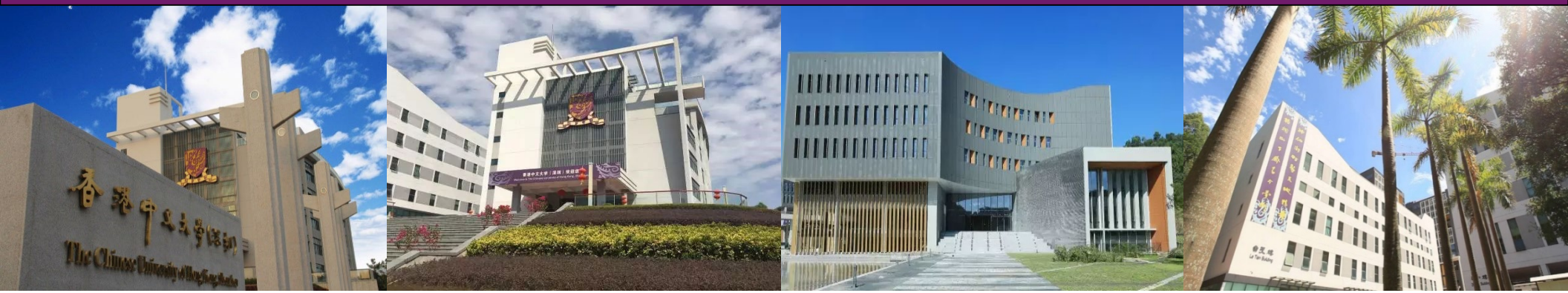
Last update: 2023/2/27

EIE 2050 Digital Logic and Systems

Chapter 6 : Functions of Combinational Logic

Simon Pun, Ph.D.



Last Week

- Basic Combinational Logic Circuits
  - AND-OR Logic
  - AND-OR-Invert Logic
  - Exclusive-OR Logic $X = A\bar{B} + \bar{A}B = A \oplus B$
  - Exclusive-NOR Logic $X = \bar{A}\bar{B} + AB$
- The Universal Property of NAND Gates
  - $\overline{AB} = \bar{A} + \bar{B}$ $\overline{A + B} = \bar{A}\bar{B}$
  - Combinations of NAND gates can function as NOT, AND, OR, and NOR
  - Combinations of NOR gates can function as NOT, AND, OR, and NAND.
- Dual Symbols :
  - NAND Logic Diagrams Using Dual Symbols : NAND and Negative-OR;
  - NOR Logic Diagrams Using Dual Symbols : NOR and Negative-AND
- Troubleshooting: Pulse Waveform Operation

Half Adders

- Input: two binary digits;
- Output: two binary digits: a sum bit and a carry bit.

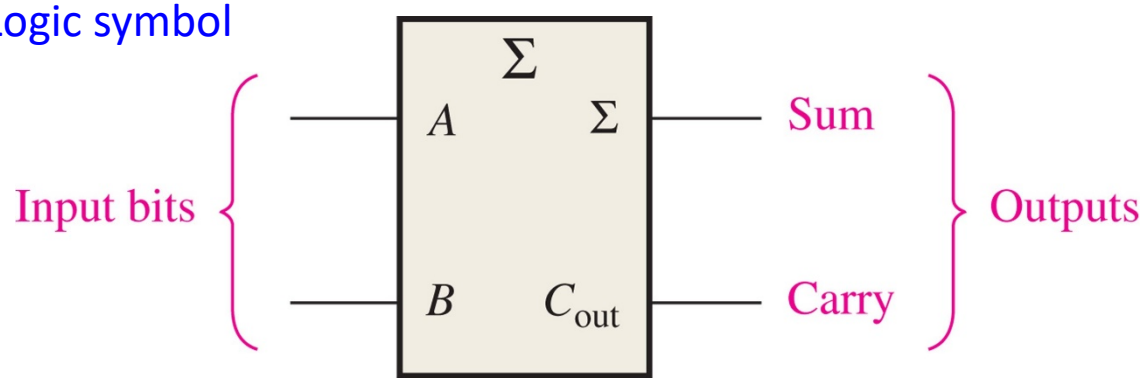
TABLE 6–1

Half-adder truth table.

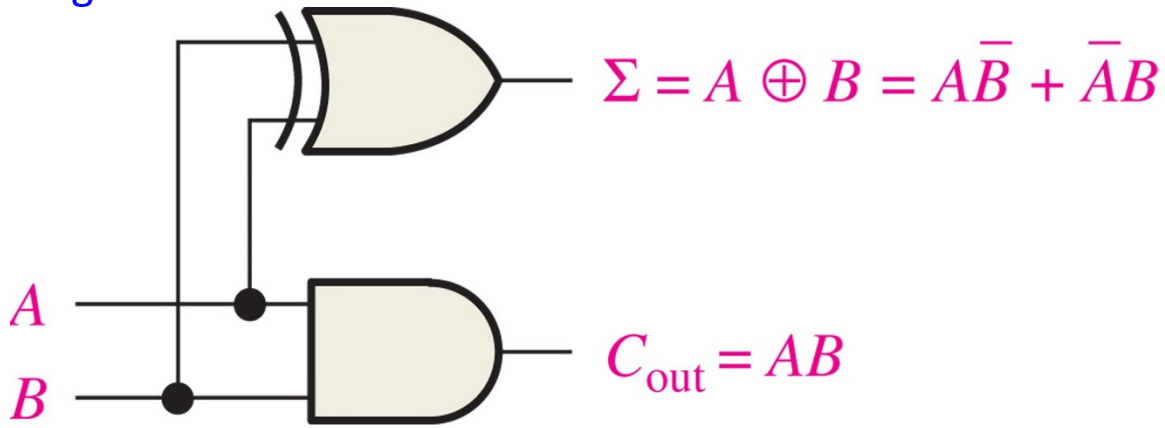
A	B	C_{out}	Σ
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Σ = sum
C_{out} = output carry
A and B = input variables (operands)

Logic symbol



Logic diagram



Full Adders

- Input: two binary digits and **an input carry**
- Output: two binary digits: a sum bit and a carry bit.

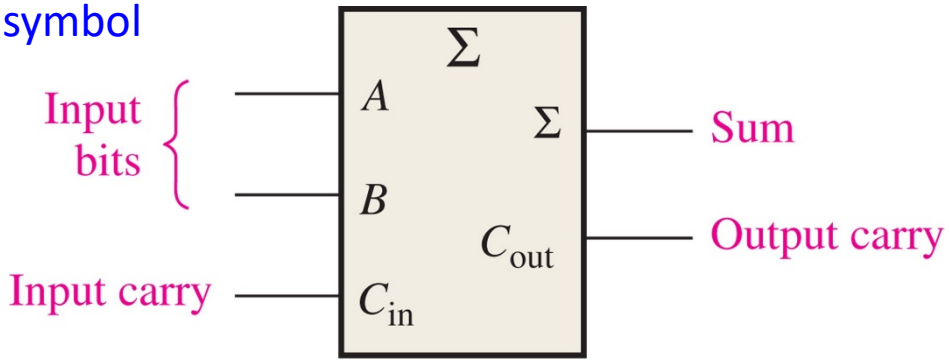
TABLE 6–2

Full-adder truth table.

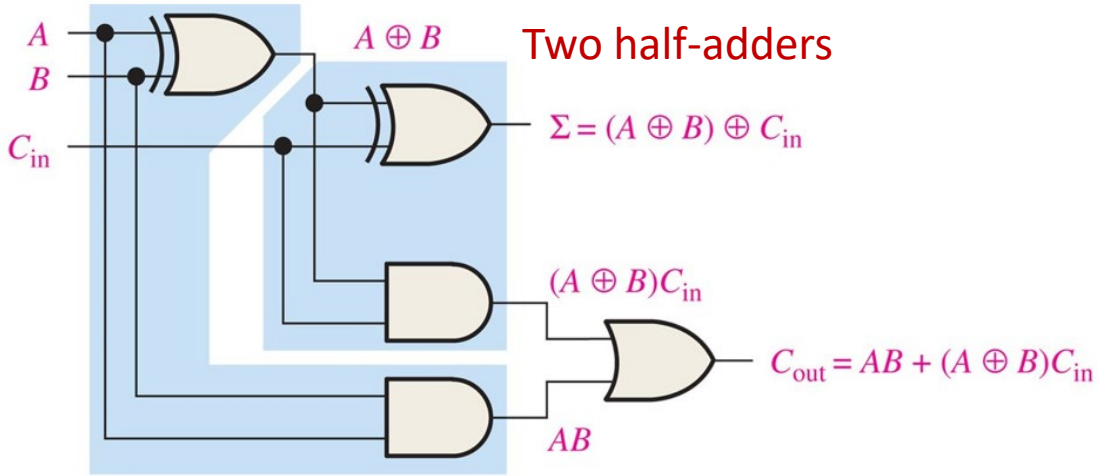
A	B	C_{in}	C_{out}	Σ
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

C_{in} = input carry, sometimes designated as CI
C_{out} = output carry, sometimes designated as CO
Σ = sum
A and B = input variables (operands)

Logic symbol

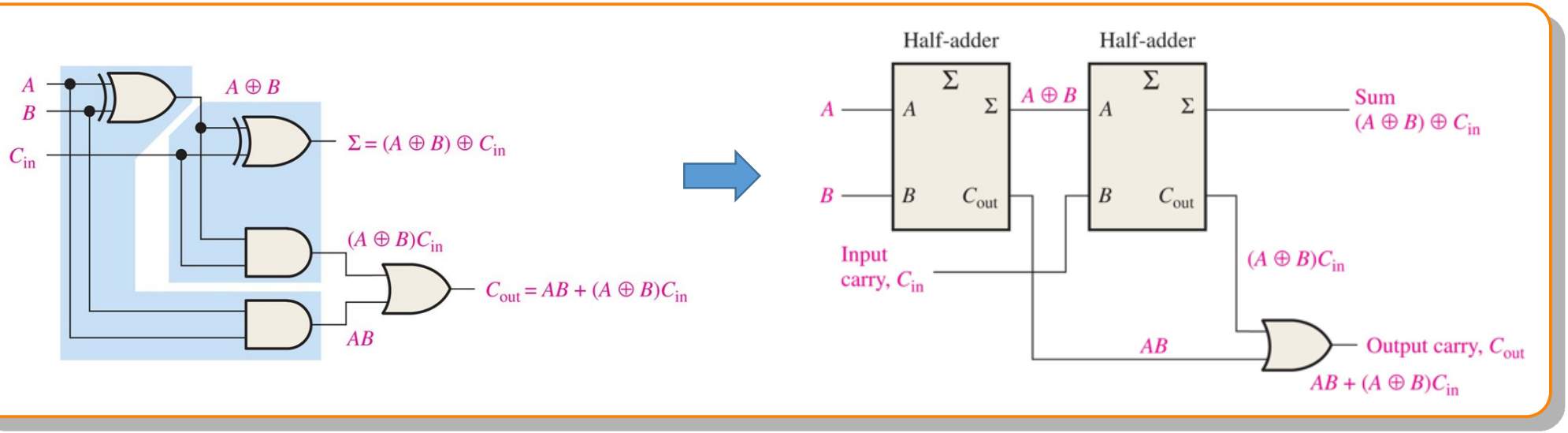


Logic diagram

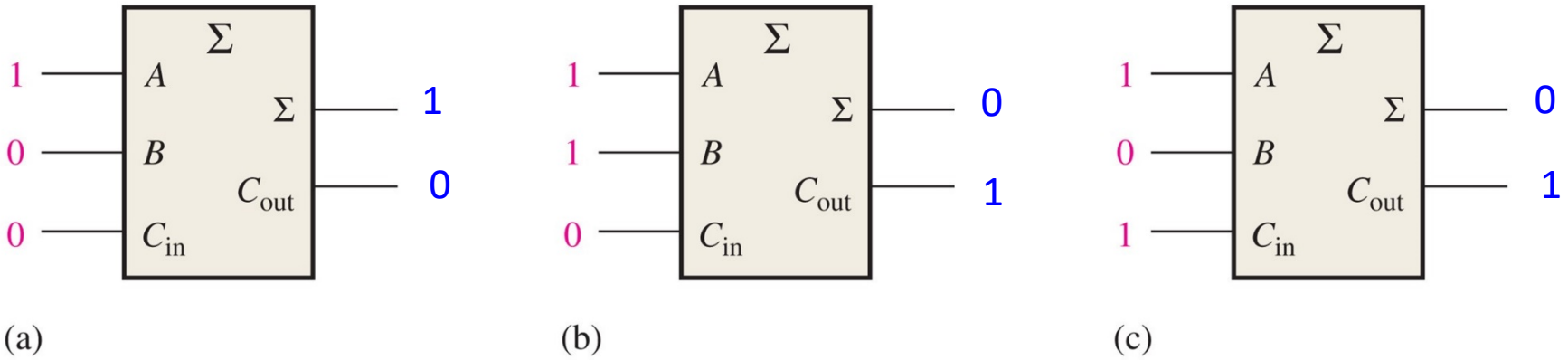


Two half-adders

Full-Adder Implemented with Half-adders.



EXAMPLE 6－1



Parallel Binary Adders

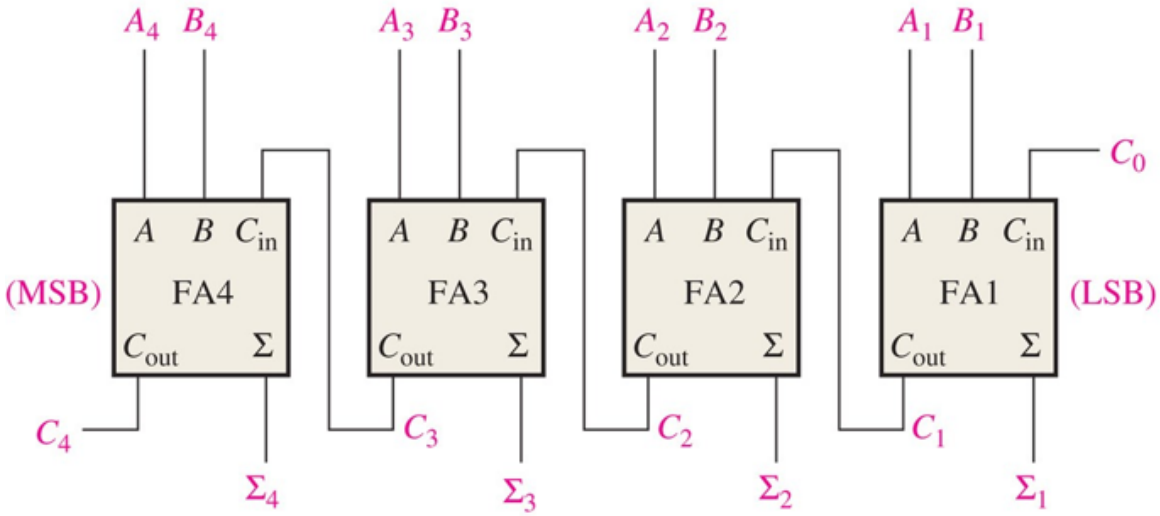
- Two or more full-adders are connected to form parallel binary adders.
- Using the following 4-bit parallel adder as an example, $A_4A_3A_2A_1 = 1100$ and $B_4B_3B_2B_1 = 1100$,

$\Sigma_1=0, C_1=0$; $\Sigma_2=0, C_2=0$; $\Sigma_3=0, C_3=1$; $\Sigma_4=1, C_4=1$

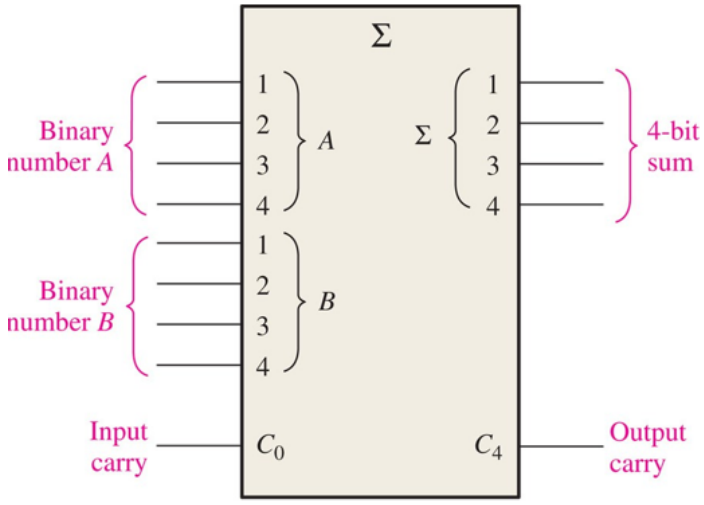
TABLE 6–3

Truth table for each stage of a 4-bit parallel adder.

