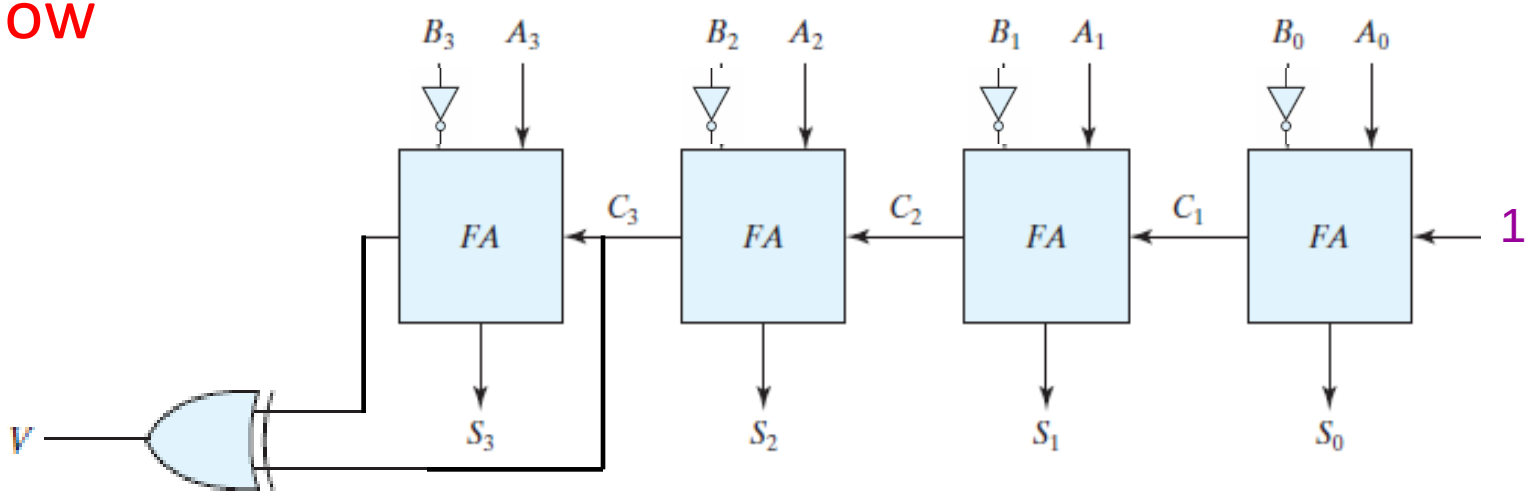
$$= (B'_{n-1}\ldots B'_1 B'_0) + (0\ldots 01)_2$$

- $A \geq B \rightarrow A-B=S$
- $A < B \rightarrow A-B=-2\text{'scomp.}(S)$



Signed Binary Subtractor

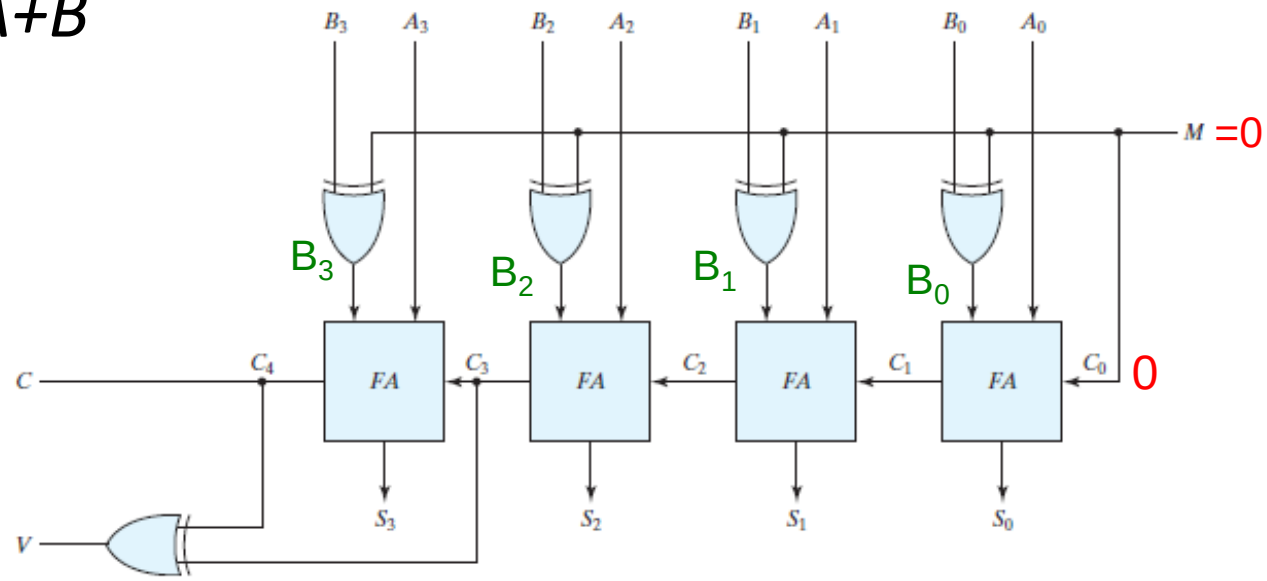
- Let A and B be signed numbers represented in the signed 2's complement number system
- The output of this circuit (S) generates A-B if there is no overflow (V=0)
  If V=1→ there is an overflow



