
Chapter 4

Combinational Logic

Combinational logic circuits

- outputs logical functions of inputs
- new outputs appear shortly after changed inputs (propagation delay)
- no feedback loops
- no clock

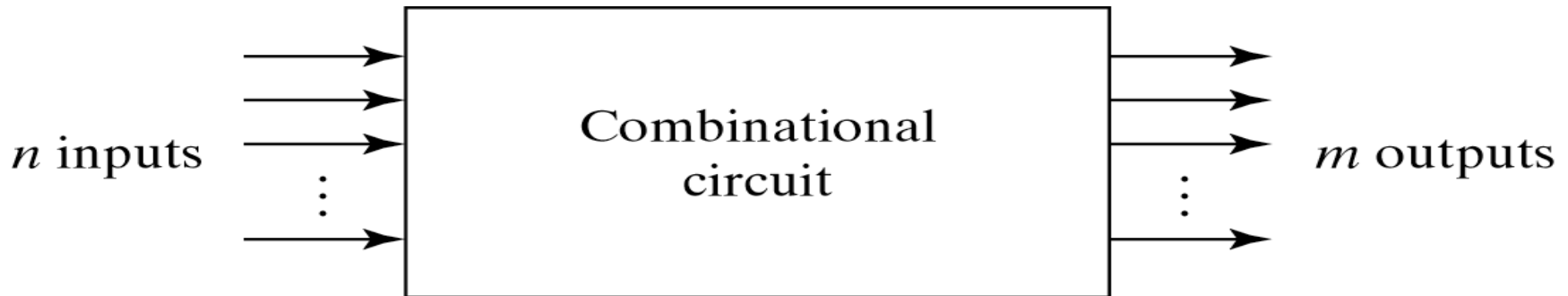


Fig. 4-1 Block Diagram of Combinational Circuit

Design Procedure

Design a circuit from a specification.

1. Determine number of required inputs and outputs.
2. Derive truth table
3. Obtain simplified Boolean functions
4. Draw logic diagram and verify correctness

Example: Code Converter

- A circuit that translates one binary code to another
- Example: **BCD to Excess-3 Code Converter**
 - Excess-3 code: decimal code + 3
 - BCD inputs 1010 to 1111 are don't care conditions

Decimal Digit	Input BCD				Output Excess-3			
	A	B	C	D	W	X	Y	Z
0	0	0	0	0	0	0	1	1
1	0	0	0	1	0	1	0	0
2	0	0	1	0	0	1	0	1
3	0	0	1	1	0	1	1	0
4	0	1	0	0	0	1	1	1
5	0	1	0	1	1	0	0	0
6	0	1	1	0	1	0	0	1
7	0	1	1	1	1	0	1	0
8	1	0	0	0	1	0	1	1
9	1	0	0	1	1	1	0	0