C_{n-1}	A_n	B_n	Σ_n	C_n
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1



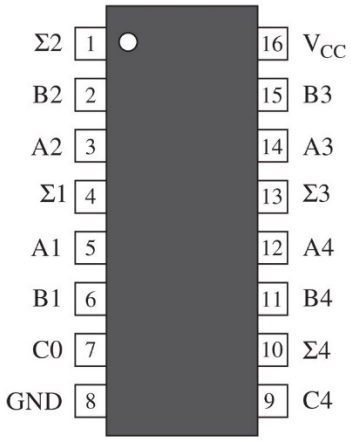
Block diagram



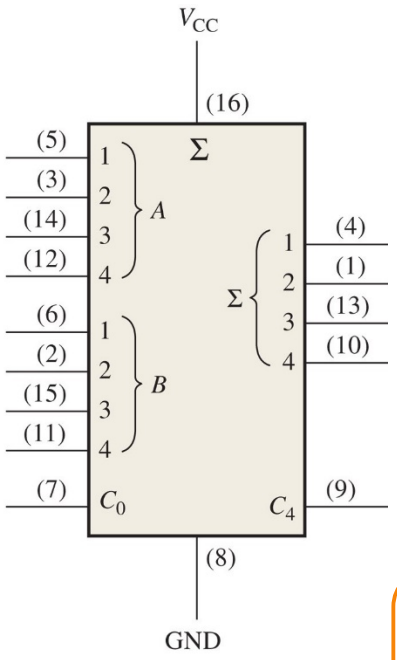
Logic symbol

Parallel Adders : 74HC283/74LS283

74HC283/74LS283



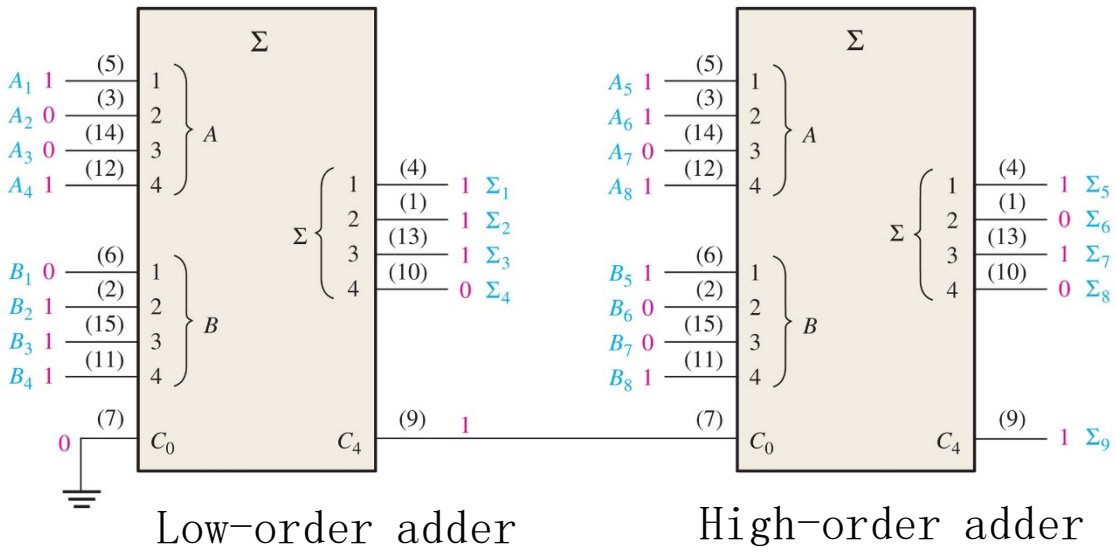
(a) Pin diagram



(b) Logic symbol

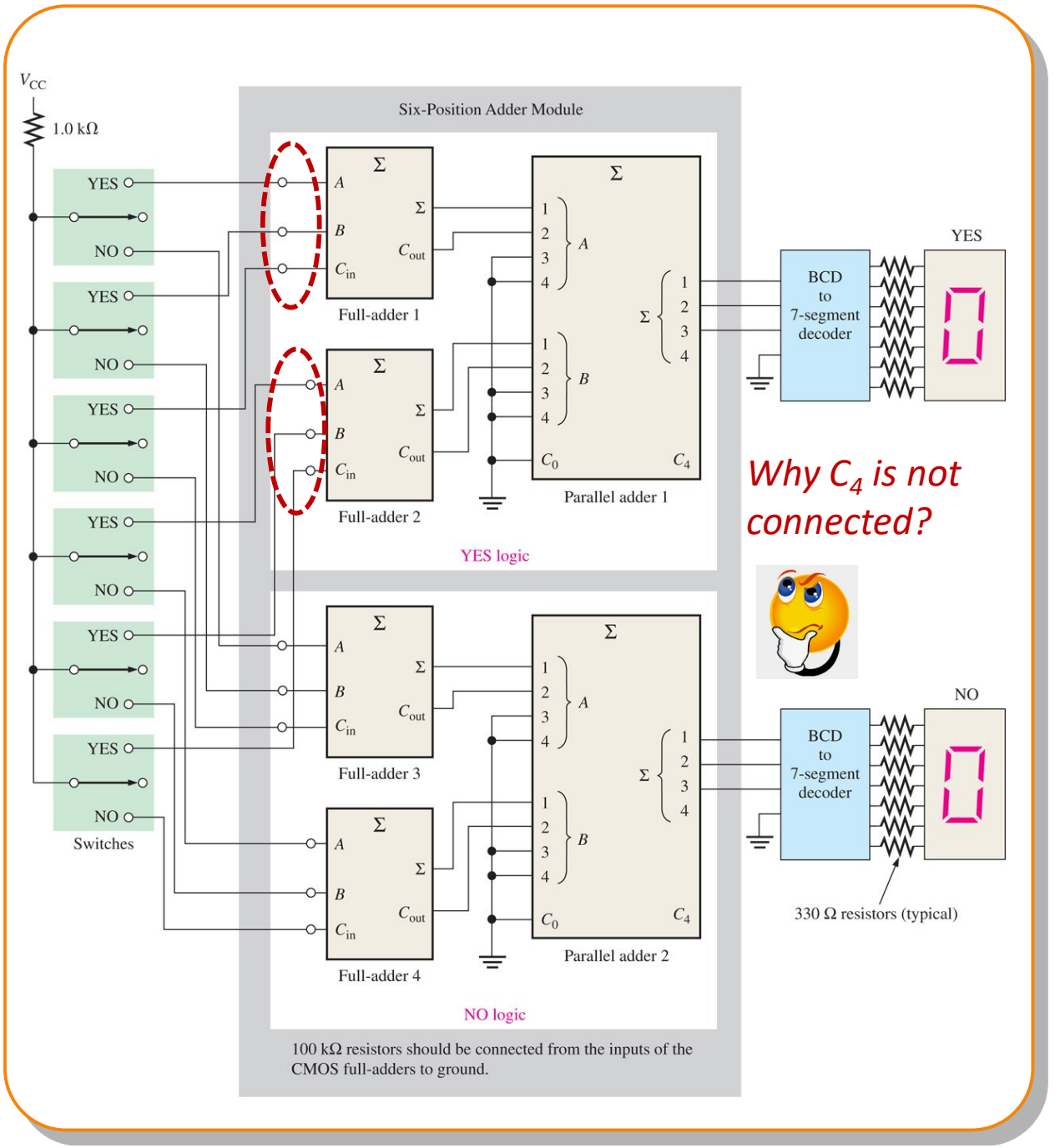
- 74HC283/74LS283: 4-bit parallel adders
- Cascading two 4-bit parallel adders to handle the addition of two 8-bit numbers
- The carry input of the low-order adder is connected to ground

- The carry output of the low-order adder is connected to the carry input of the high-order adder



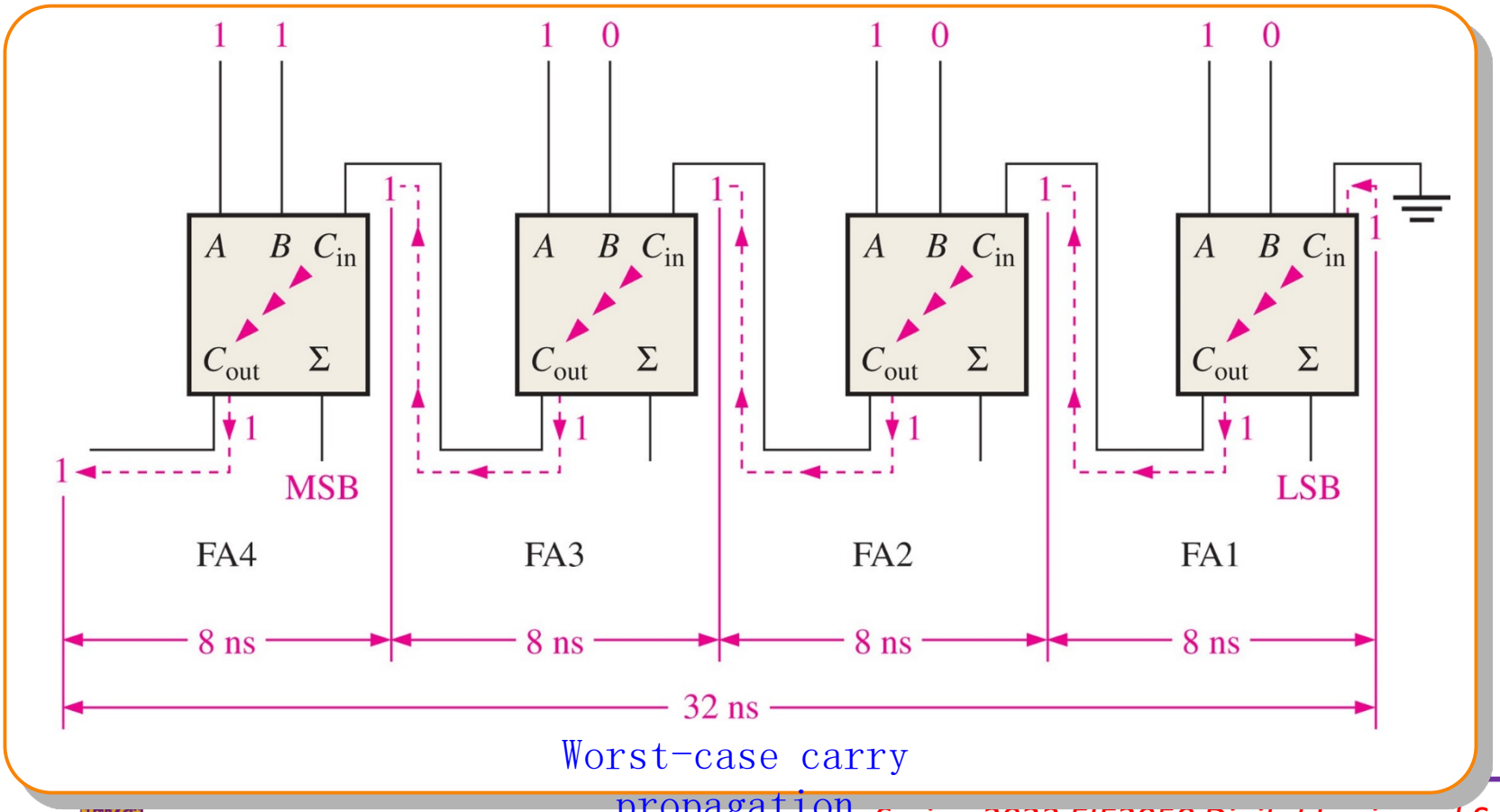
Applications

- A voting system that simultaneously counts # of "yes" and # of "no" votes.
- Assuming 6-position setup for simplicity
- Each input is LOW when the switch is in the neutral
- When a switch is moved to the "yes" or "no" position, a HIGH level (V_{CC}) is applied to the associated full adder input.



The Ripple Carry Adder

- The carry output of each full-adder is connected to the carry input of the next higher-order stage (a stage is one full-adder);
- The sum and the output carry of any stage cannot be produced until the input carry occurs, which causes a time delay in the addition process.

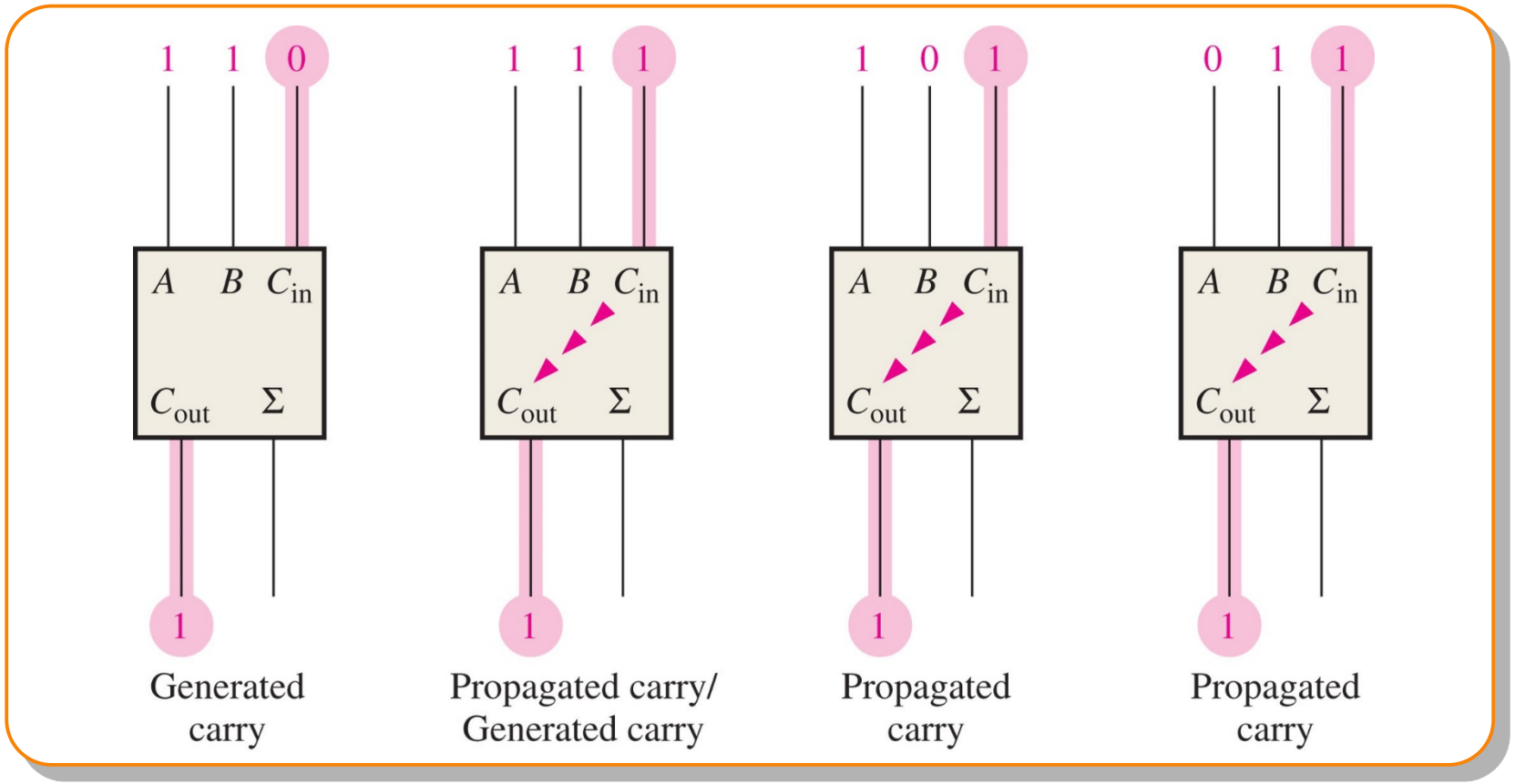


Worst-case carry propagation

The input carry to the least significant stage has to ripple through all the adders before a final sum is produced.

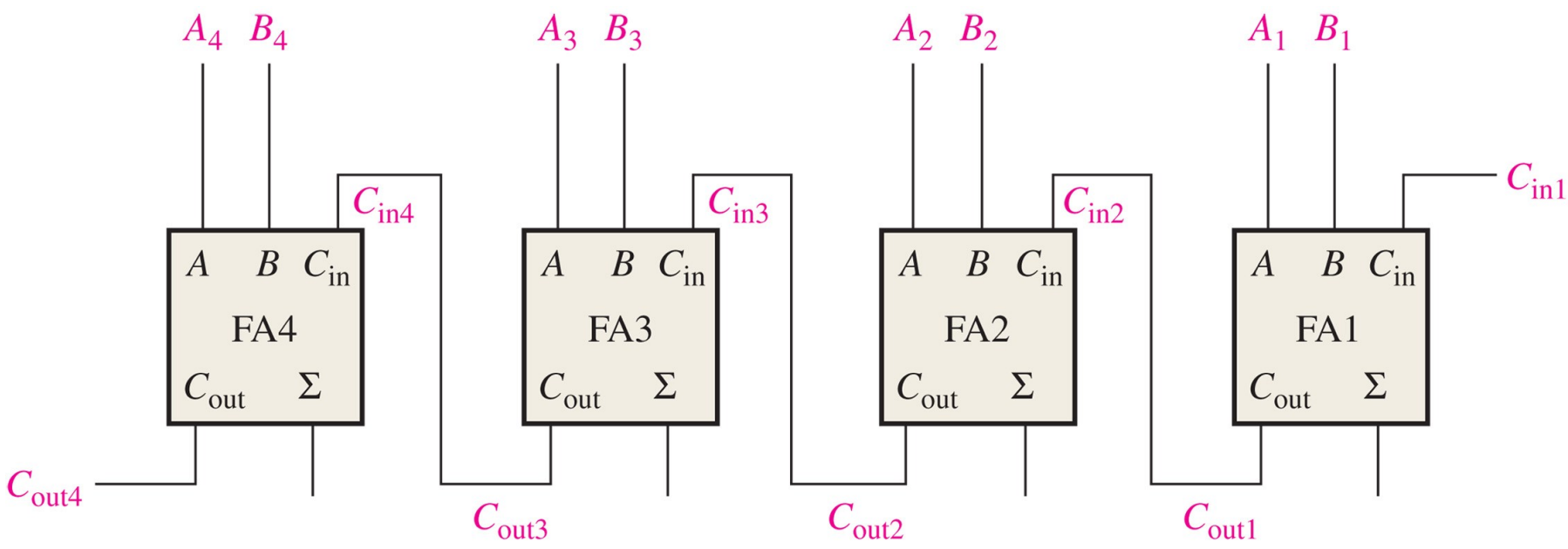
The Look-Ahead Carry Adder

- To speed up the addition process by eliminating the ripple carry delay
- Carry generation: $C_g = AB$
- Carry propagation: Given the input carry is 1, $C_p = A + B$
- Thus, the output carry is given by $C_{out} = Cg + CpCin$



A 4-Bit Look-Ahead Carry Adder (I)

- Adding $A_4A_3A_2A_1$ and $B_4B_3B_2B_1$

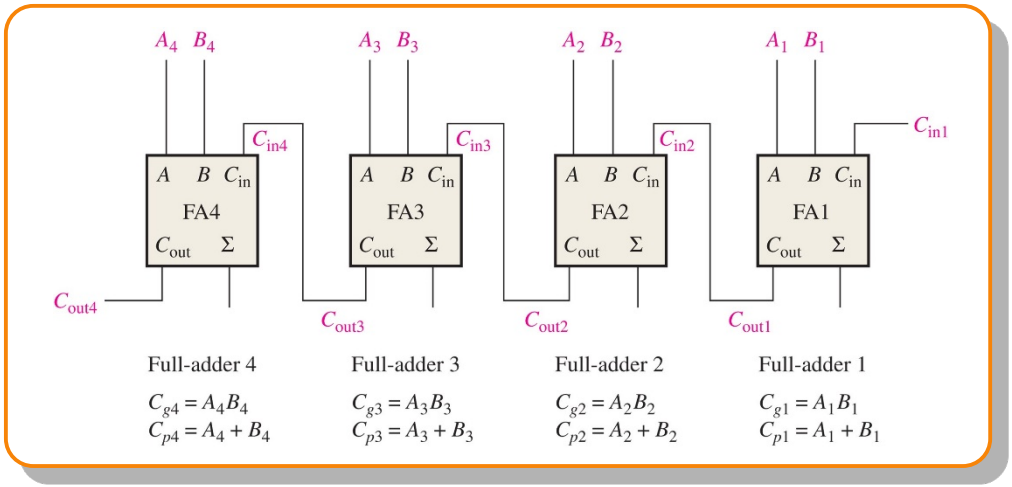


Immediately available

Full-adder 4	Full-adder 3	Full-adder 2	Full-adder 1
$C_{g4} = A_4B_4$	$C_{g3} = A_3B_3$	$C_{g2} = A_2B_2$	$C_{g1} = A_1B_1$
$C_{p4} = A_4 + B_4$	$C_{p3} = A_3 + B_3$	$C_{p2} = A_2 + B_2$	$C_{p1} = A_1 + B_1$

A 4-Bit Look-Ahead Carry Adder (II)

- Express C_{out} in terms of C_p, C_g and C_{in1}



Full-adder 1:

$$C_{out1} = C_{g1} + C_{p1}C_{in1}$$

Full-adder 2:

$$C_{in2} = C_{out1}$$

$$C_{out2} = C_{g2} + C_{p2}C_{in2} = C_{g2} + C_{p2}C_{out1} = C_{g2} + C_{p2}(C_{g1} + C_{p1}C_{in1})$$
$$= C_{g2} + C_{p2}C_{g1} + C_{p2}C_{p1}C_{in1}$$