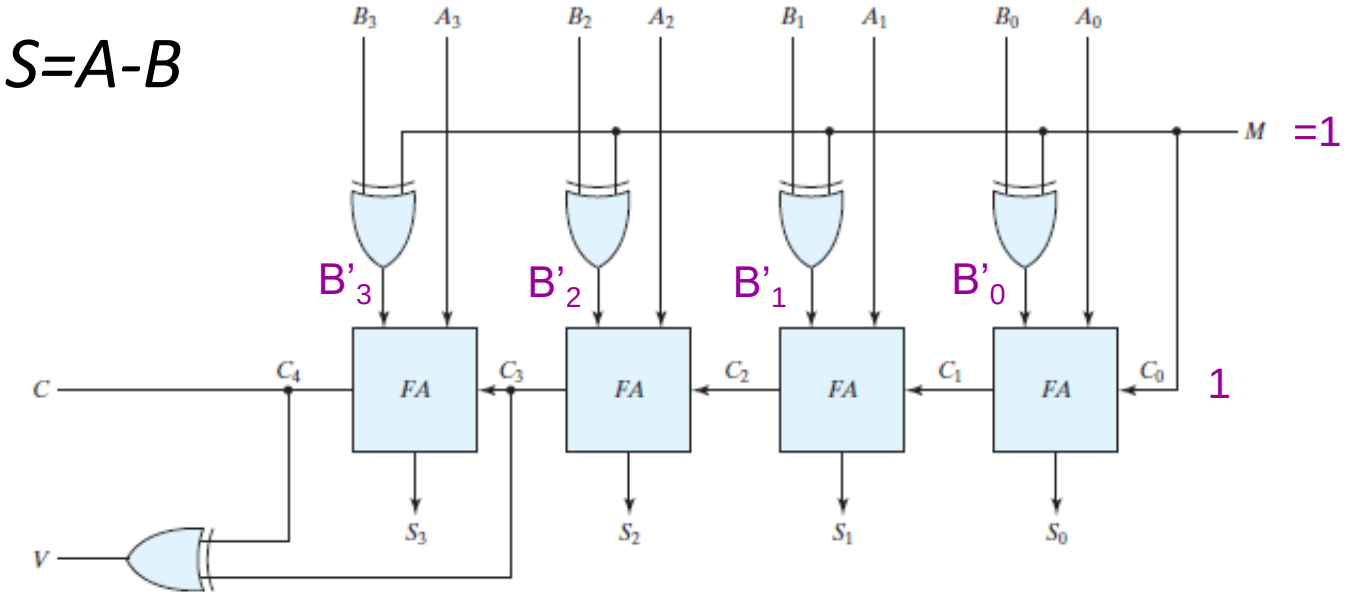
- So, this circuit can also be used for unsigned subtraction