Simplification with K-map

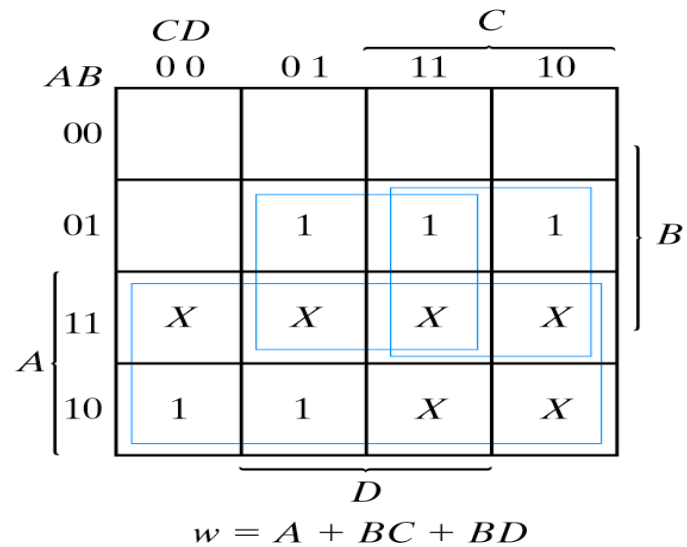
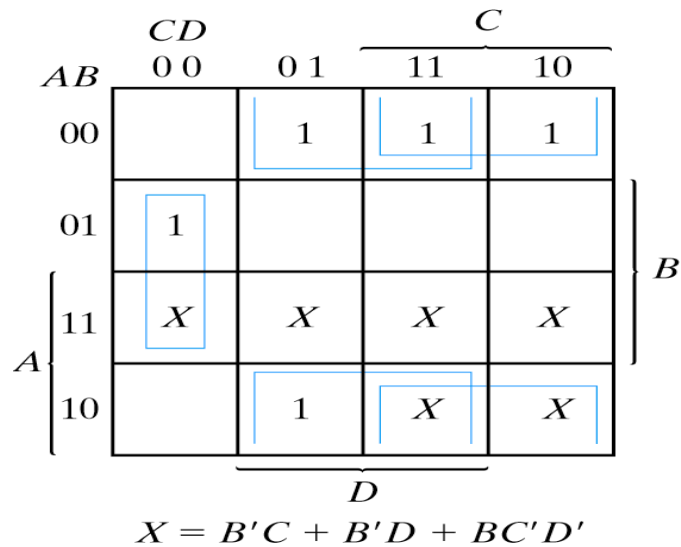
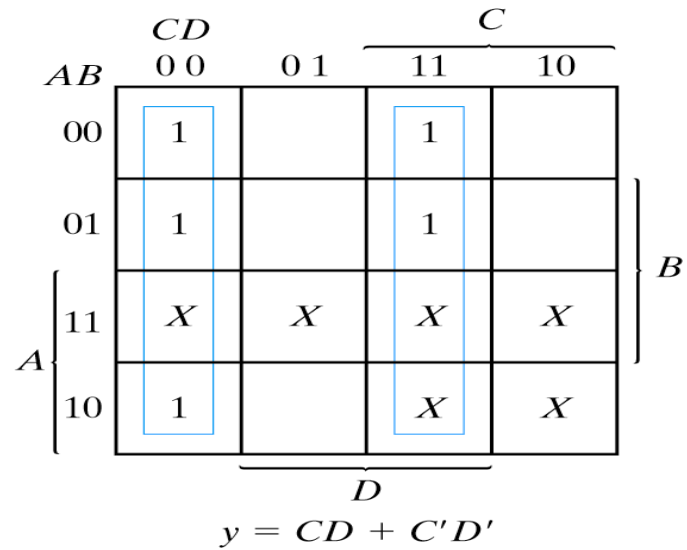
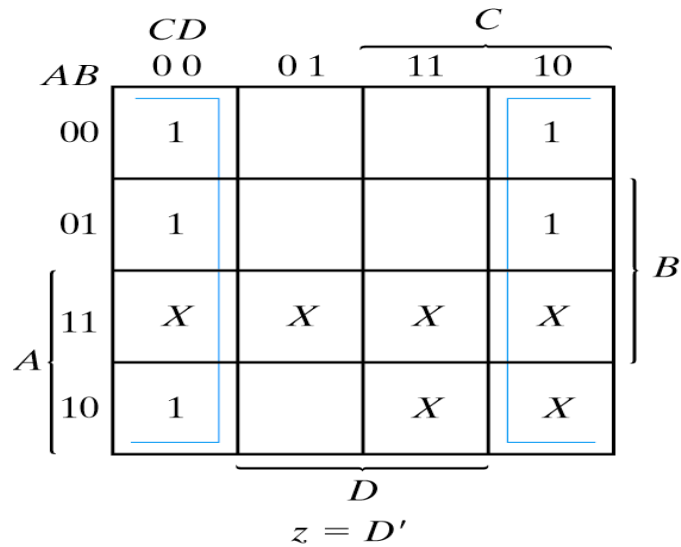


Fig. 4-3 Maps for BCD to Excess-3 Code Converter

Further manipulation of simplified expressions

- Two-level AND-OR implementation for the circuit can be obtained directly from the Boolean expression derived from the K-MAPS.
- Further manipulation can be done on the function to allow use of common gates for **multiple-output circuits**.
- Thus there are several possibilities for the implementation. The following shows the implementation with 3 levels of gates.

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

$$\begin{aligned} X &= B'C + B'D + BC'D' = B'(C + D) + BC'D' \\ &= B'(C+D) + B(C+D)' \end{aligned}$$