Full-adder 3:

$$C_{in3} = C_{out2}$$

$$C_{out3} = C_{g3} + C_{p3}C_{in3} = C_{g3} + C_{p3}C_{out2} = C_{g3} + C_{p3}(C_{g2} + C_{p2}C_{g1} + C_{p2}C_{p1}C_{in1})$$
$$= C_{g3} + C_{p3}C_{g2} + C_{p3}C_{p2}C_{g1} + C_{p3}C_{p2}C_{p1}C_{in1}$$

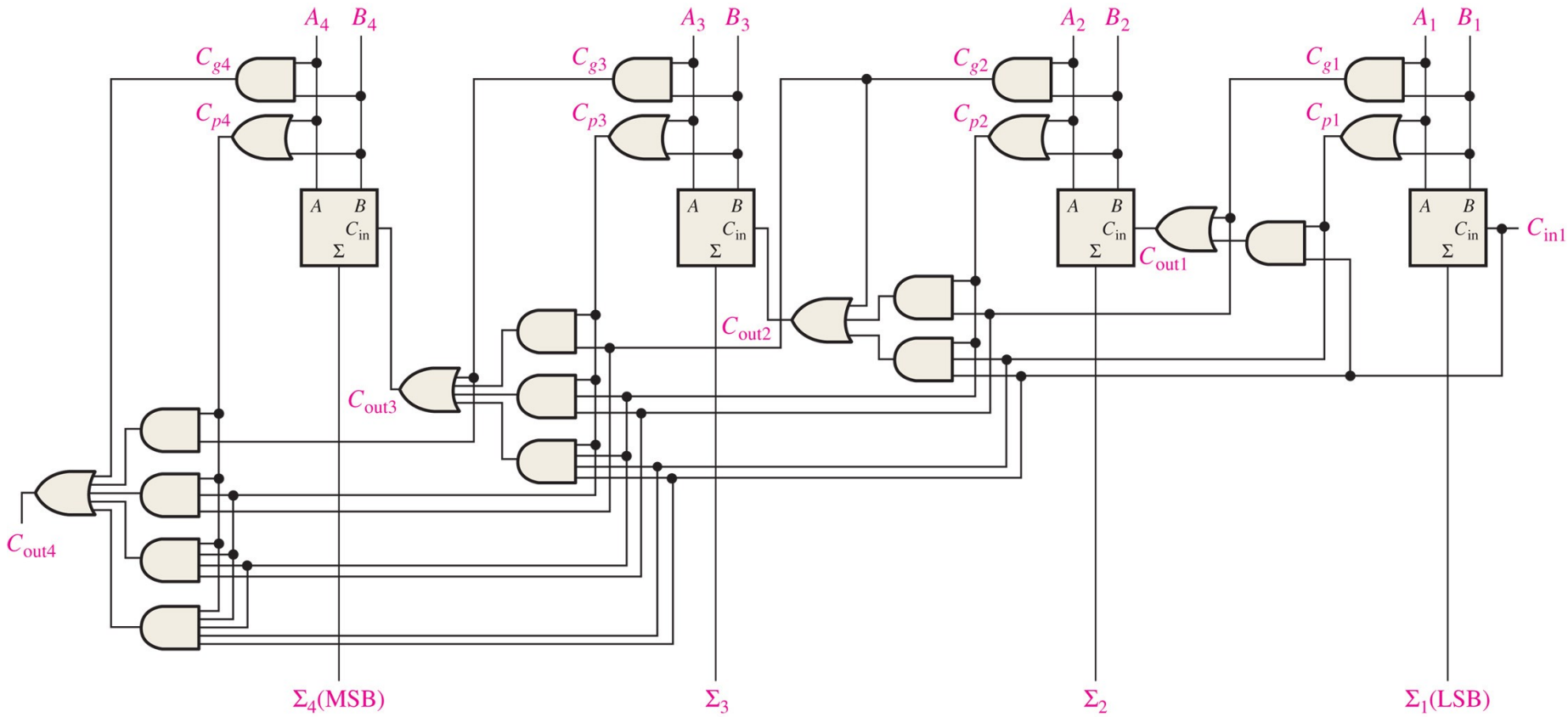
Full-adder 4:

$$C_{in4} = C_{out3}$$

$$C_{out4} = C_{g4} + C_{p4}C_{in4} = C_{g4} + C_{p4}C_{out3}$$
$$= C_{g4} + C_{p4}(C_{g3} + C_{p3}C_{g2} + C_{p3}C_{p2}C_{g1} + C_{p3}C_{p2}C_{p1}C_{in1})$$
$$= C_{g4} + C_{p4}C_{g3} + C_{p4}C_{p3}C_{g2} + C_{p4}C_{p3}C_{p2}C_{g1} + C_{p4}C_{p3}C_{p2}C_{p1}C_{in1}$$

A 4-Bit Look-Ahead Carry Adder (II)

- Logic diagram for a 4-stage look-ahead carry adder

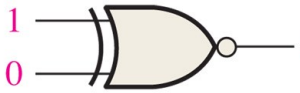


- Combination Look-Ahead and Ripple Carry Adders: Cascading two look-ahead carry adders by connecting the output carry of one adder to the input carry of the next adder.

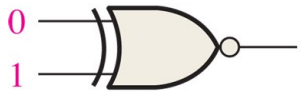
Comparators (I)

- Exclusive-NOR gate can be used as a basic comparator

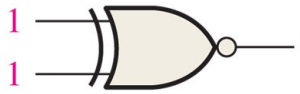
The input bits are equal.



The input bits are not equal.

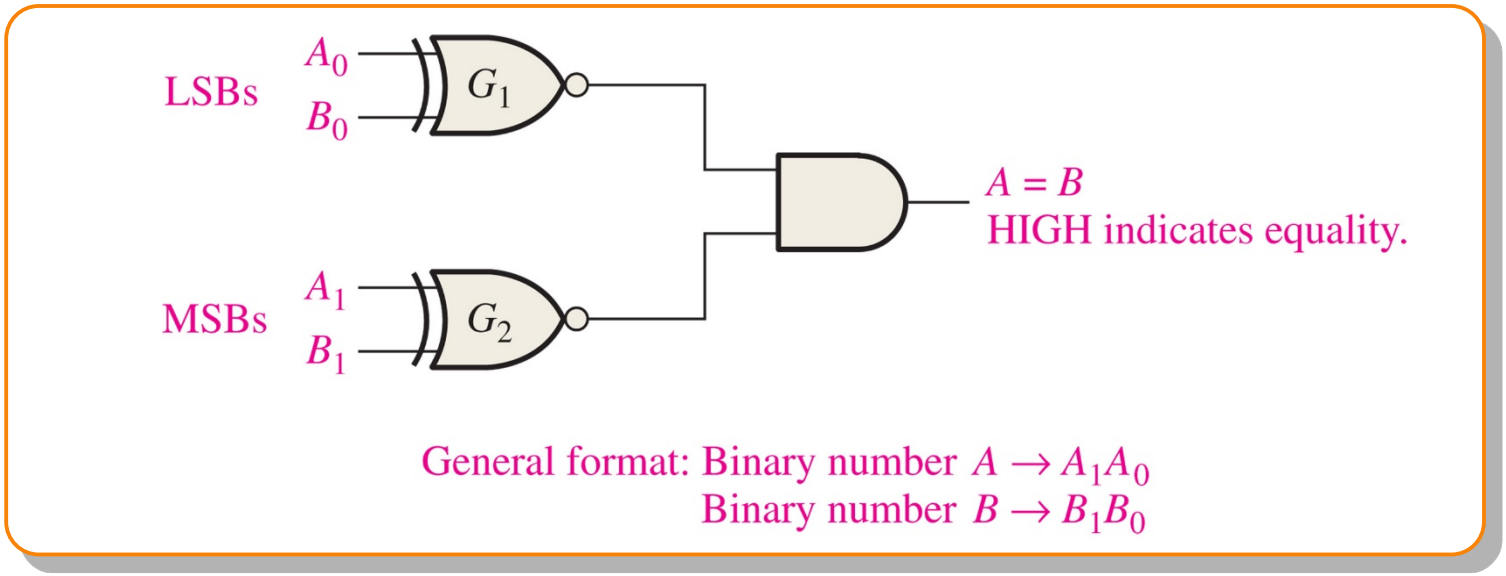


The input bits are not equal.



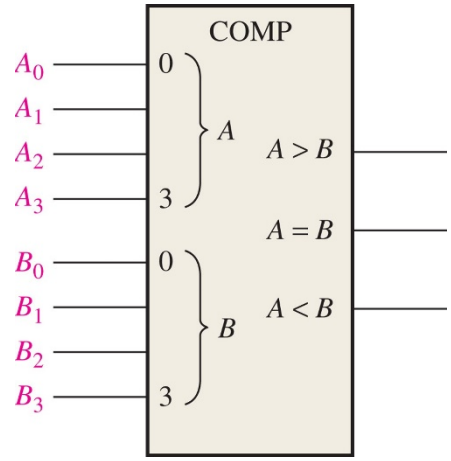
The input bits are equal.

- Equality comparison of two 2-bit numbers



Comparators (II)

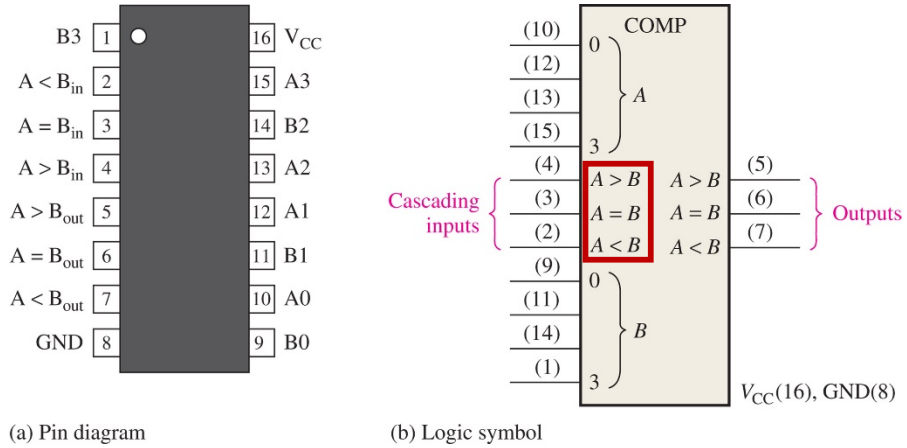
- Inequality comparison with three outputs indicating
  - Number A is greater than number B (A>B)
  - Number A is less than number B (A< B)
  - Number A is less than number B (A = B).
- General procedures
  - To check for an inequality in a bit position, starting with the highest-order bits (MSBs).
  - When such an inequality is found, the relationship of the two numbers is established, and any other inequalities in lower-order bit positions must be ignored
- Using a 4-bit comparator as an example
  - If $A_3 = 1$ and $B_3 = 0$, number A is greater than number B.
  - If $A_3 = 0$ and $B_3 = 1$, number A is less than number B.
  - If $A_3 = B_3$, then examine the next lower bit position.



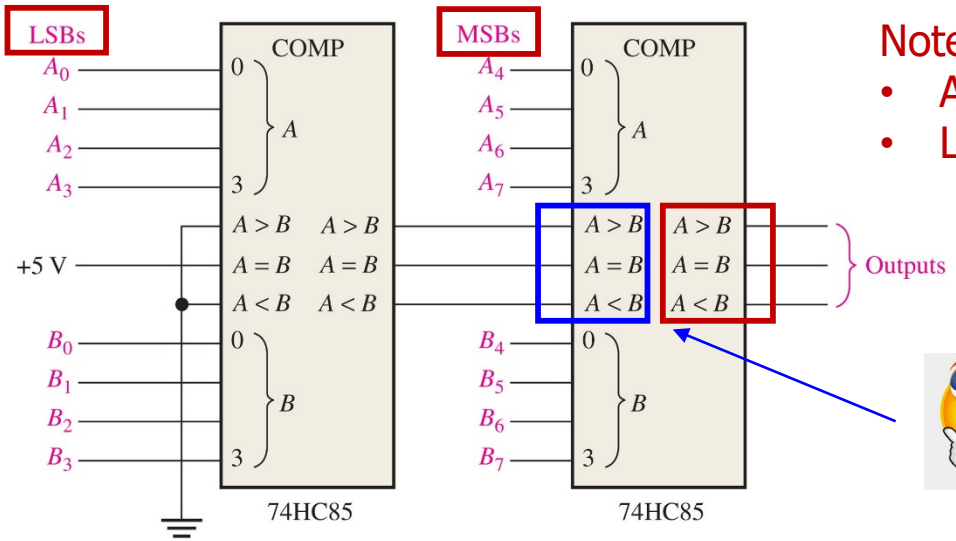
Logic Symbol

Comparators (III) : 74HC85/74LS85

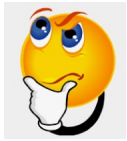
- The 74HC85/74LS85 pin diagram and logic symbol



- Several comparators can be cascaded for number of more bits



Note the lowest-order comparator must have
- A HIGH on the $A = B$ input and
- LOWs on the $A>B$ and $A<B$ inputs.



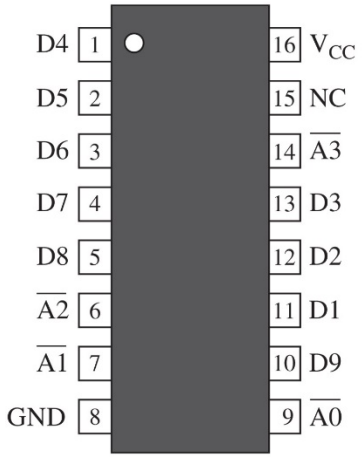
How these inputs (2)-(4) and outputs (5)-(7) are connected?

Encoders

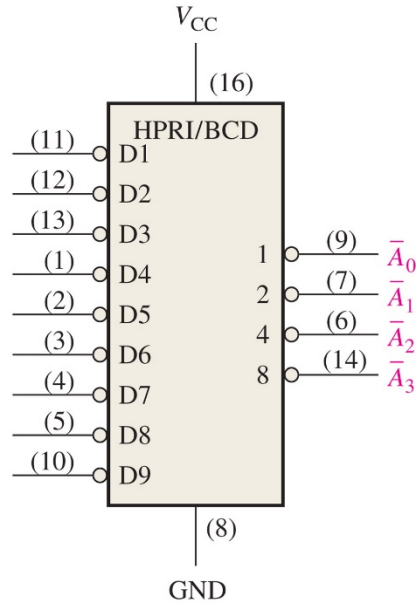
- Input: Accepts an active level on one of its multiple input lines
- Output: multiple *output* lines
- Example: a 10-line-to-4-line encoder

TABLE 6–6

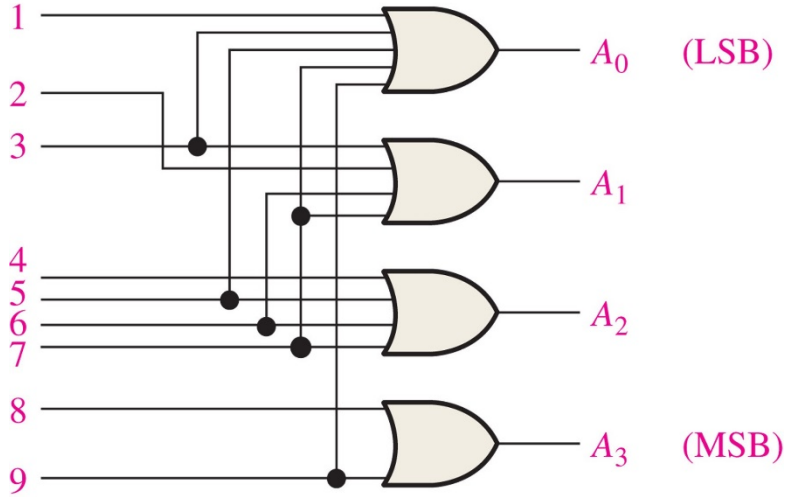
Decimal Digit	BCD Code A_3	A_2	A_1	A_0
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1



(a) Pin diagram



(b) Logic diagram

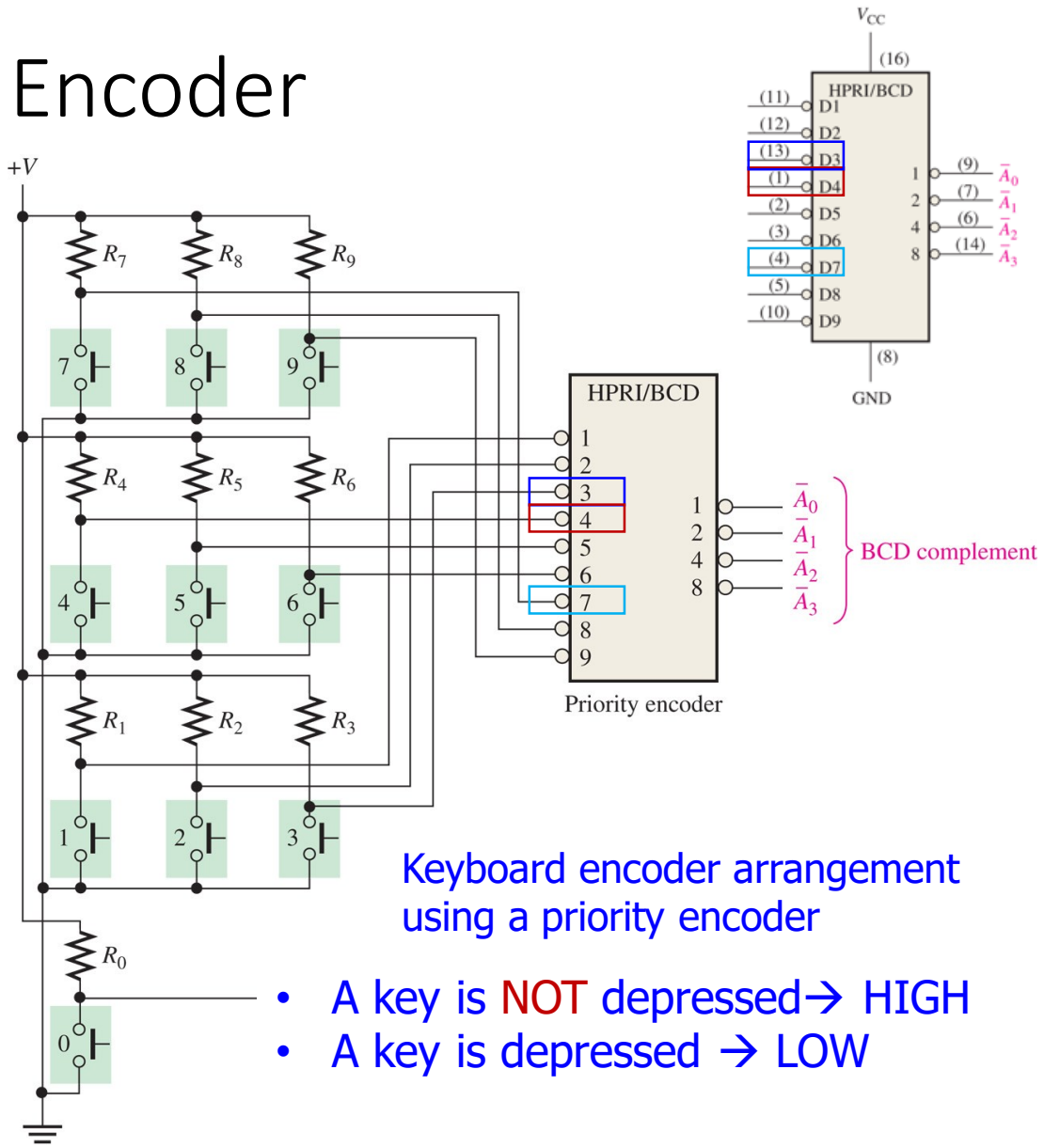


Decimal-to-BCD Priority Encoder

- Produce a BCD output corresponding to the highest-order decimal digit input that is active and will ignore any other lower-order active inputs