Adder-Subtractor

- If *M=0* (Adder) $B_i \oplus 0 = B_i \rightarrow S=A+B$

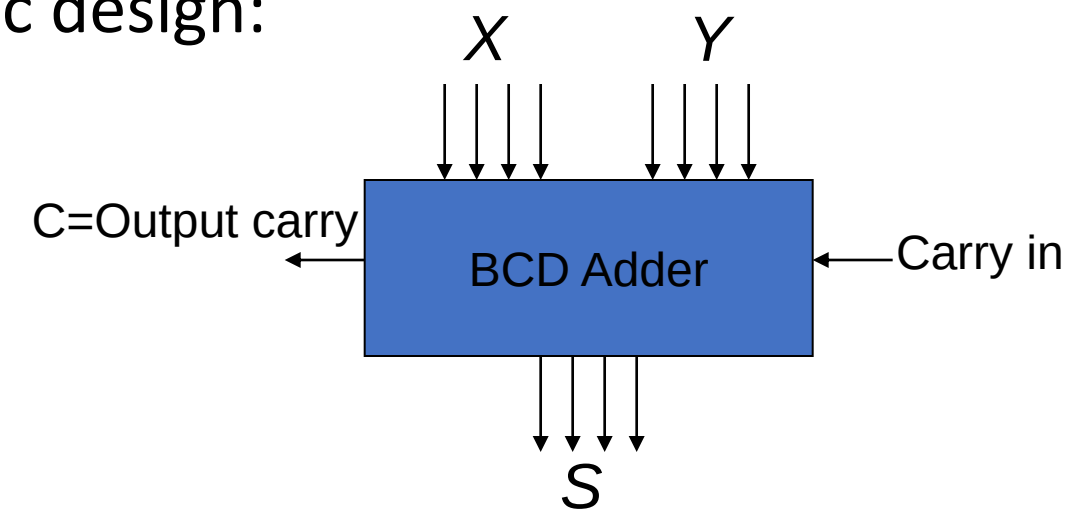


- *If M=1 (subtractor)* $B_i \oplus 1 = B'_i \rightarrow S=A-B$



BCD Adder

- A BCD adder is a combinational circuit that adds two decimal numbers *X* and *Y*, $0 \le X,Y \le 9$, with a "Carry in" and generates the decimal sum S, $0 \le S \le 9$, with an Output carry C.
- How many inputs and outputs does it have?
- A systematic design:

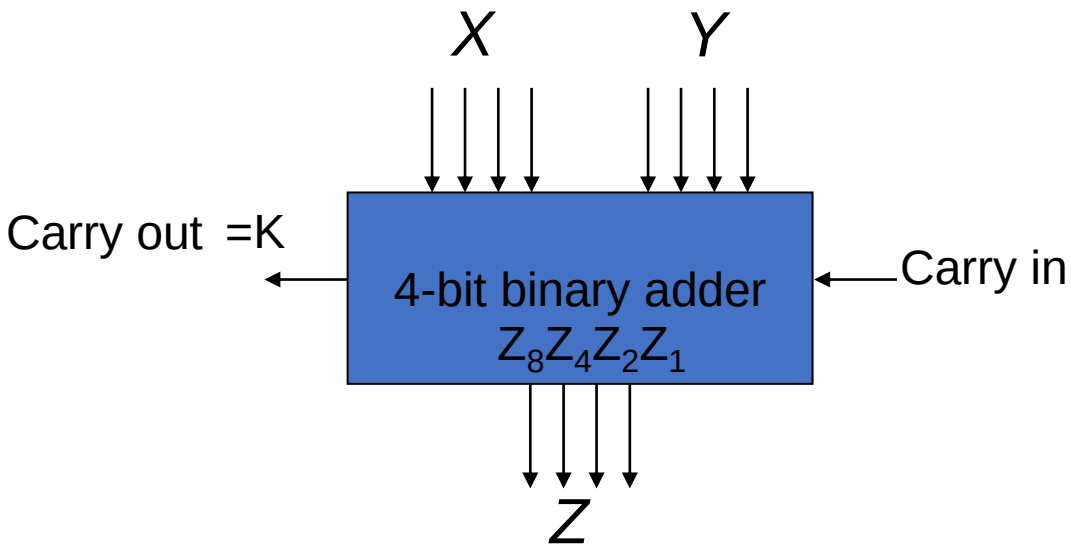


- 5 output functions with 9 input variables

BCD Adder

- We want to use binary adders and an additional logic circuit to implement it
- First, add *X*, *Y* and carry in using a 4-bit binary adder
- The output of this addition produces 5-bit which range from:

$$
\begin{array}{r}
0 \le X \le 9 \\
0 \le Y \le 9 \\
+\quad 0 \le Carry\ in \le 1 \\
\hline
0 \le X+Y+Carry\ in \le 19
\end{array}
$$



- The binary addition of "X+Y+carry in" is represented by: $KZ_8Z_4Z_2Z_1$

BCD Adder--Addition Rules

- If the result of binary addition ≥ 10
  - The Output carry C=1. It will be added to the next decimal digits
  - In order to get a BCD code at S, it should be added with 6:

$$S_8S_4S_2S_1 = Z_8Z_4Z_2Z_1 + 0110$$

- Otherwise, the result of Z are in the BCD format
  - Output carry=C=0

$$S_8S_4S_2S_1 = Z_8Z_4Z_2Z_1 + 0000$$

- The above two cases can be combined as

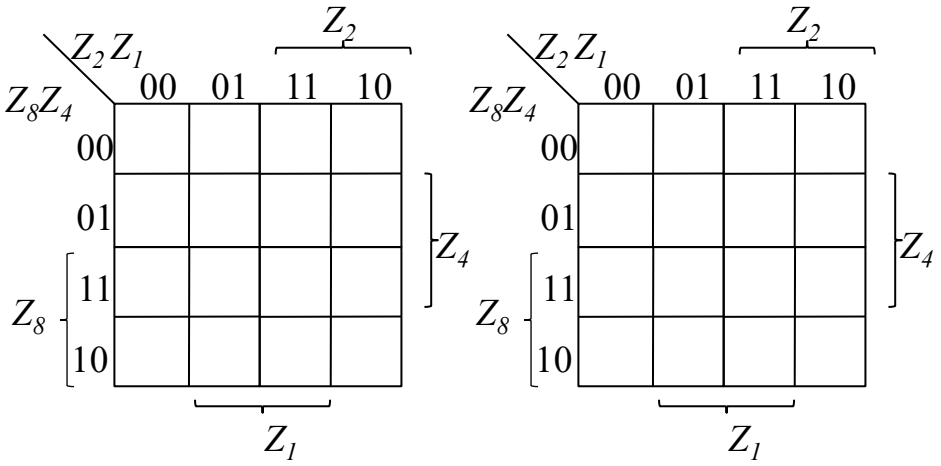
$$S_8S_4S_2S_1 = Z_8Z_4Z_2Z_1 + 0CC0$$

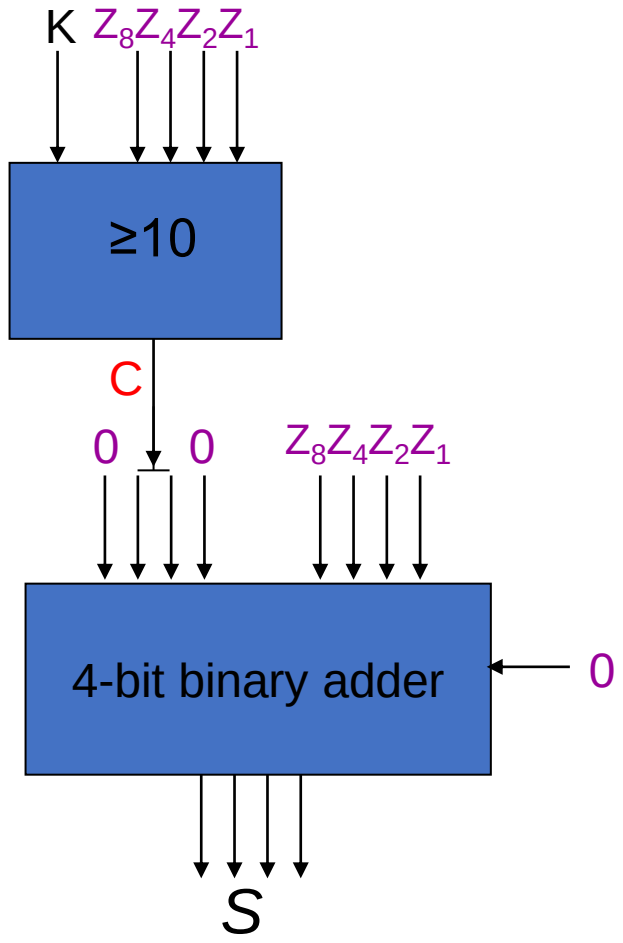
- It can be implemented with another 4-bit adder and a combinational logic that generates C

BCD Adder

Example 15: Design C (the combinational logic ≥10):

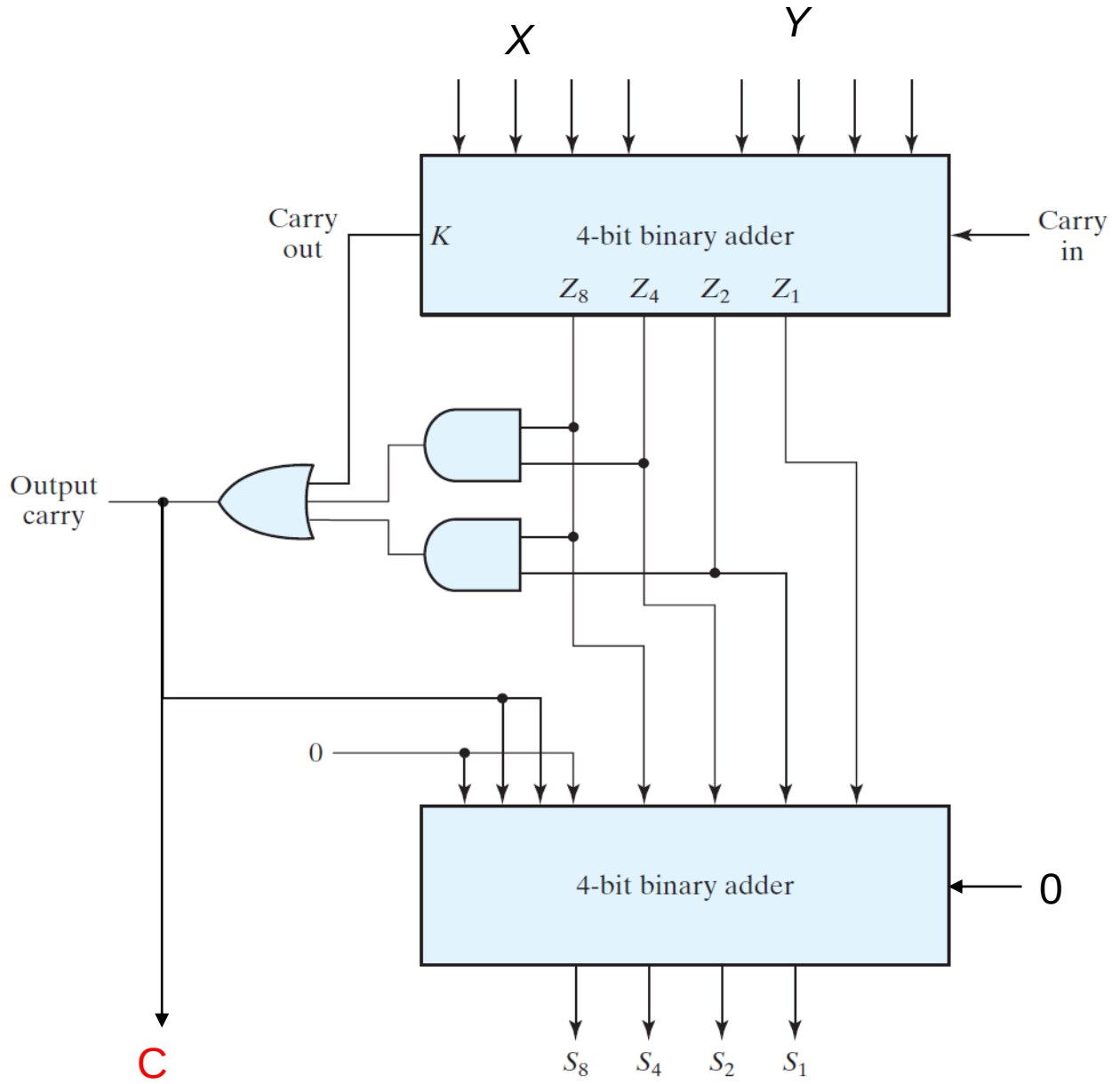
$C(K,Z_8,Z_4,Z_2,Z_1)=\sum(10,11,12,\ldots,18,19)+d(20, 21,\ldots,31)$





BCD Adder

- Combining all designs:

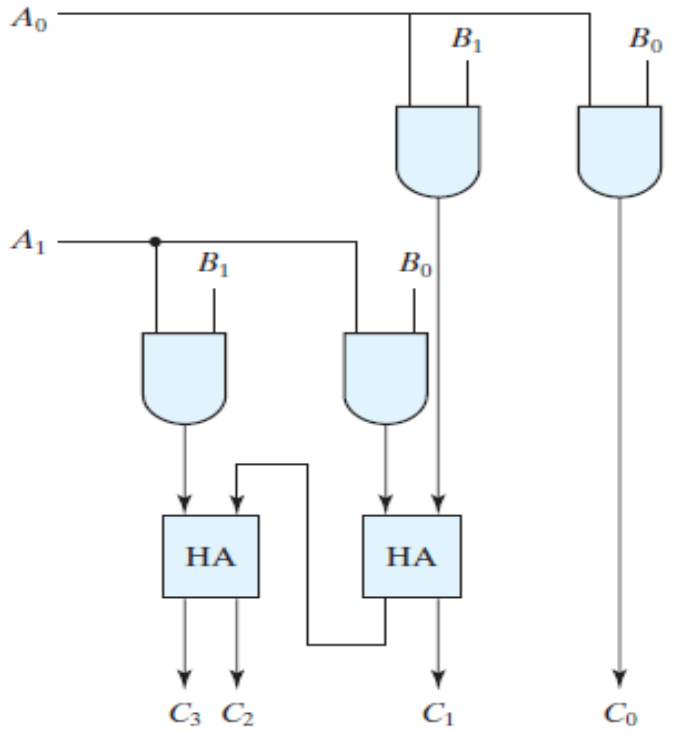


Binary Multiplier

- An n-bit by m-bit binary multiplier is a combinational circuit that multiplies two binary numbers and generates an m+n-bit product