TABLE 6–6

Decimal Digit	BCD Code A_3	A_2	A_1	A_0
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1



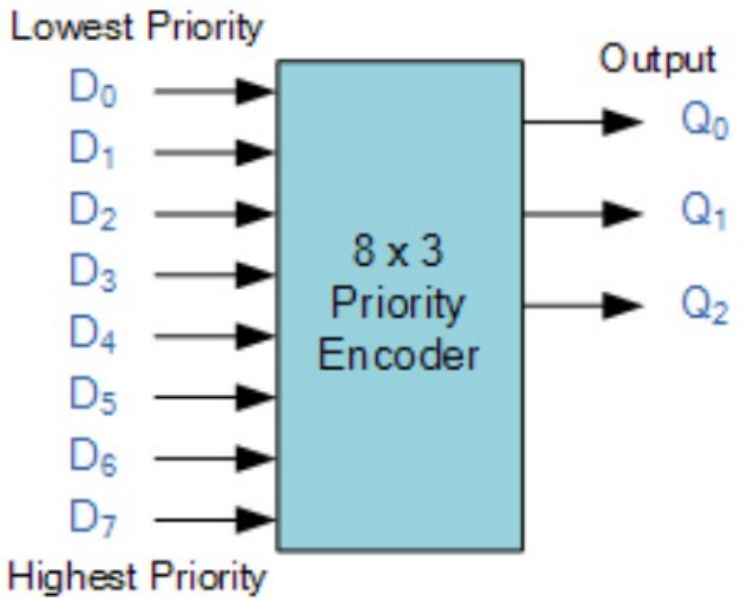
Keyboard encoder arrangement using a priority encoder

- A key is NOT depressed→ HIGH
- A key is depressed → LOW

Example 6-11 If LOW levels appear on pins, 1, 4, and 13 of the 74HC147 → Pin 4 is the highest-order decimal digit input having a LOW level and represents decimal 7. Therefore, the output levels indicate the BCD code for decimal 7

Priority Encoder

- Use "Don't Care" to indicate priority by design



Inputs								Outputs		
D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	Q_2	Q_1	Q_0
0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	1	x	0	0	1
0	0	0	0	0	1	x	x	0	1	0
0	0	0	0	1	x	x	x	0	1	1
0	0	0	1	x	x	x	x	1	0	0
0	0	1	x	x	x	x	x	1	0	1
0	1	x	x	x	x	x	x	1	1	0
1	x	x	x	x	x	x	x	1	1	1

X = dont care

Code Converters : BCD-to-Binary Conversion

- The binary numbers representing the weights of the BCD bits are summed to produce the total binary number.

TABLE 6–7

An 8-bit BCD and its bit weights

BCD Bit	BCD Weight	(MSB) 64	32	Binary Representation 16	8	4	2	(LSB) 1
A_0	1	0	0	0	0	0	0	1
A_1	2	0	0	0	0	0	1	0
A_2	4	0	0	0	0	1	0	0
A_3	8	0	0	0	1	0	0	0
B_0	10	0	0	0	1	0	1	0
B_1	20	0	0	1	0	1	0	0
B_2	40	0	1	0	1	0	0	0
B_3	80	1	0	1	0	0	0	0

$B_3\ B_2\ B_1\ B_0\ A_3\ A_2\ A_1\ A_0$

80 40 20 10 8 4 2 1

0 0 1 0 0 1 1 1

```
0000001    1
0000010    2
0000100    4
+0010100   20
0011011    Binary number for decimal 27
```

80 40 20 10 8 4 2 1

1 0 0 1 1 0 0 0

```
0001000    8
0001010    10
+1010000   80
1100010    Binary number for decimal 98
```

- Convert the BCD numbers 00100111 (decimal 27) and 10011000 (decimal 98) to binary.

Decoders: Basics

- Input: n input lines
- Output: max. 2^n output lines
- Example: a 4-bit Decoder

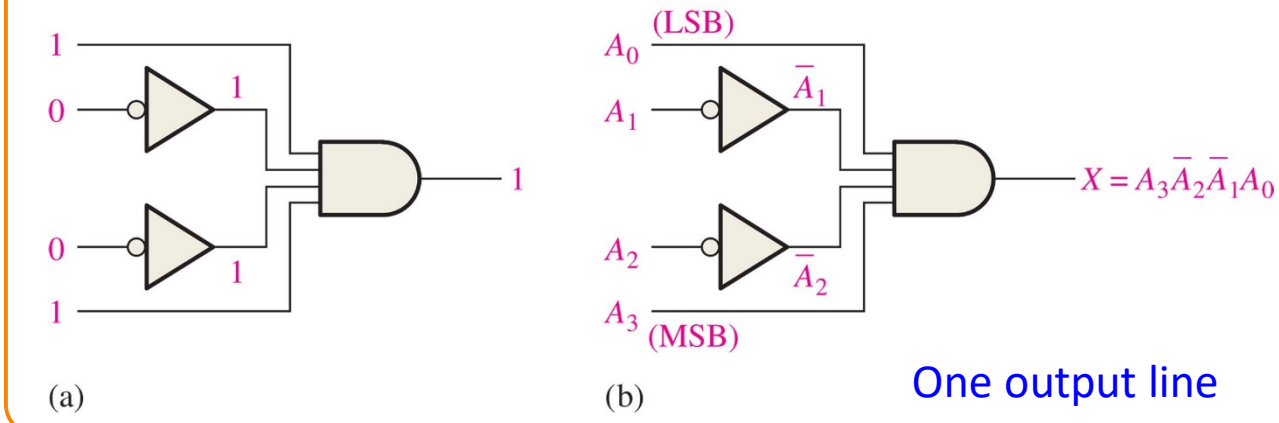
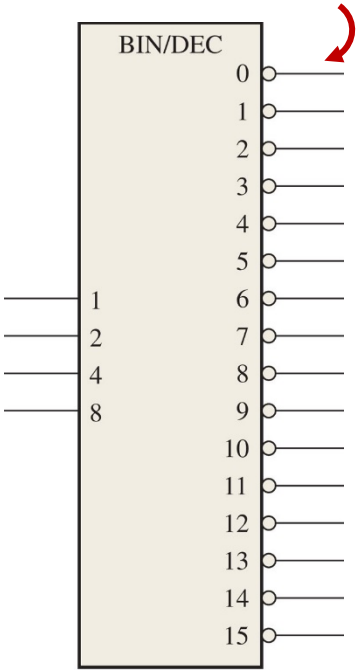


TABLE 6–4

Decoding functions and truth table for a 4-line-to-16-line (1-of-16) decoder with active-LOW outputs.

Decimal Digit	Binary Inputs A_3	A_2	A_1	A_0	Decoding Function	Outputs 0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	0	$\overline{A}_3\overline{A}_2\overline{A}_1\overline{A}_0$	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	0	0	0	1	$\overline{A}_3\overline{A}_2\overline{A}_1A_0$	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1
2	0	0	1	0	$\overline{A}_3\overline{A}_2A_1\overline{A}_0$	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1
3	0	0	1	1	$\overline{A}_3\overline{A}_2A_1A_0$	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1
4	0	1	0	0	$\overline{A}_3A_2\overline{A}_1\overline{A}_0$	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1
5	0	1	0	1	$\overline{A}_3A_2\overline{A}_1A_0$	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1
6	0	1	1	0	$\overline{A}_3A_2A_1\overline{A}_0$	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1
7	0	1	1	1	$\overline{A}_3A_2A_1A_0$	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1
8	1	0	0	0	$A_3\overline{A}_2\overline{A}_1\overline{A}_0$	1	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1
9	1	0	0	1	$A_3\overline{A}_2\overline{A}_1A_0$	1	1	1	1	1	1	1	1	1	0	1	1	1	1	1	1
10	1	0	1	0	$A_3\overline{A}_2A_1\overline{A}_0$	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1	1
11	1	0	1	1	$A_3\overline{A}_2A_1A_0$	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1
12	1	1	0	0	$A_3A_2\overline{A}_1\overline{A}_0$	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1
13	1	1	0	1	$A_3A_2\overline{A}_1A_0$	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1
14	1	1	1	0	$A_3A_2A_1\overline{A}_0$	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1
15	1	1	1	1	$A_3A_2A_1A_0$	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0

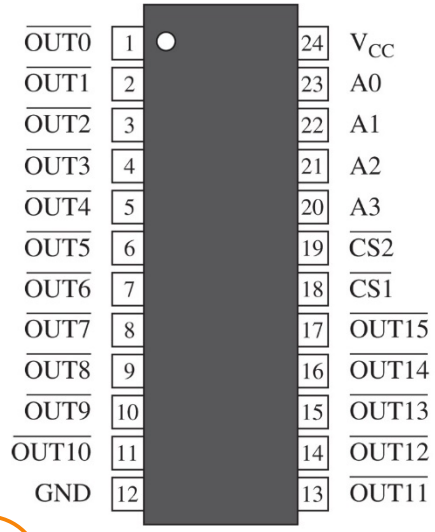
16 NAND Gates

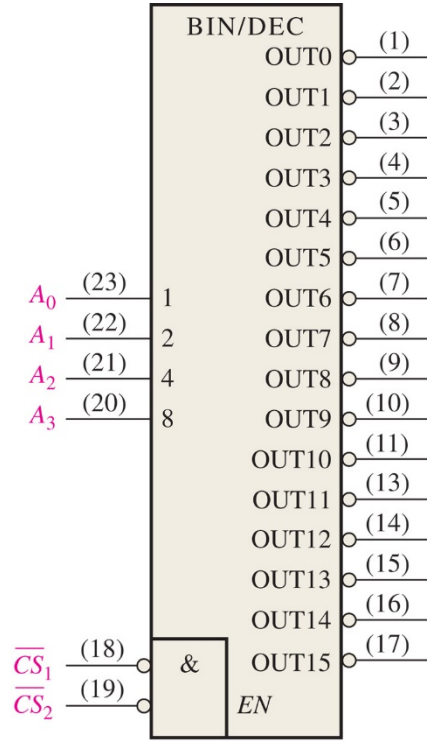


Decoding binary 4-bit numbers

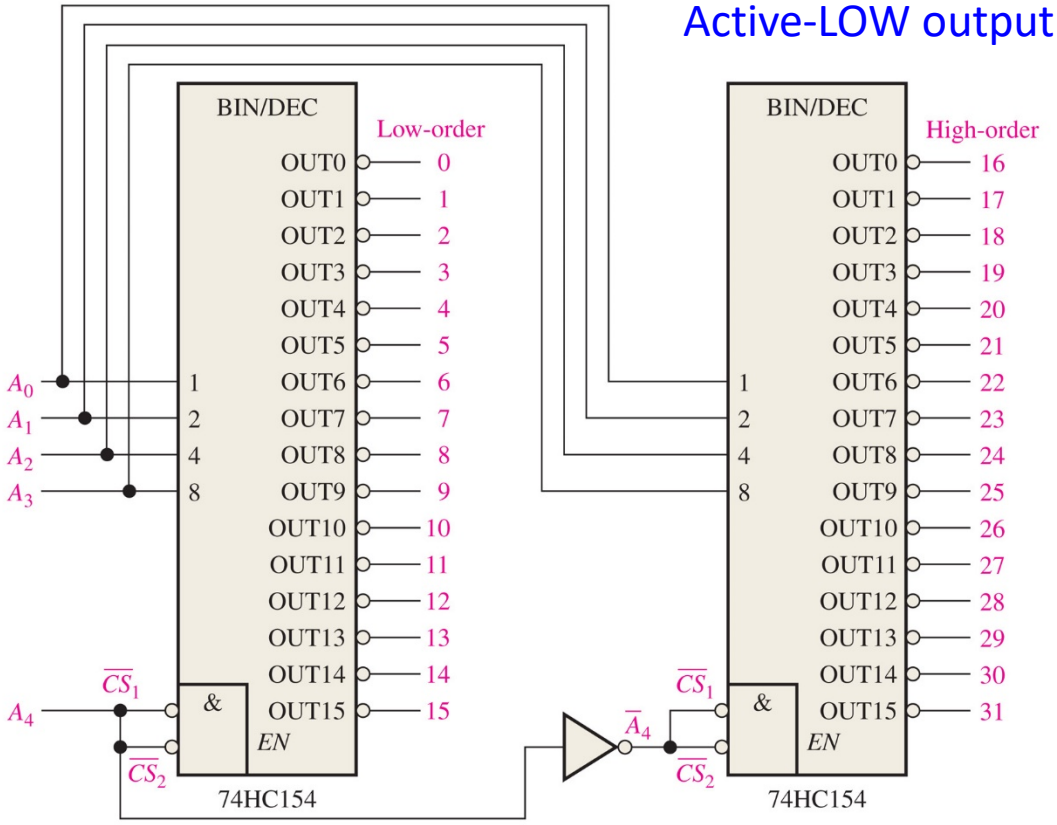
Decoders: 74HC154

- EN: $\overline{CS_1}$ and $\overline{CS_2}$ both LOW → Output HIGH
- EN is connected to an input of each NAND
- EN: HIGH to enable the decoder; otherwise all decoder outputs will be HIGH





Active-LOW output

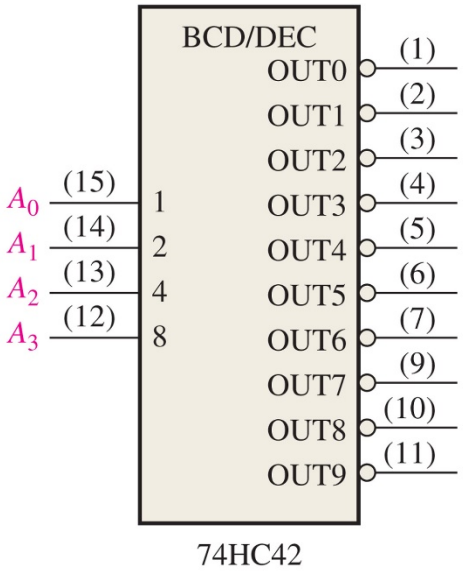


$\overline{A_4}$=0 when >15

- Implement a 5-bit decoder using two 4-bit decoders (74HC154)
- A_4 and $\overline{A_4}$ connected to $\overline{CS_1}$ and $\overline{CS_2}$ of two decoders, respectively;
- Enable the high-order decoder when the decimal number >15; otherwise, the low-order decoder.

Decoders: BCD-to-Decimal Decoder

- 4-line-to-10-line decoder: only the ten decimal digits 0 through 9.



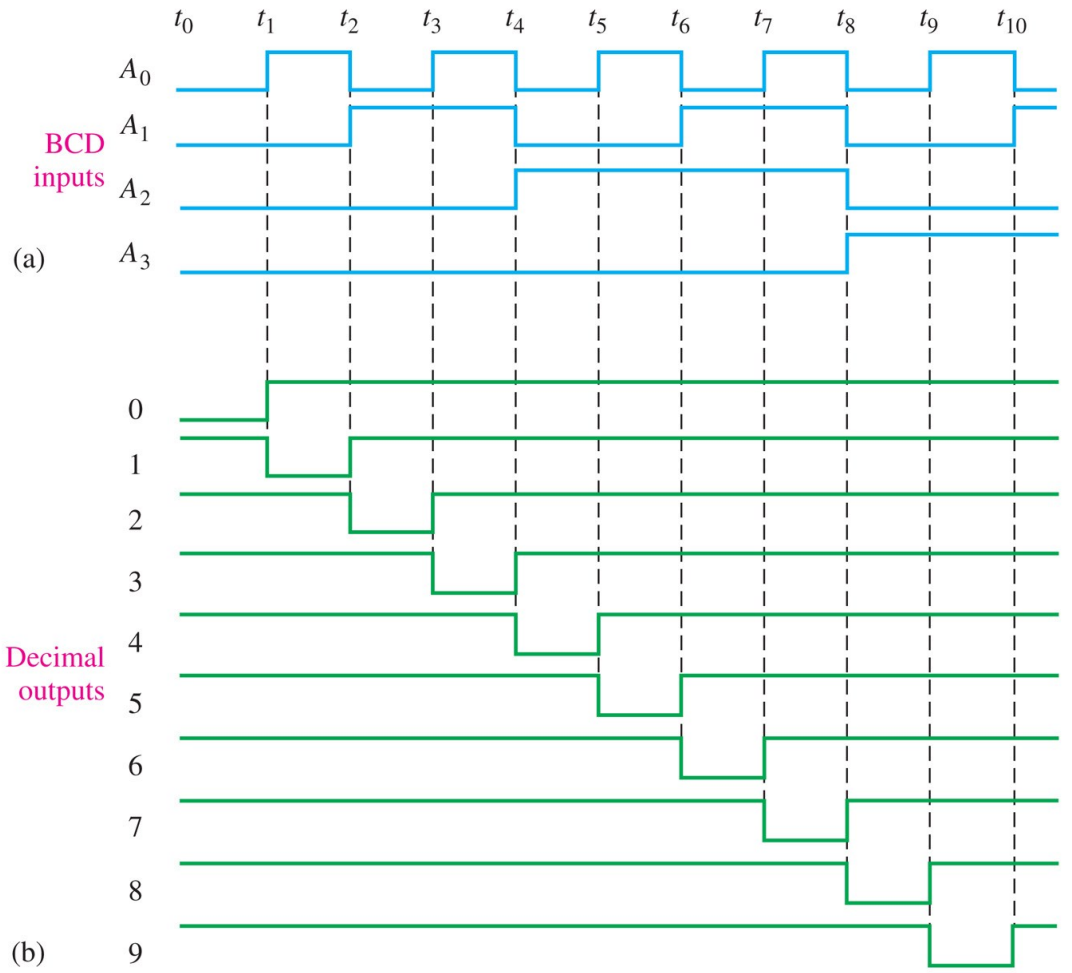
Active-LOW output

TABLE 6–5

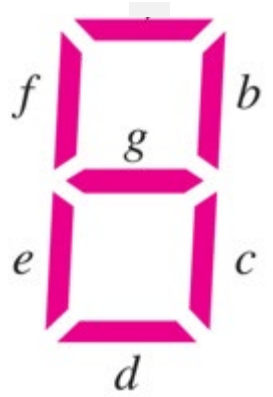
BCD decoding functions.