- Example: a 2x2 multiplier

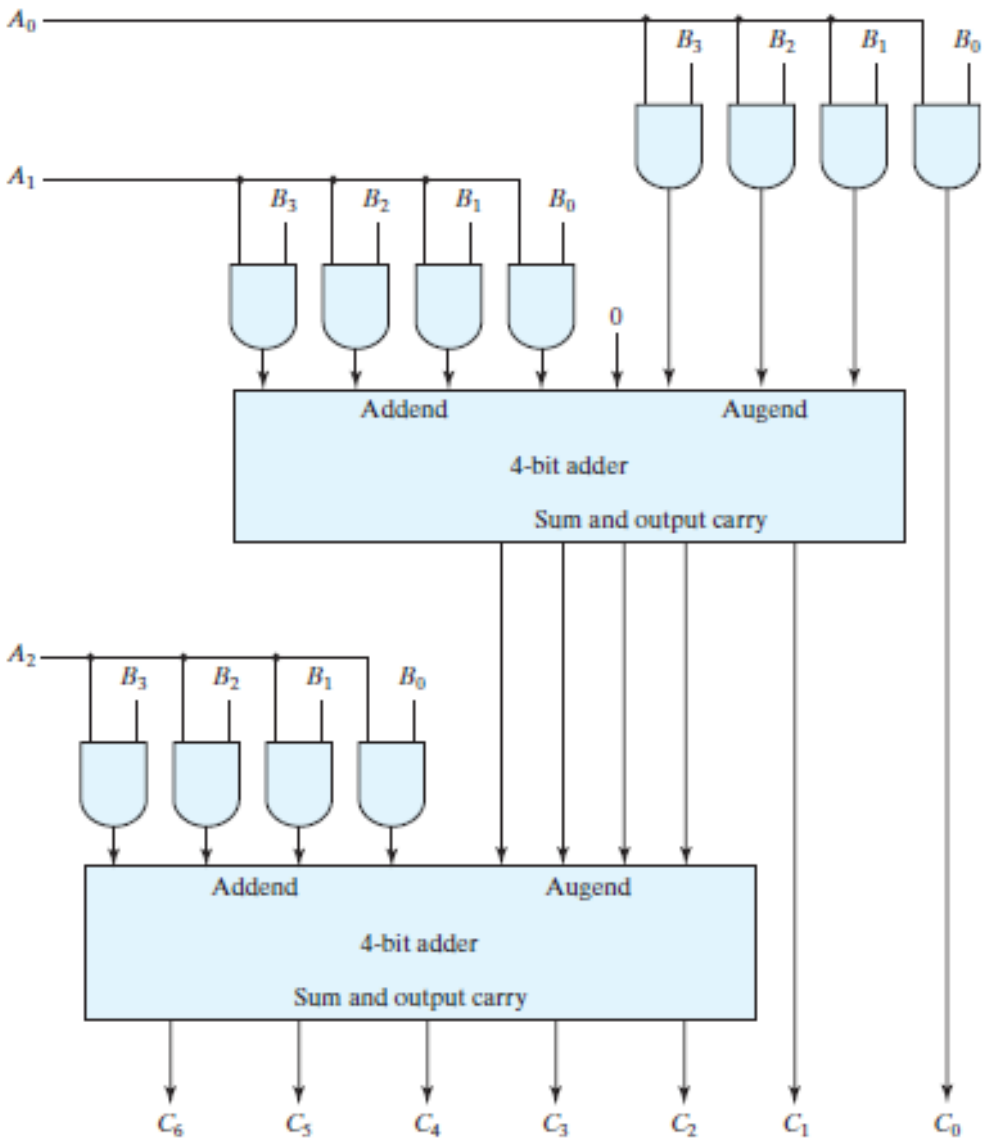
$$
\begin{array}{cccc}
 & & B_1 & B_0 \\
 & & A_1 & A_0 \\
\hline
 & & A_0B_1 & A_0B_0 \\
 & A_1B_1 & A_1B_0 & \\
\hline
C_3 & C_2 & C_1 & C_0
\end{array}
$$



Binary Multiplier

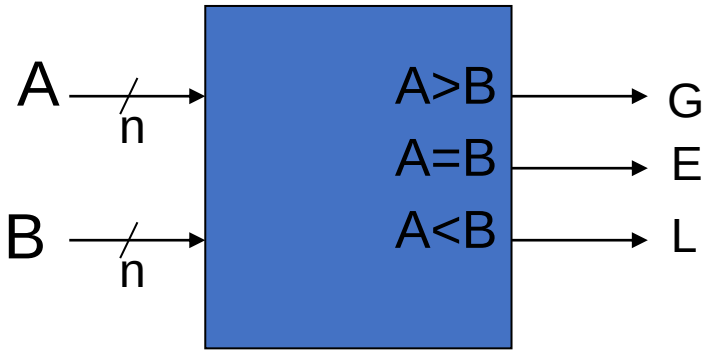
- Example16: Design a multiplier of $(A_2A_1A_0)x(B_3B_2B_1B_0)= (C_6C_5C_4C_3C_2C_1C_0)$