Decimal Digit	BCD Code A_3	A_2	A_1	A_0	Decoding Function
0	0	0	0	0	$\bar{A}_3\bar{A}_2\bar{A}_1\bar{A}_0$
1	0	0	0	1	$\bar{A}_3\bar{A}_2\bar{A}_1A_0$
2	0	0	1	0	$\bar{A}_3\bar{A}_2A_1\bar{A}_0$
3	0	0	1	1	$\bar{A}_3\bar{A}_2A_1A_0$
4	0	1	0	0	$\bar{A}_3A_2\bar{A}_1\bar{A}_0$
5	0	1	0	1	$\bar{A}_3A_2\bar{A}_1A_0$
6	0	1	1	0	$\bar{A}_3A_2A_1\bar{A}_0$
7	0	1	1	1	$\bar{A}_3A_2A_1A_0$
8	1	0	0	0	$A_3\bar{A}_2\bar{A}_1\bar{A}_0$
9	1	0	0	1	$A_3\bar{A}_2\bar{A}_1A_0$

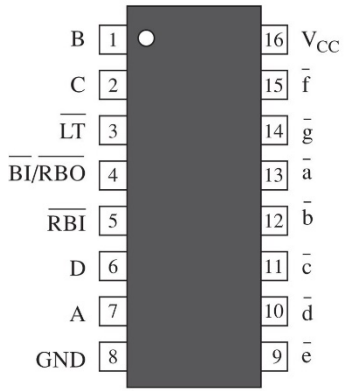
The inputs are sequenced through the BCD for digits 0 through 9

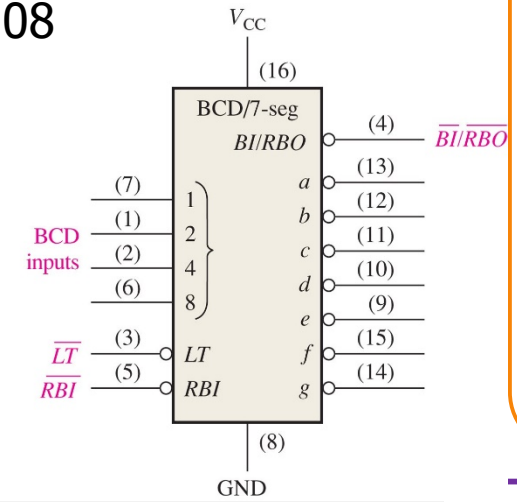


Decoders: The BCD-to-7-Segment Decoder

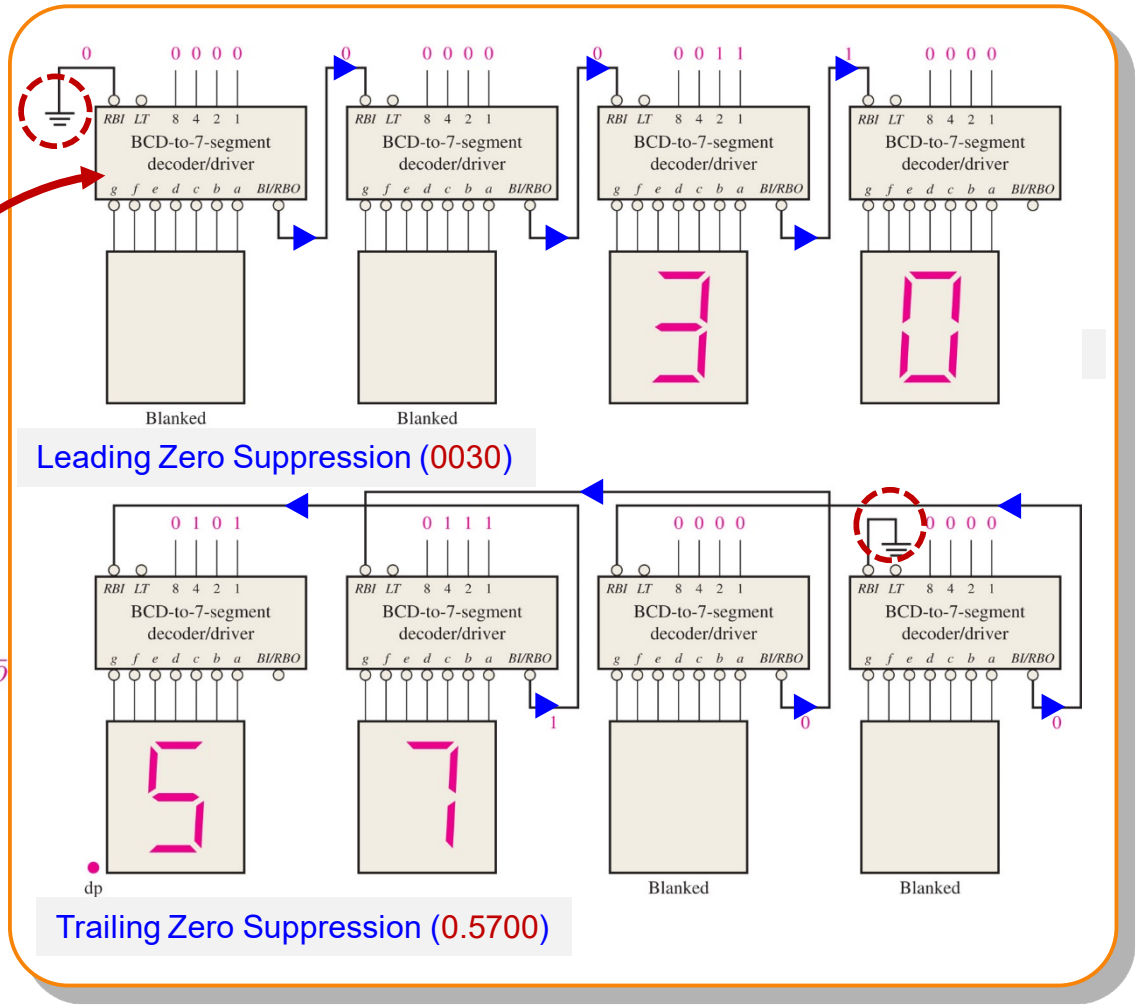


- Inequality comparison with three outputs indicating
  - LT : Lamp test
  - RBI : Ripple blanking input
  - BI /RBO : either Blanking input or Ripple blanking output
- Lamp test : $\overline{LT}$ and $\overline{BI}$ / $\overline{RBO}$ are LOW, all segments are turned on
- BCD inputs = 0000 and $\overline{RBI}$ is LOW, → decoder outputs HIGH, display blanked & $\overline{RBO}$ LOW.
- Zero suppression: 0030.0800 will be displayed as 30.08

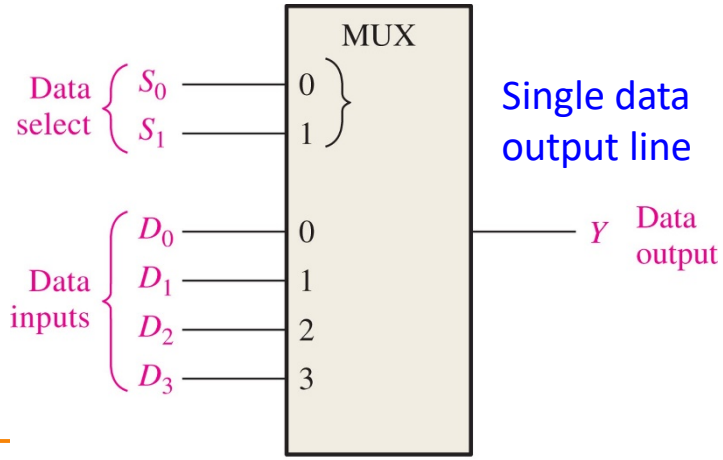




The 74HC47 BCD-to-7-segment decoder/driver.



Multiplexers (Data Selectors)

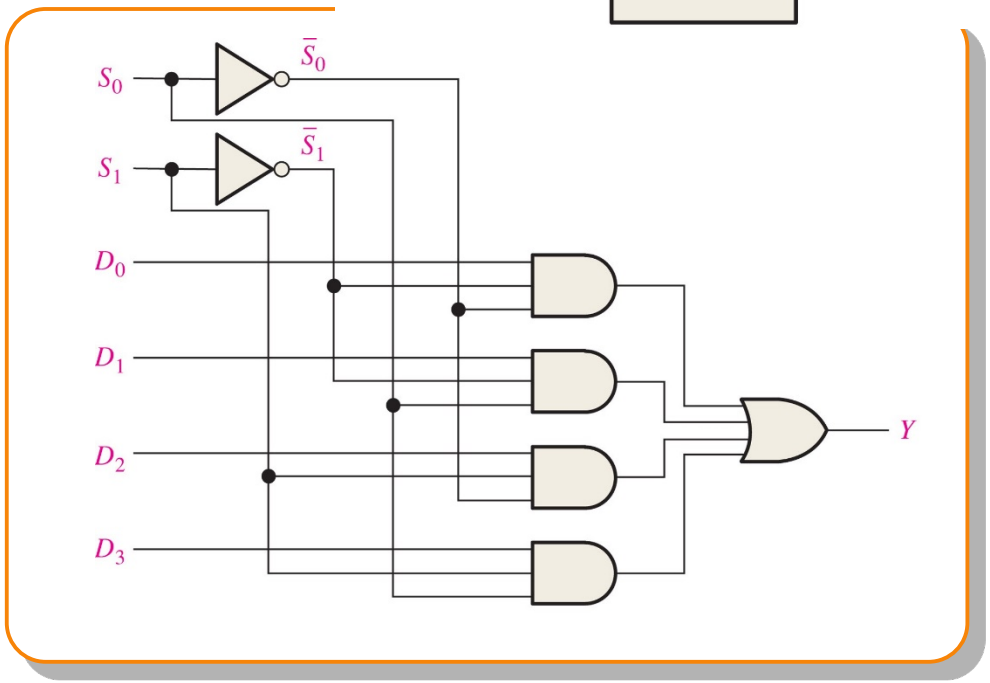


- To route several sources onto a single line linking to a common destination.

TABLE 6–8

Data selection for a 1-of-4-multiplexer.

Data-Select Inputs		
S_1	S_0	Input Selected
0	0	D_0
0	1	D_1
1	0	D_2
1	1	D_3



- The data output is equal to
  - D_0 only if $S_1 = 0$ and $S_0 = 0$: $Y = D_0\overline{S}_1\overline{S}_0$.
  - D_1 only if $S_1 = 0$ and $S_0 = 1$: $Y = D_1\overline{S}_1S_0$.
  - D_2 only if $S_1 = 1$ and $S_0 = 0$: $Y = D_2S_1\overline{S}_0$.
  - D_3 only if $S_1 = 1$ and $S_0 = 1$: $Y = D_3S_1S_0$.

$$\Rightarrow Y = D_0\overline{S}_1\overline{S}_0 + D_1\overline{S}_1S_0 + D_2S_1\overline{S}_0 + D_3S_1S_0$$

Example 6-44

- Determine the output waveform in relation to the inputs.

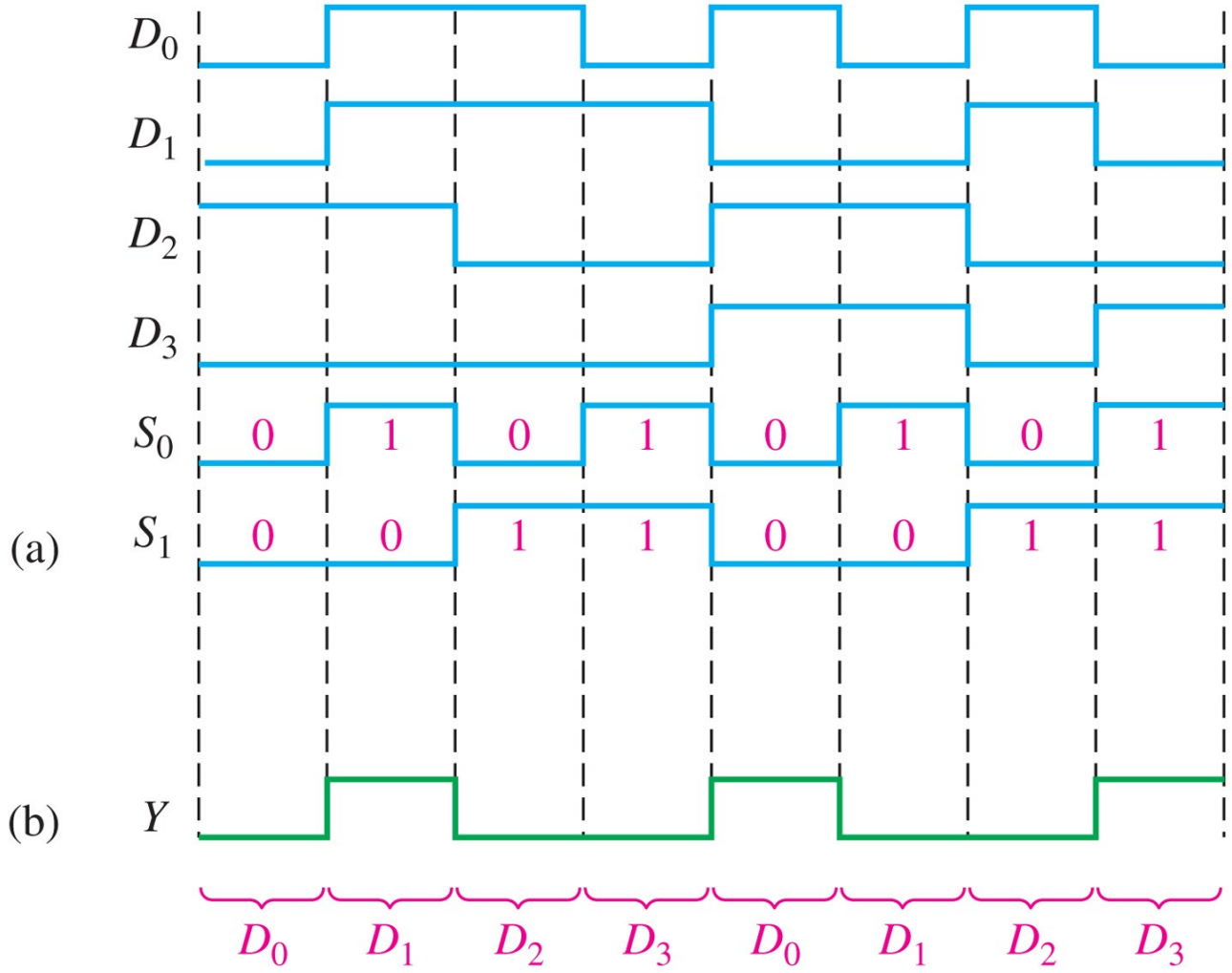


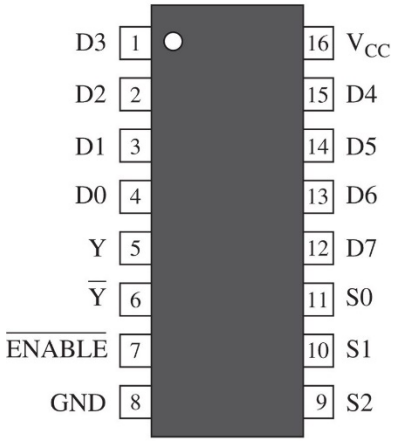
TABLE 6–8

Data selection for a 1-of-4-multiplexer.

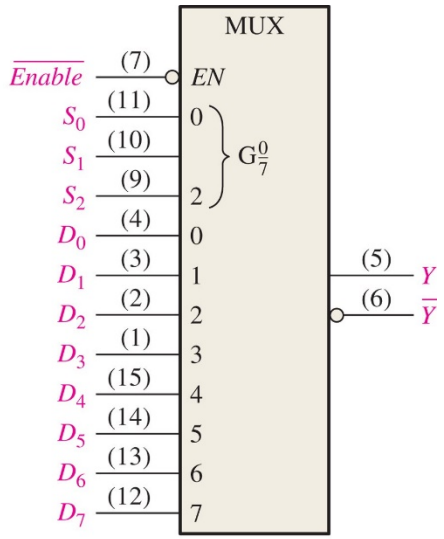
Data-Select Inputs		
S_1	S_0	Input Selected
0	0	D_0
0	1	D_1
1	0	D_2
1	1	D_3

Multiplexers 74HC151

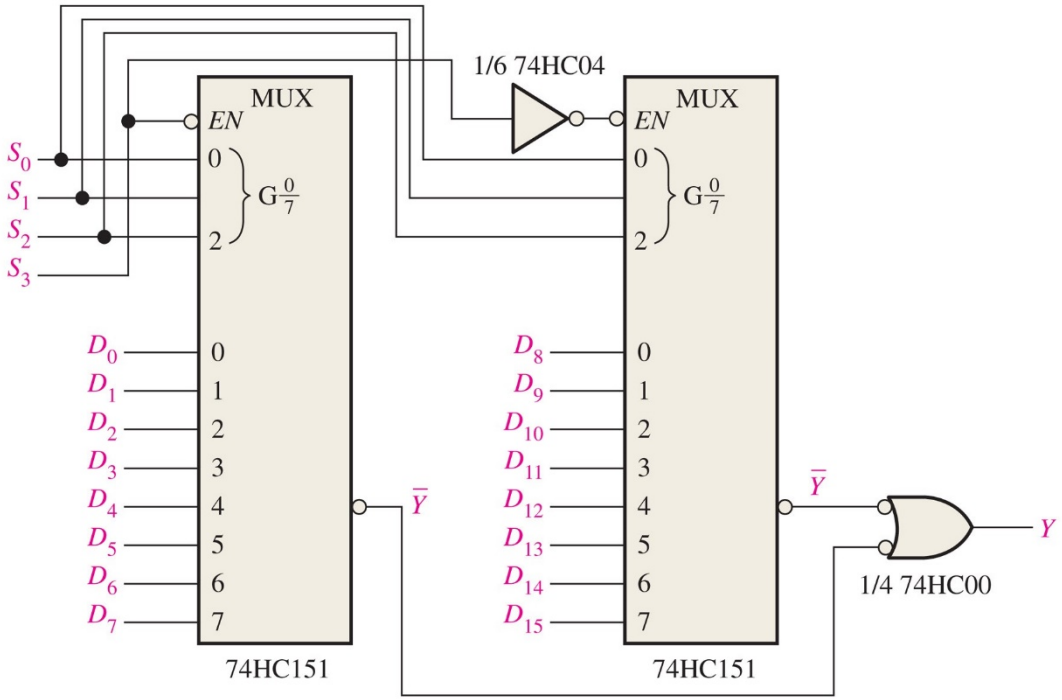
- 74HC151 : Eight data inputs (D_0–D_7), three data-select or address input lines (S_0–S_2).
- Example 6-15: Use 74HC151s and any other logic necessary to multiplex 16 data lines onto a single data-output line.



(a) Pin diagram



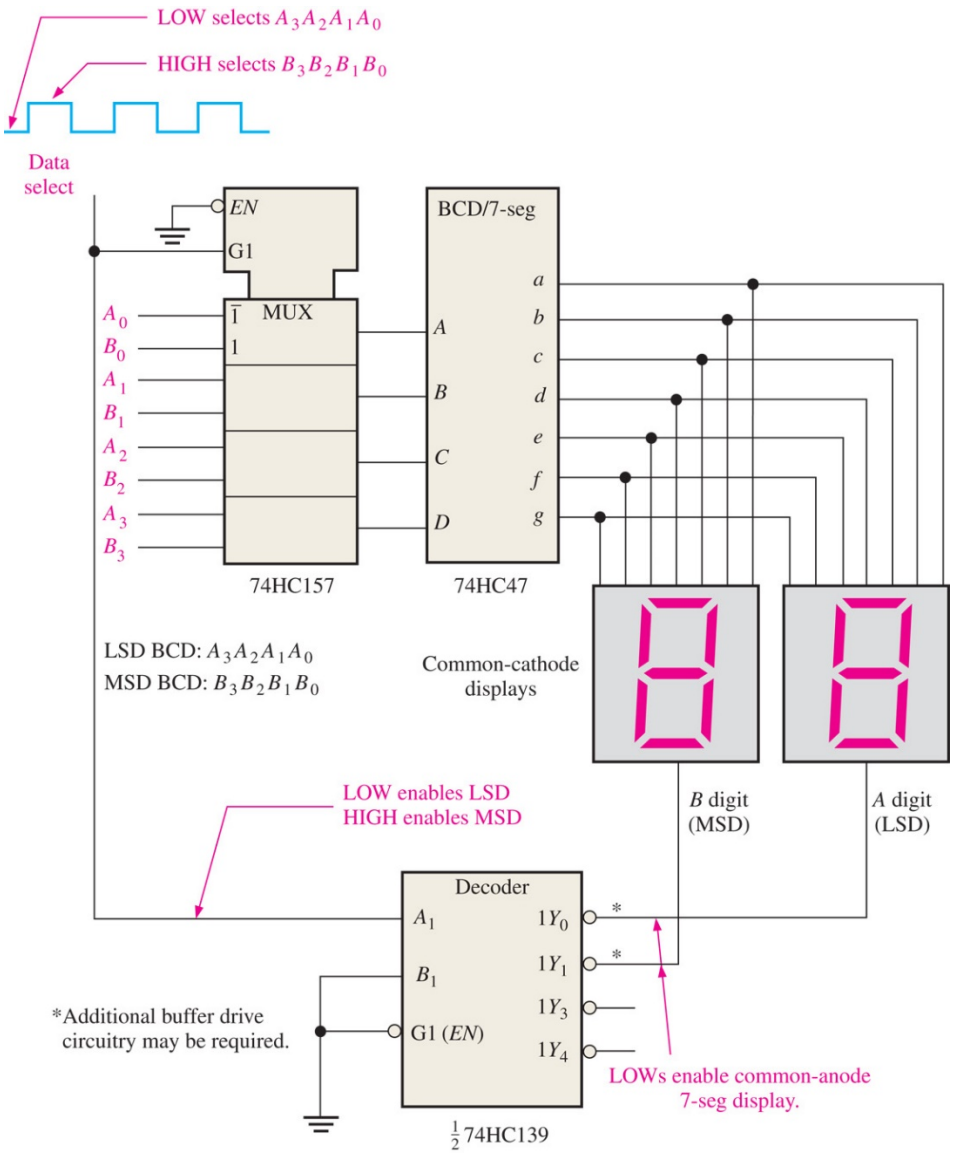
(b) Logic symbol



- $\overline{\text{Enable}}$ input is used as the most significant data-select bit.
- MSB→ LOW : Left MUX selected while right MUX outputs LOW
- MSB→ HIGH : Right MUX selected while left MUX outputs LOW

Applications : A 7-Segment Display Multiplexer

- 74HC151 : Eight data inputs (D_0–D_7), three data-select or address input lines (S_0–S_2).
- Inputs:
  - Two BCD digits (A3A2A1A0 and B3B2B1B0) and
  - A square wave (LOW enables A digit and HIGH B digit)
- Display A and B alternatively at a frequency defined by the data-select square wave → The frequency must be high enough to prevent visual flicker.



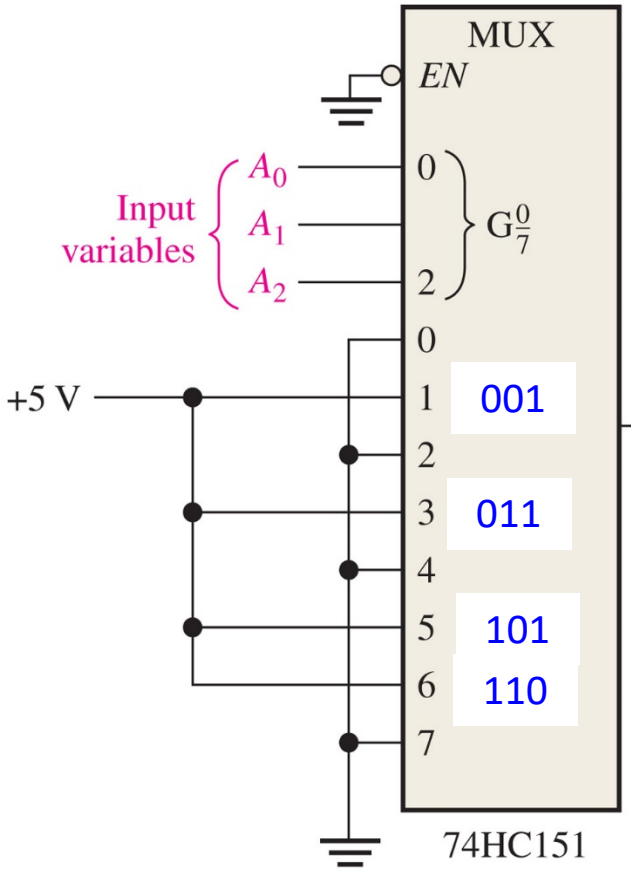
Use a square wave to enable LSD/MSD display

Applications : A Logic Function Generator

- Implement the logic function specified in the truth table by using a 74HC151 8-input data selector/multiplexer. Compare this method with a discrete logic gate implementation.

TABLE 6–9

Inputs			Output
A_2	A_1	A_0	Y
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0



$$Y = \bar{A}_2\bar{A}_1A_0 + \bar{A}_2A_1A_0 + A_2\bar{A}_1A_0 + A_2A_1\bar{A}_0$$

001 011 101 110

- This is an example of a logic function generator with three variables. Can we use 74HC151 to implement a true table with 16 combinations?

Example 6-17

- Truth table: 4 input variables of 16 combinations of input variables. However, only an 8-bit data selector is used.
- Use A_0 in conjunction with the data inputs.

TABLE 6–10

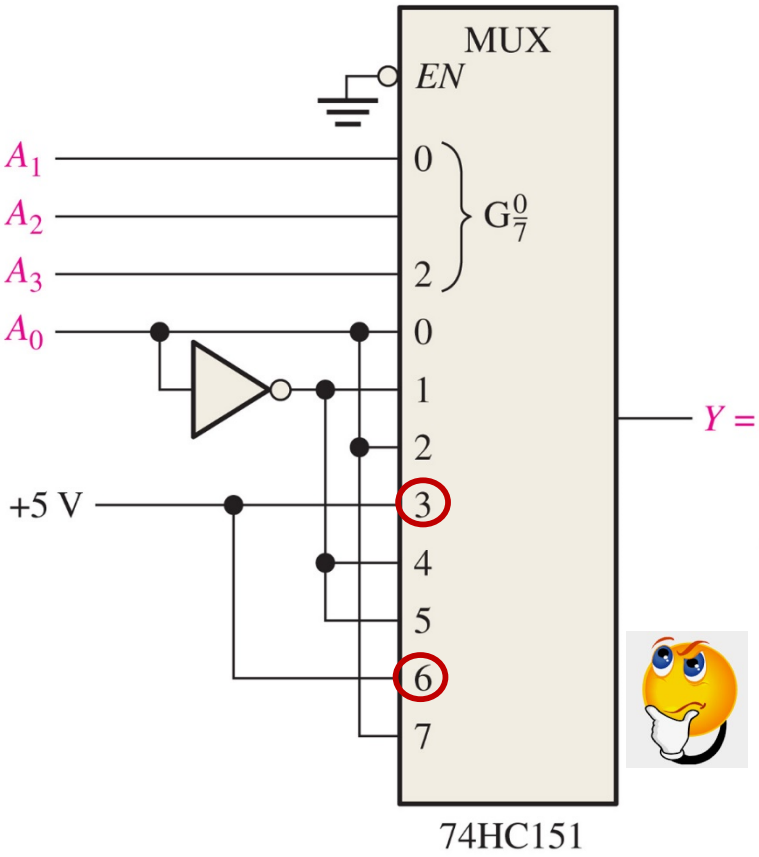
Decimal Digit	Inputs A_3	A_2	A_1	A_0	Output Y
0	0	0	0	0	0
1	0	0	0	1	1
2	0	0	1	0	1
3	0	0	1	1	0
4	0	1	0	0	0
5	0	1	0	1	1
6	0	1	1	0	1
7	0	1	1	1	1
8	1	0	0	0	1
9	1	0	0	1	0
10	1	0	1	0	1
11	1	0	1	1	0
12	1	1	0	0	1
13	1	1	0	1	1
14	1	1	1	0	0
15	1	1	1	1	1

$Y = A_0$

$Y = \overline{A_0}$

$Y = 1$

$Y = 1$



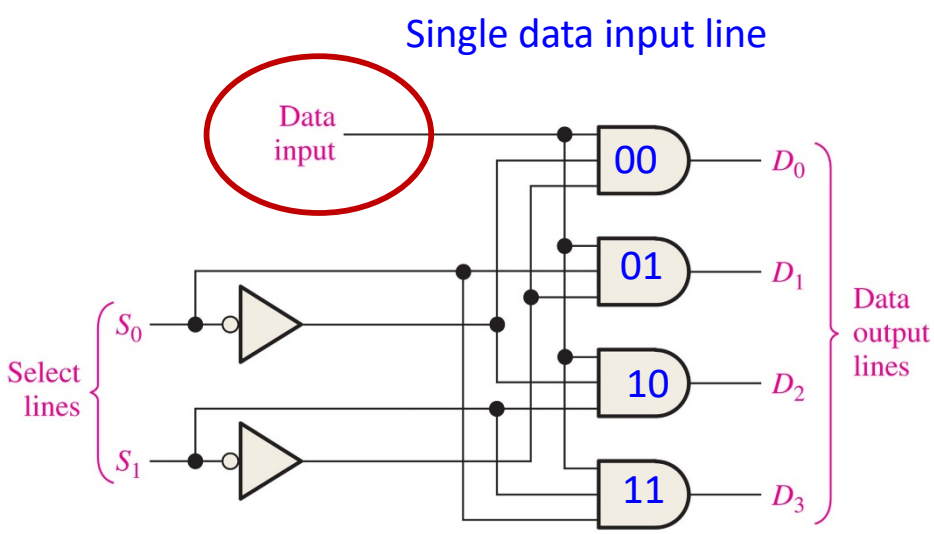
$$Y = \bar{A}_3\bar{A}_2\bar{A}_1A_0 + \bar{A}_3\bar{A}_2A_1\bar{A}_0 + \bar{A}_3A_2\bar{A}_1A_0 + \bar{A}_3A_2A_1\bar{A}_0 + \bar{A}_3A_2A_1A_0 + A_3\bar{A}_2\bar{A}_1\bar{A}_0 + A_3\bar{A}_2A_1\bar{A}_0 + A_3A_2\bar{A}_1\bar{A}_0 + A_3A_2A_1A_0$$

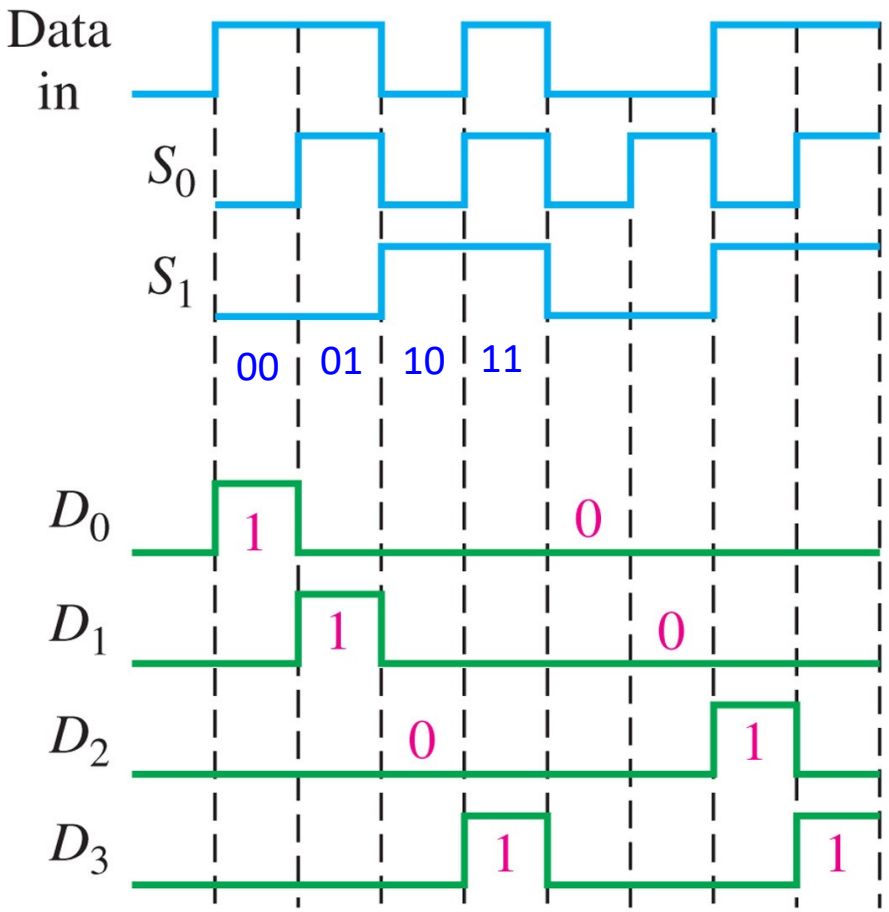
Is this truth table a special case or any 4-variable truth tables can be implemented in this manner?

Decimal Digit	A_3	A_2	A_1	Output
0	0	0	0	A_0
1	0	0	1	$\overline{A_0}$
2	0	1	0	A_0
3	0	1	1	1
4	1	0	0	$\overline{A_0}$
5	1	0	1	$\overline{A_0}$
6	1	1	0	1
7	1	1	1	A_0

Demultiplexers

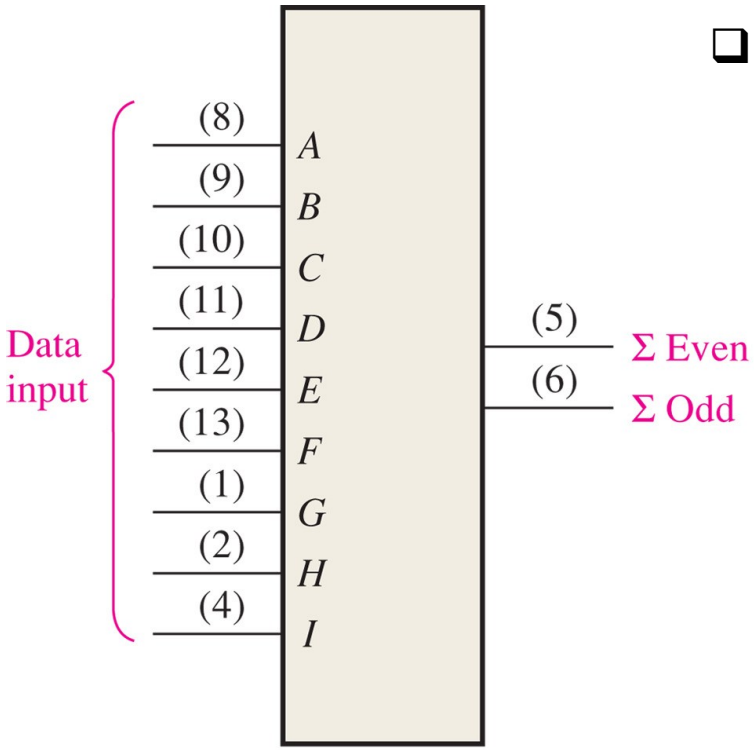
- Also known as a data distributor that reverses the multiplexing function





Parity Generators/Checkers

- 74HC280 can be used to check for odd or even parity on a 9-bit code (eight data bits and one parity bit), or it can be used to generate a parity bit for a binary code with up to nine bits.



(a) Traditional logic symbol

- Parity Checker: For **even** parity checkers, the # of input bits should always be even; and when a parity error occurs, the Σ Even output goes LOW and the Σ Odd output goes HIGH.

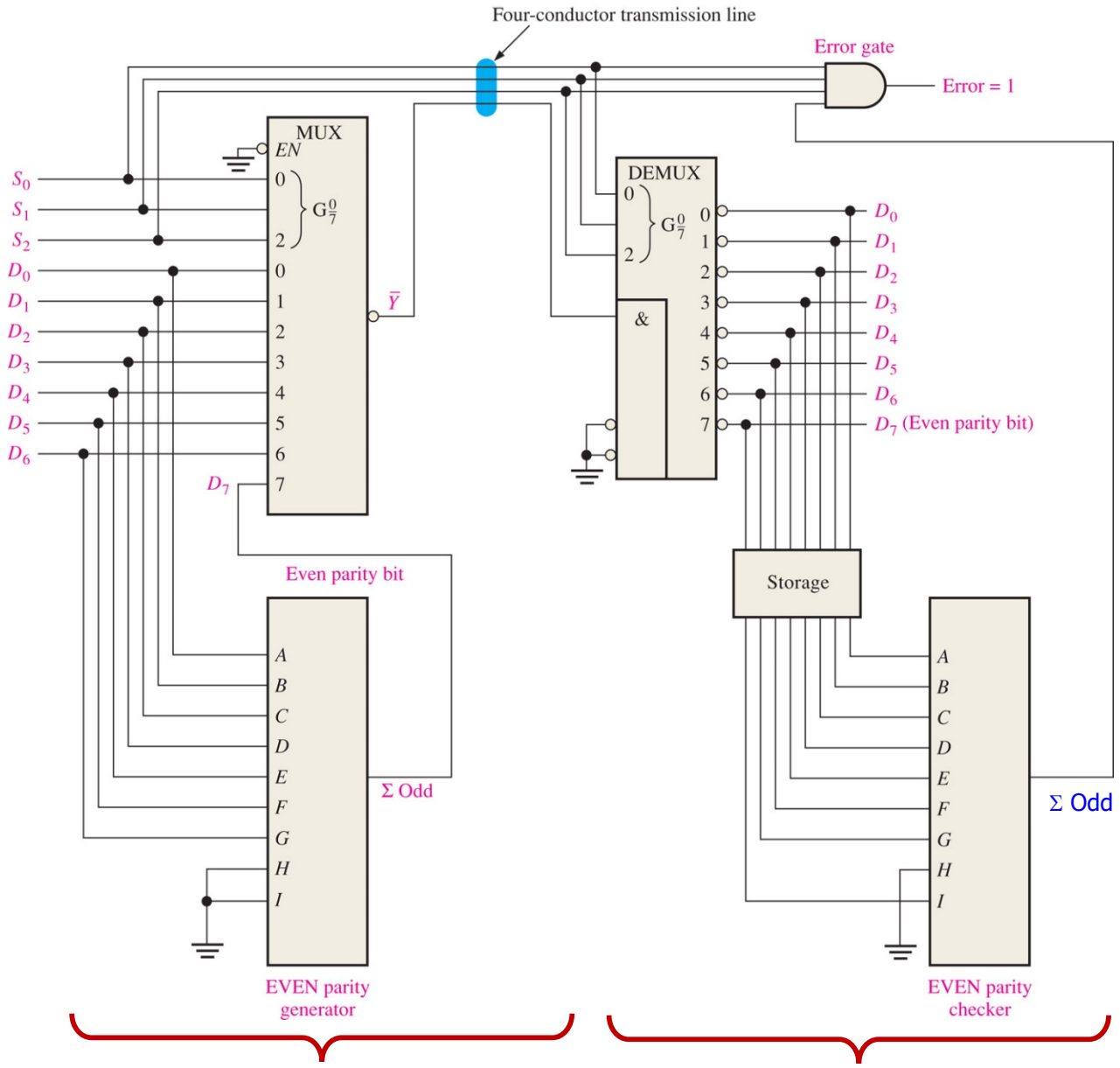
Number of Inputs *A–I* that Are High	Outputs Σ Even	Σ Odd
0, 2, 4, 6, 8	H	L
1, 3, 5, 7, 9	L	H

(b) Function table

- Parity Generator: For **even** parity generators, the parity bit is taken at the Σ Odd output.