$$\begin{array}{r} B_3\ B_2\ B_1\ B_0 \\ A_2\ A_1\ A_0 \\ \hline \end{array}$$



Magnitude Comparator

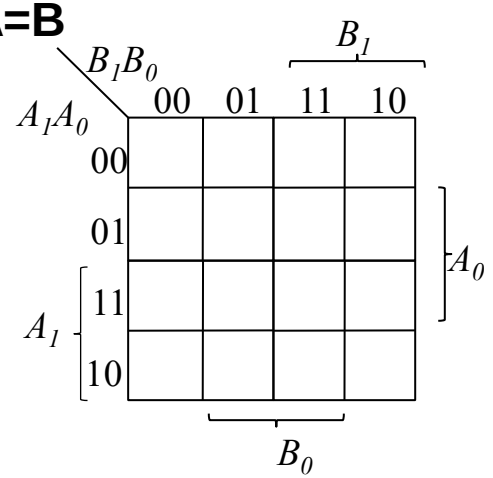
- A magnitude comparator is a combinational circuit that compares two numbers, A and B, and determines their relative magnitudes.
- The outcome of the comparison is specified by three binary variables (G, E, L) that indicate whether (A>B), (A=B), or (A<B).
- It requires a truth table with 2n inputs



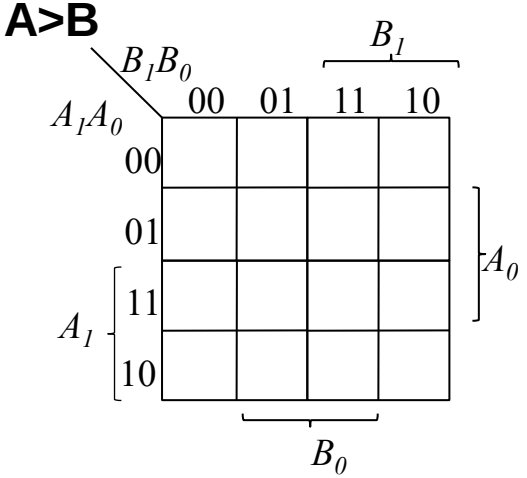
Magnitude Comparator

Example 17: Using K-maps, design a magnitude comparator for 2-bit numbers A and B (n=2) with outputs (G, E, L) that indicate whether (A>B), (A=B), or (A<B)

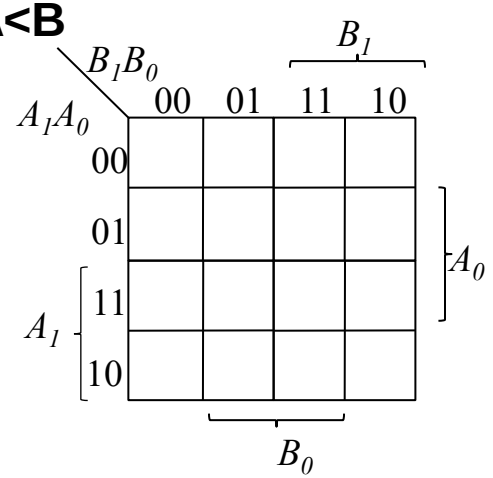
A=B



A>B



A<B



Magnitude Comparator

- It becomes hard with ***n*** > 2 if a truth table is used

Example 18: Design a 4-bit comparator without using truth table

- Let us define a variable x_i which is 1 if the pair of bits in position *i* are equal, i.e., $A_i=B_i$
- To obtain E (A=B) → $(A_3A_2A_1A_0)=(B_3B_2B_1B_0)$