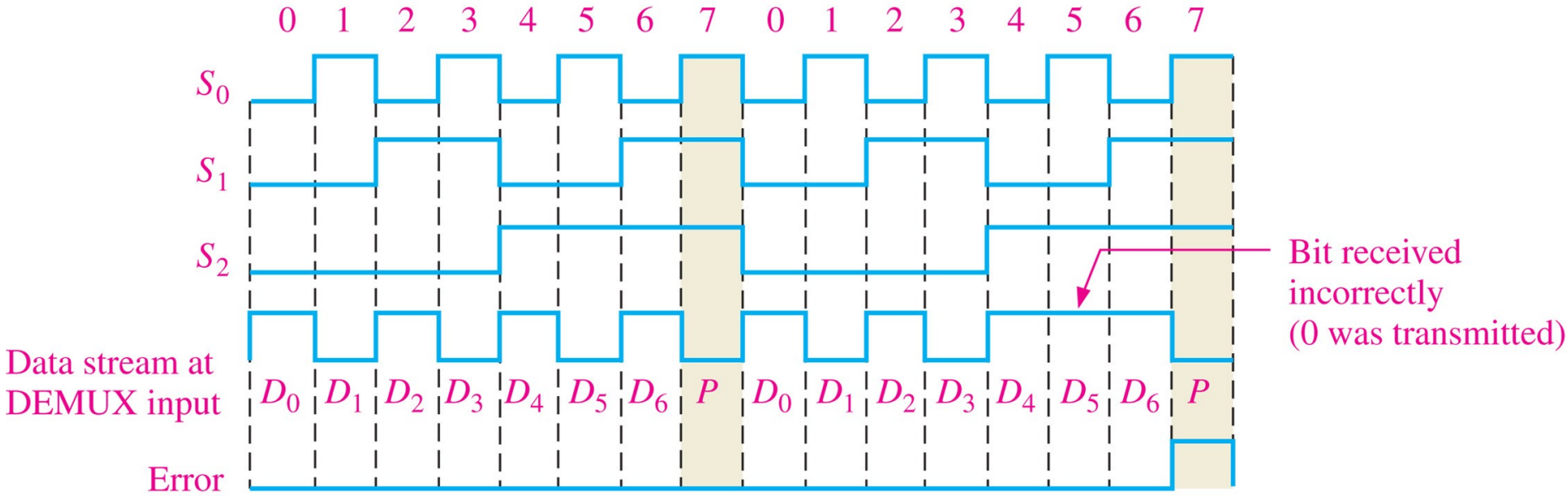
A Data Transmission System with Error Detection

- Digital data from seven sources (D_0-D_6) are multiplexed onto a single line for transmission
- The data-select inputs (S_0-S_2) are repeatedly cycled through a binary sequence, and each data bit, starting from D_0, is serially passed through and onto the transmission line $\bar{Y}$.
- At the demultiplexer, the data bits are distributed by the demultiplexer onto the output lines in the order in which they occurred on the multiplexer inputs.



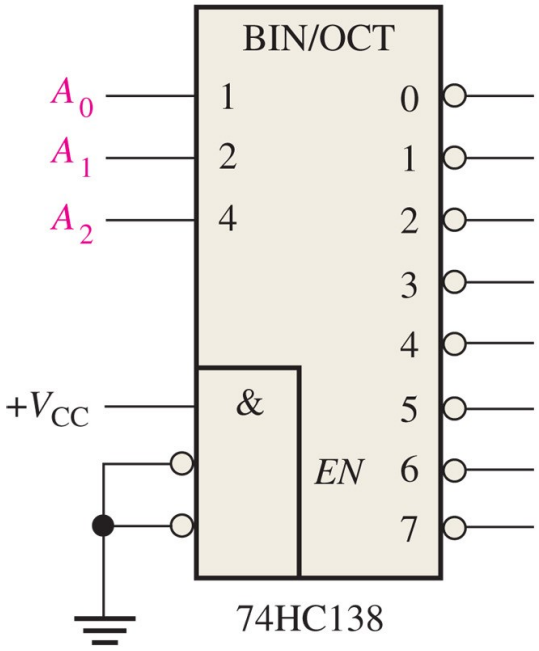
A Data Transmission System with Error Detection

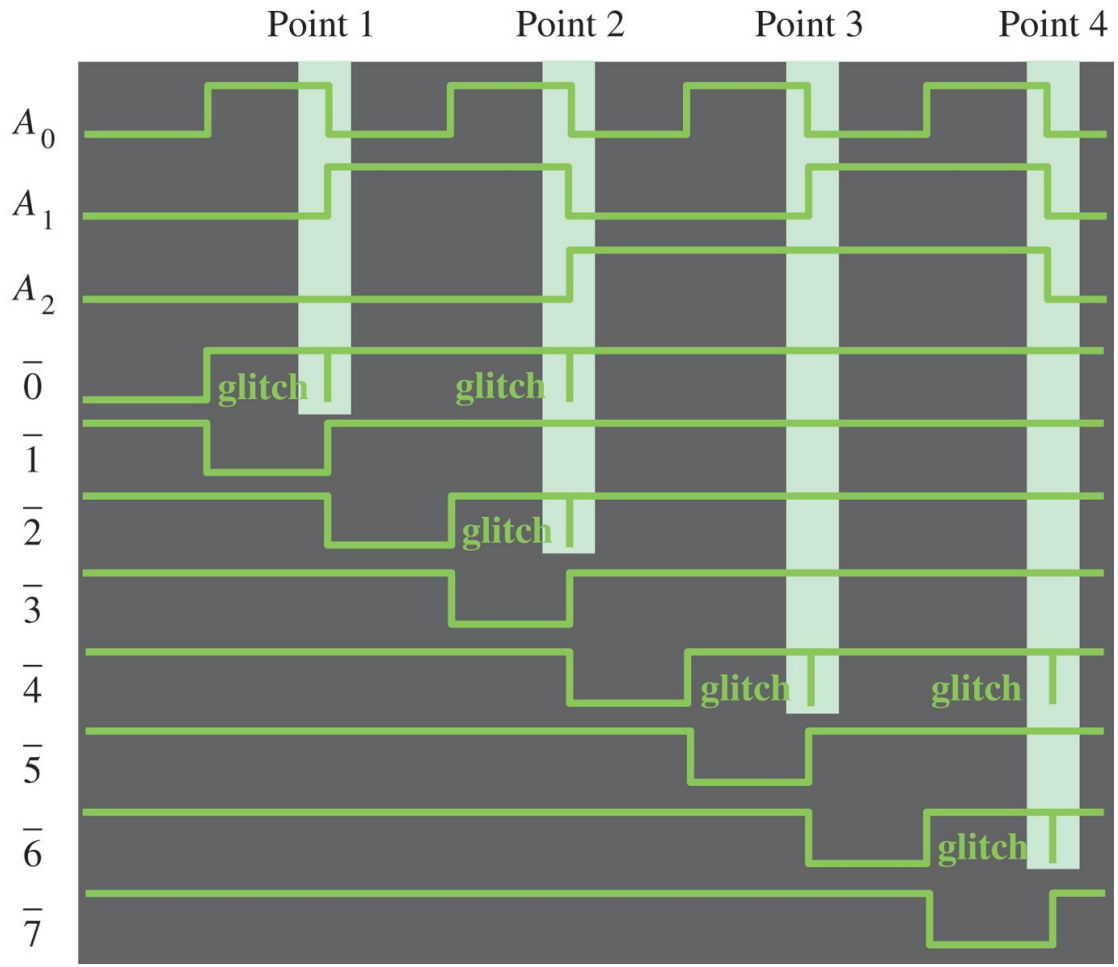
- The parity bit comes out on the D_7 output. These eight bits are temporarily stored and applied to the even parity checker.
- At this time, the error gate is enabled by the data-select code 111 (recall S0-S2 are repeatedly cycled from 000 to 111).
- If the parity is correct, a 0 appears on the Σ Odd output, keeping the Error output at 0.
- If the parity is incorrect, all 1s appear on the error gate inputs, and a 1 on the Error output results.



Troubleshooting : Glitches (I)

- A glitch is any undesired voltage or current spike (pulse) of very short duration. A glitch can be interpreted as a valid signal by a logic circuit and may cause improper operation.





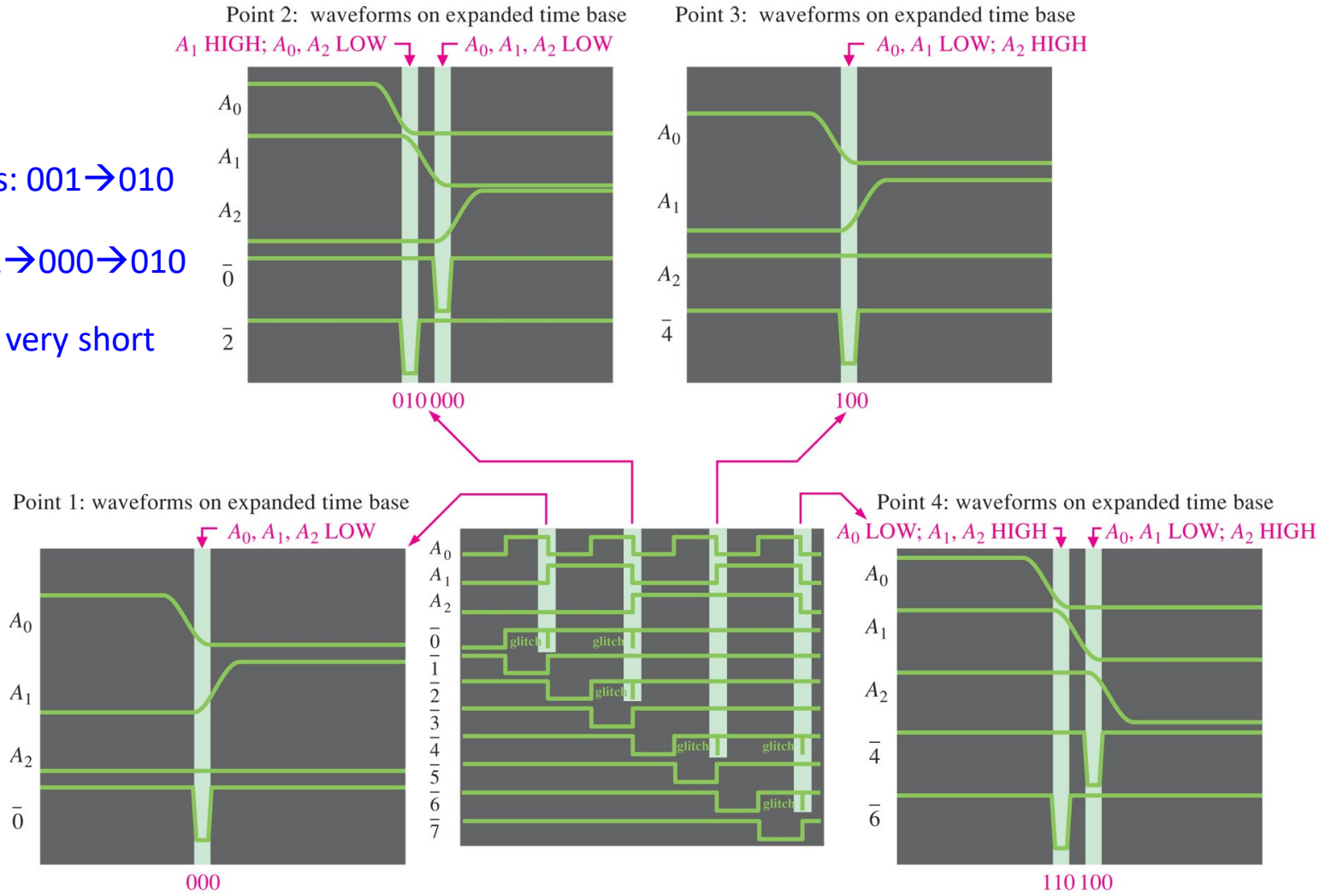
Troubleshooting : Glitches (II)

- At point 1, there is a transitional state of 000 due to delay differences in the waveforms. This causes the first glitch on the 0 output of the decoder.

Without glitches: 001→010

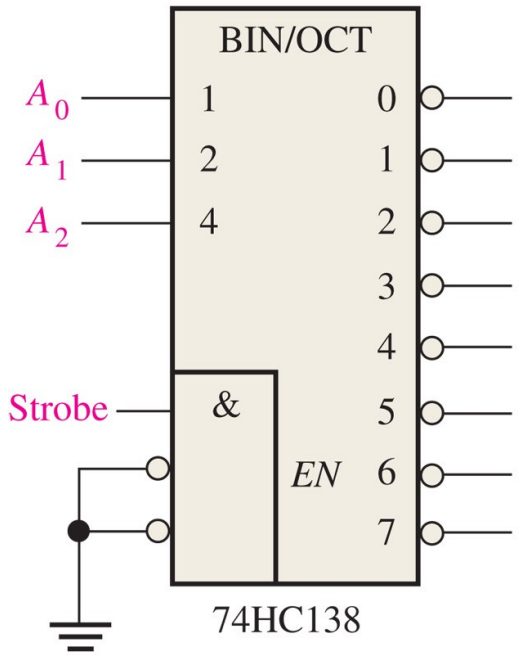
With glitch: 001→000→010

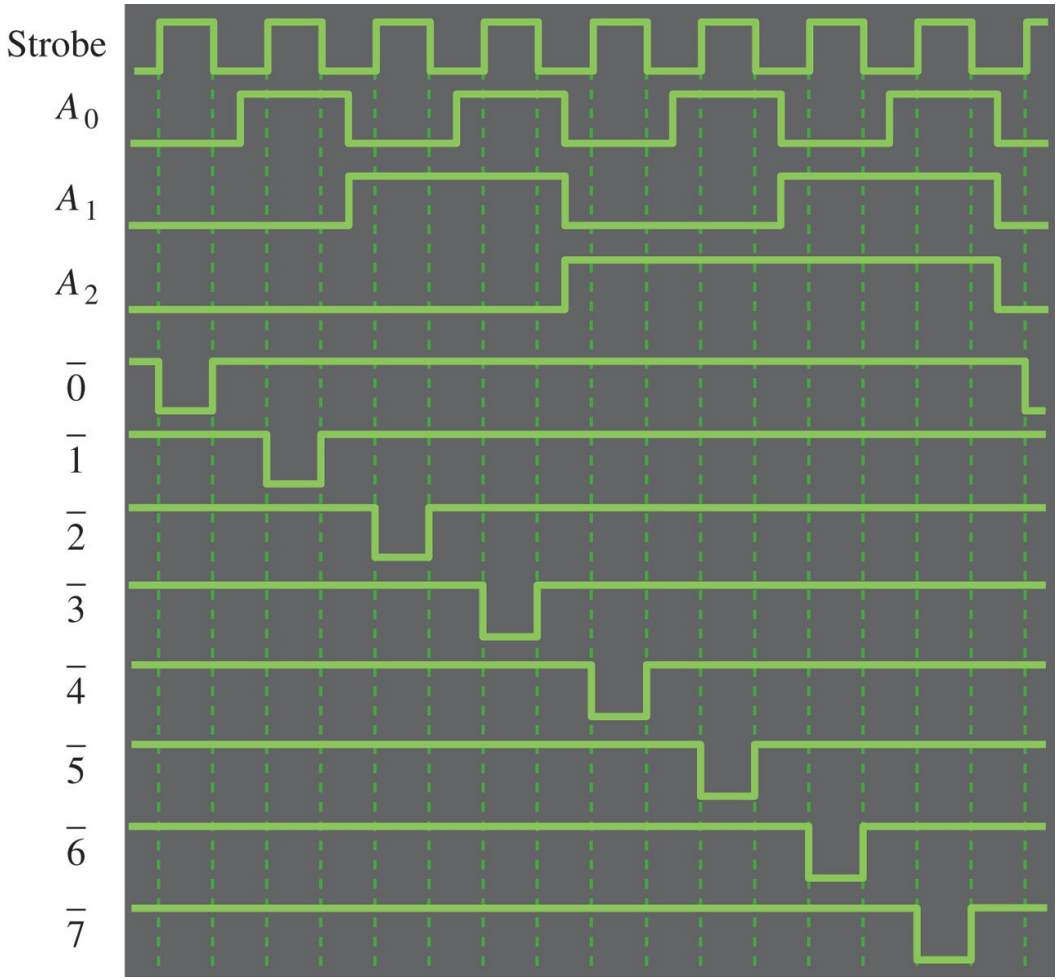
The 000 state is very short



Troubleshooting: Strobing

- Strobing: the decoder is enabled by a strobe (reference) pulse only during the times when the waveforms are not in transition.





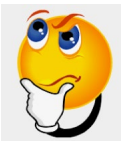
Applied Logic: Traffic Signal Controller

- Timing Requirements
  - The green light for the main street will stay on for a minimum of 25s or as long as there is no vehicle on the side street.
  - The green light for the side street will stay on until either (1) there is no vehicle on the side street or (2) a maximum duration of 25 s elapses .
  - The yellow caution light will stay on for 4 s between changes from green to red on both the main street and the side street.

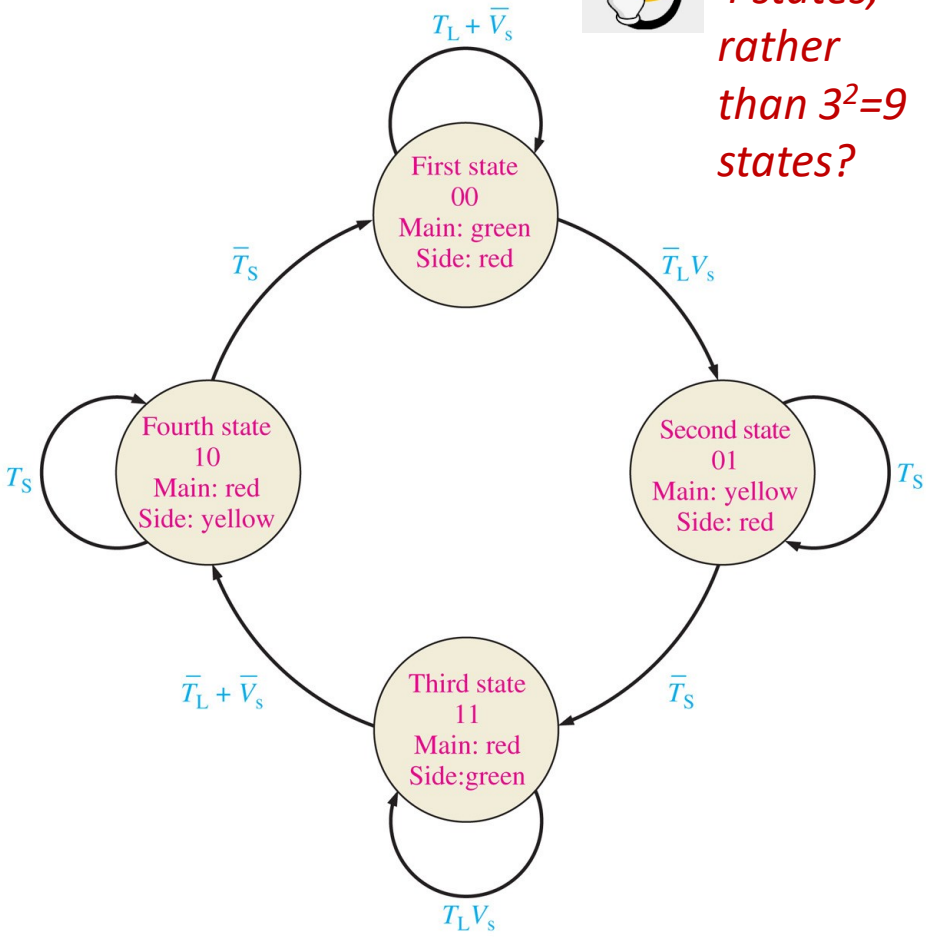
State Diagram

- **First State** : GREEN on the main street and RED on the side street for 25s when the long timer is on **or** there is no vehicle on the side street. The system transitions to the next state when the long timer goes off after 25s and there is a vehicle on the side street.
- **Second State** : YELLOW on the main street and RED on the side street. The system remains in this state for 4s when the short timer is on. The system transitions to the next state when the short timer goes off after 4s.
- **Third State** : RED on the main street and GREEN on the side street for 25s when the long timer is on as long as there is a vehicle on the side street. The system transitions to the next state when the long timer goes off after 25s or when there is no vehicle on the side street.
- **Fourth State** : RED on the main street and YELLOW on the side street. The system remains in this state for 4s when the short timer is on. The system transitions back to the first state when the short timer goes off after 4s.

V_s : A vehicle is present on the side street.
T_L : The 25 s timer is on.
T_S : The 4 s timer is on.

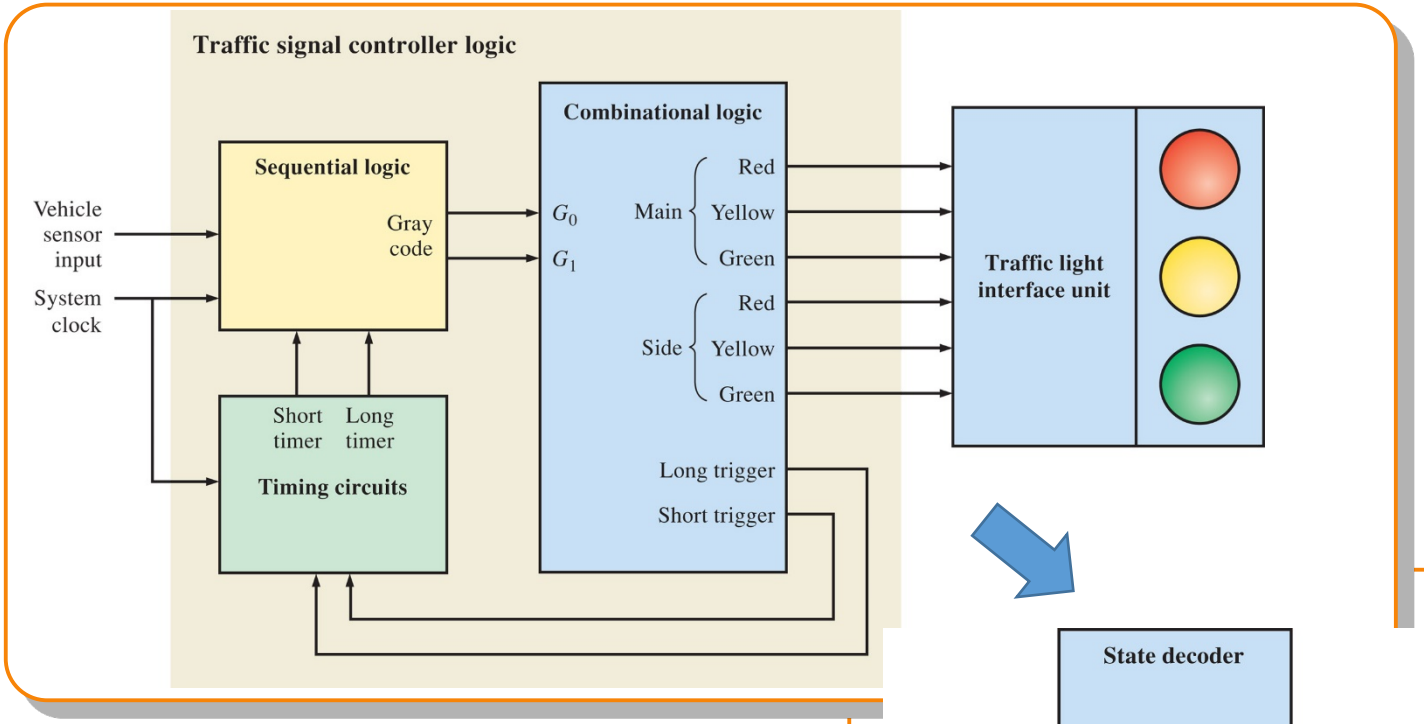


Why only 4 states, rather than 3^2=9 states?

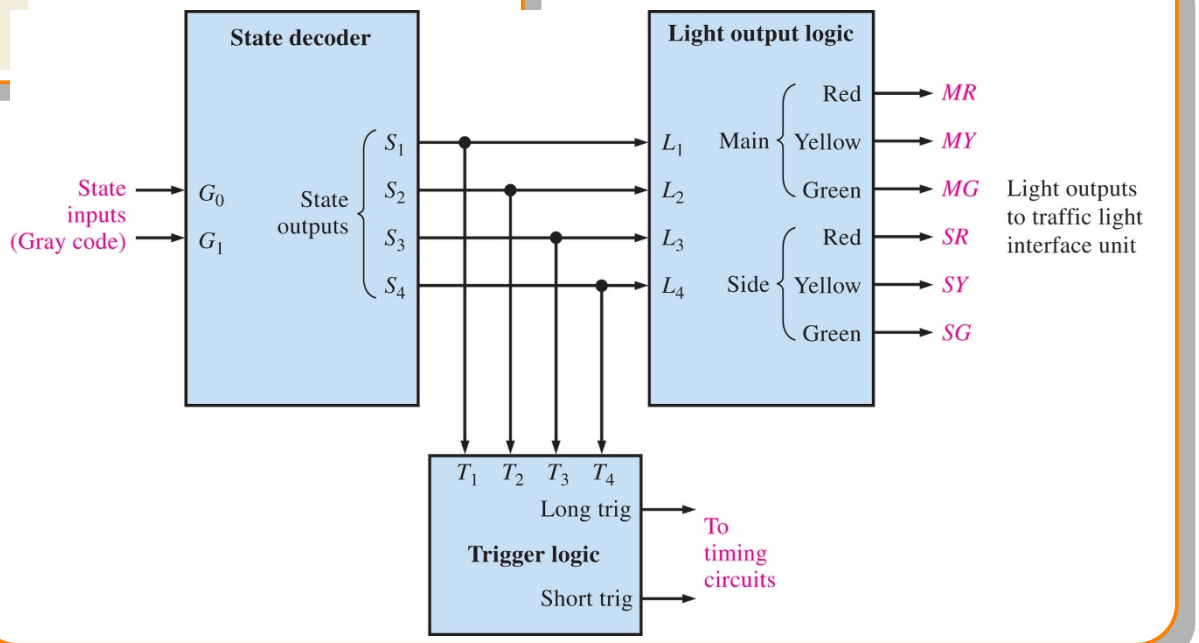


State diagram for the traffic signal control.

Block Diagram



Block diagram of the traffic signal controller.



Block diagram of the combinational logic unit.

The Combinational Logic

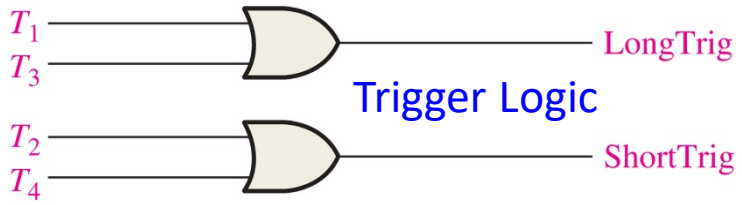
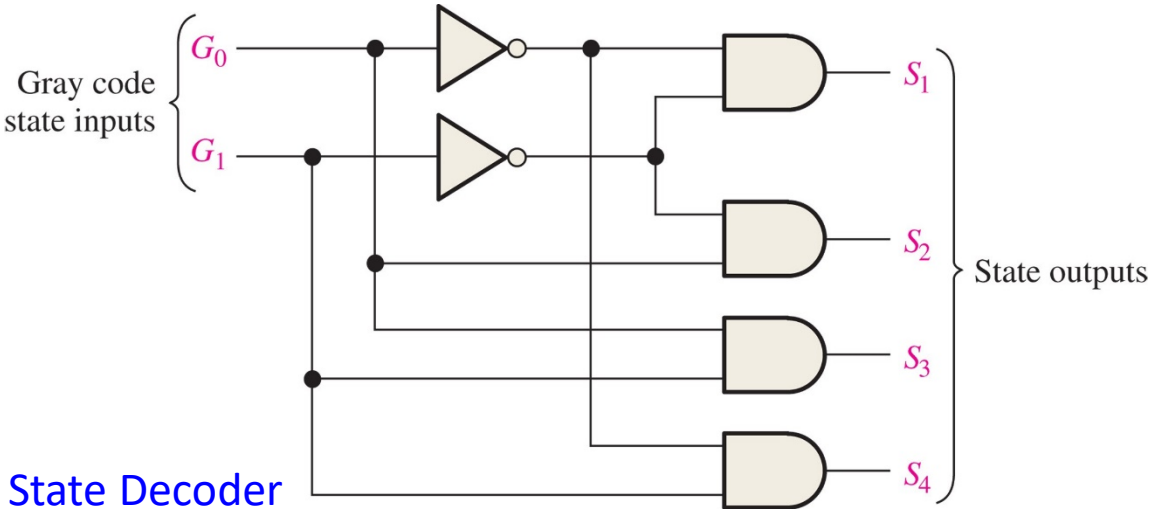
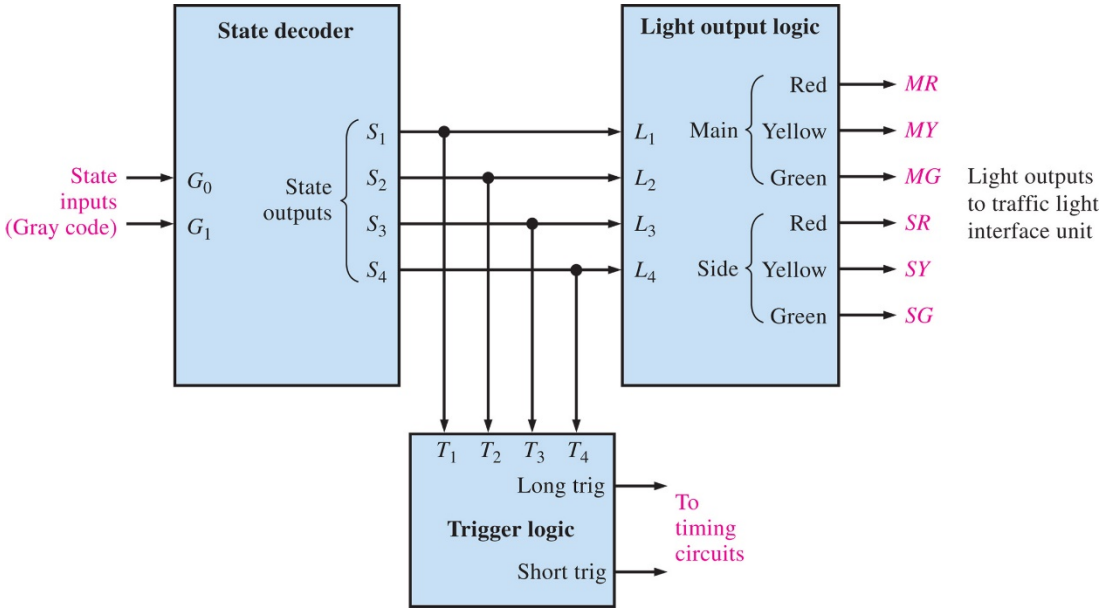


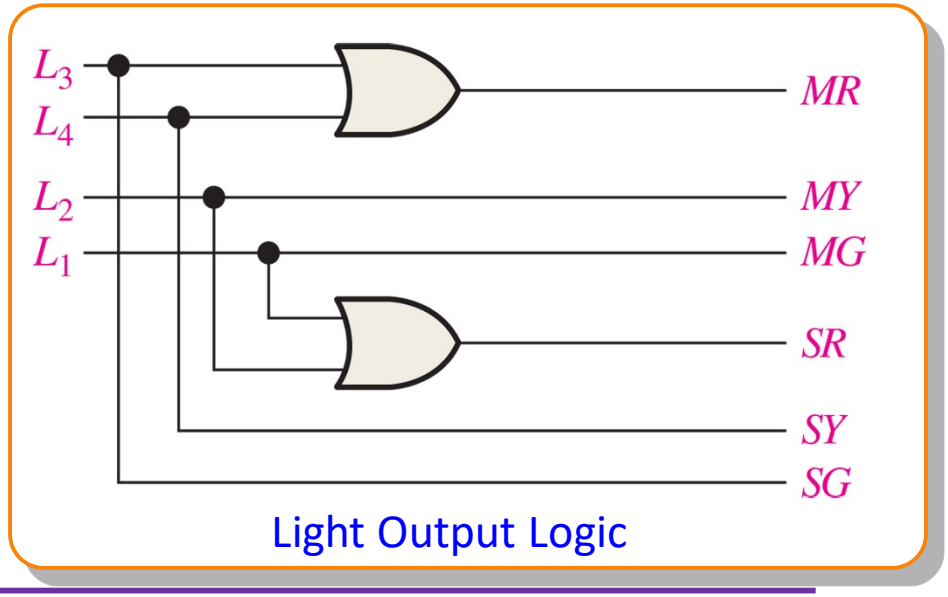
TABLE 6–11

Truth table for the state decoder.

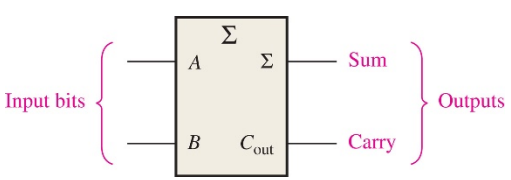
State Inputs (Gray Code)		State Outputs			
G_1	G_0	S_1	S_2	S_3	S_4
0	0	1	0	0	0
0	1	0	1	0	0
1	1	0	0	1	0
1	0	0	0	0	1

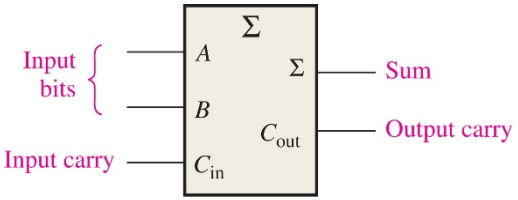


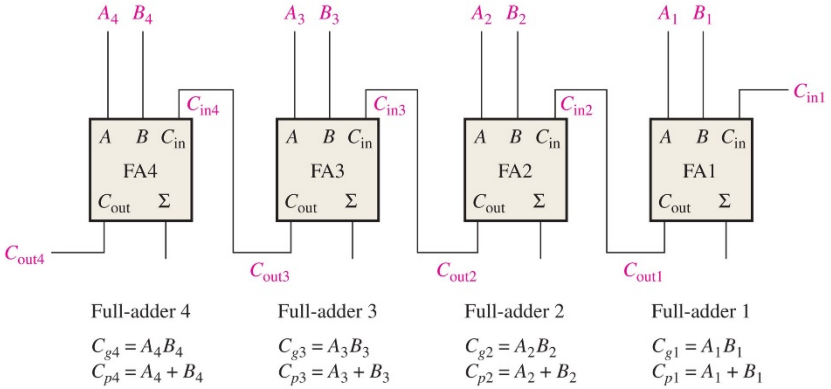




Chapter Review







- Half and Full Adders
- Parallel Binary Adders
- Ripple Carry and Look-Ahead Carry Adders
- Comparators
- Decoders (n input lines → max. 2^n output lines)
- Encoders (e.g. Decimal-to-BCD Priority Encoder)
- Code Converters (e.g. BCD-to-Binary Conversion)
- Multiplexers (To route several inputs onto a single output line)
- Demultiplexers
- Parity Generators/Checkers (Check for parity or generate a parity bit for a binary code)
- Troubleshooting (Glitches and Strobing)

True/False Quiz

- A half-adder adds two binary bits.
- A half-adder has a carry output only.
- A full adder adds two bits and produces two outputs.
- A full-adder can be realized only by using 2-input XOR gates.
- When the input bits are both 1 and the input carry bit is 1, the sum output of a full adder is 1.
- The output of a comparator is 0 when the two binary inputs given are equal.
- A decoder detects the presence of a specified combination of input bits.
- The 4-line-to-10-line decoder and the 1-of-10 decoder are two different types.
- An encoder essentially performs a reverse decoder function.
- A multiplexer is a logic circuit that allows digital information from a single source to be routed onto several lines.

True/False Quiz

- ✓ A half-adder adds two binary bits.
- ✗ A half-adder has a carry output only.
- ✗ A full adder adds two bits and produces two outputs.
- ✗ A full-adder can be realized only by using 2-input XOR gates.
- ✓ When the input bits are both 1 and the input carry bit is 1, the sum output of a full adder is 1.
- ✗ The output of a comparator is 0 when the two binary inputs given are equal.
- ✓ A decoder detects the presence of a specified combination of input bits.
- ✗ The 4-line-to-10-line decoder and the 1-of-10 decoder are two different types.
- ✓ An encoder essentially performs a reverse decoder function.
- ✗ A multiplexer is a logic circuit that allows digital information from a single source to be routed onto several lines.