  $A_3=B_3$ and $A_2=B_2$ and $A_1=B_1$ and $A_0=B_0$

A_i	B_i	x_i
0	0	
0	1	
1	0	
1	1	

Magnitude Comparator

- Let us define another variable g_i which becomes 1 if $A_i > B_i$

- To obtain G (A>B) → $(A_3A_2A_1A_0) > (B_3B_2B_1B_0)$

  $A_3 > B_3$ or

  $(A_3 = B_3 \& A_2 > B_2)$ or

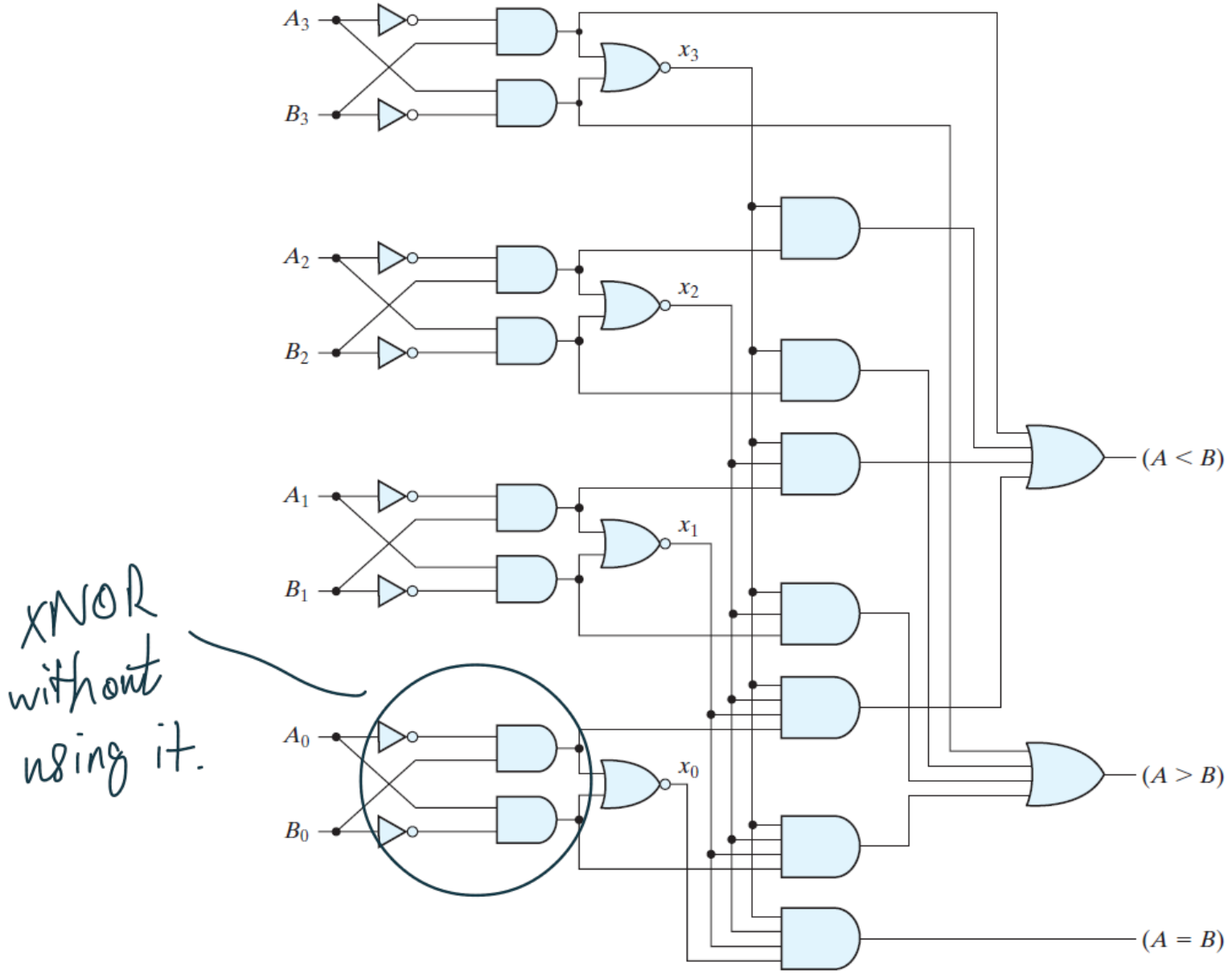
  $(A_3 = B_3 \& A_2 = B_2 \& A_1 > B_1)$ or

  $(A_3 = B_3 \& A_2 = B_2 \& A_1 = B_1 \& A_0 > B_0)$

A_i	B_i	g_i
0	0	
0	1	
1	0	
1	1	

- To obtain L (A<B) → $(A_3A_2A_1A_0) < (B_3B_2B_1B_0)$
  - Swapping all A_is and B_is from the function G
- It results in multi-level implementation

Magnitude Comparator



XNOR without using it.

Examples

- Example 19: Using a decoder and external gates, design the combinational circuit defined by $F_1=(y'+x)z$, $F_2=y'z'+xy'+yz'$, $F_3=(x'+y)z$

- Example 20: Implement $F(A,B,C,D)=\Sigma(1,2,4,7,8,9,10,11,13,15)$ with a 4x1 MUX and external gates. Connect A and B to the selection lines.

- Example 21: Implement the combinational circuit S(x, y, z) = $\Sigma(1, 2, 4, 7)$, C(x, y, z) = $\Sigma(3, 5, 6, 7)$ using a) **active-low decoders** b) implement using active-high decoders

- Example 22: Design a combinational circuit that generates the 9's complement of a Gray-code digit.