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

$$Z = D'$$

Three Level Implementation

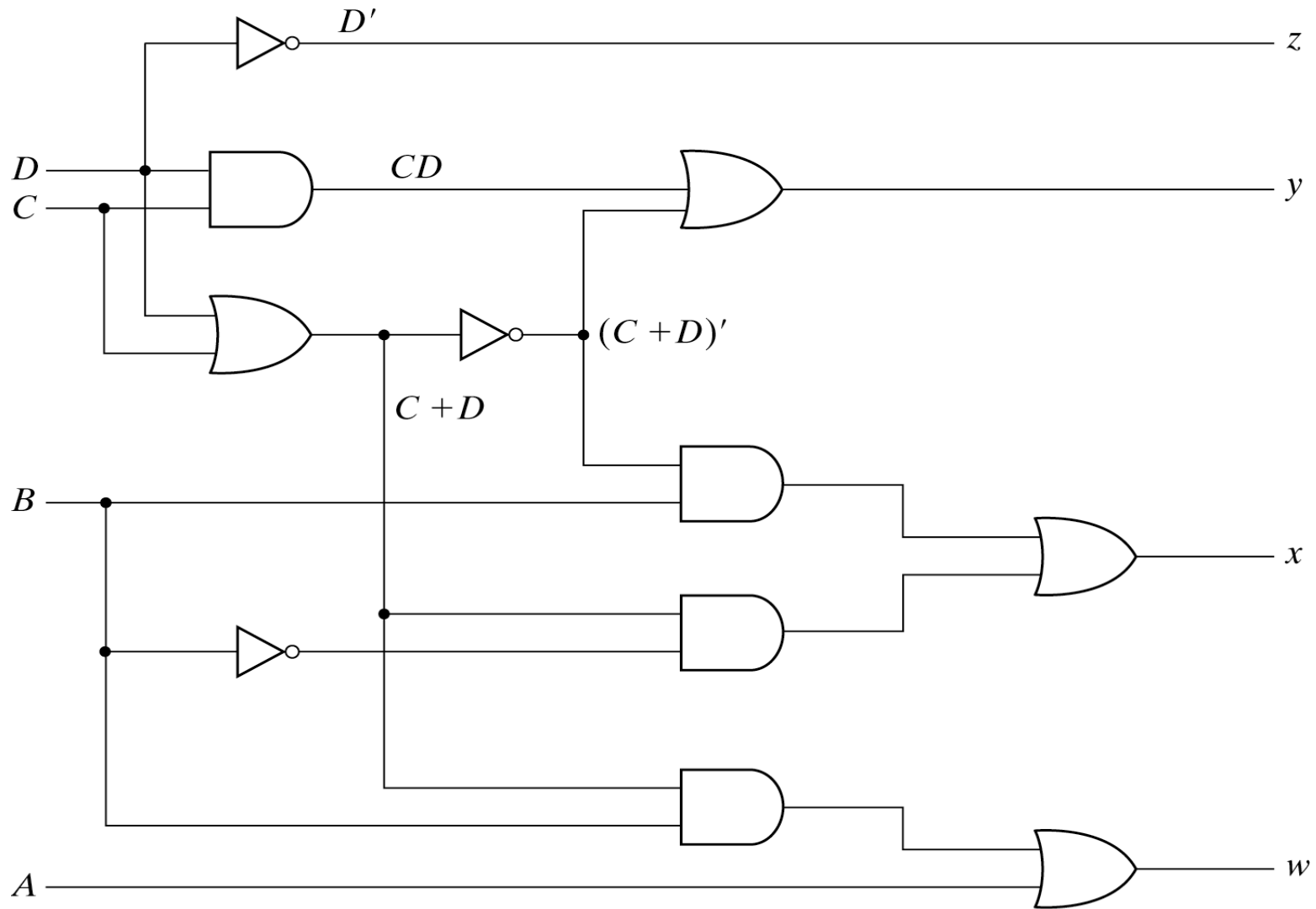


Fig. 4-4 Logic Diagram for BCD to Excess-3 Code Converter
ITI1100AA. Karmouch

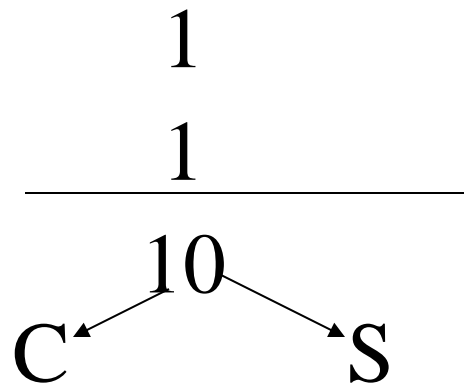
Binary Adder -Subtractor

Half Adder

→ The half-adder accepts two binary digits on its inputs and produces two binary digits on its outputs: a sum bit and a carry bit.

Truth Table

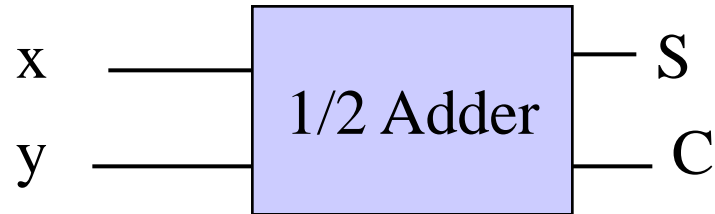
x	y	C	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0



Half-Adder (see other implementations in Chap. 2)

Truth Table

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

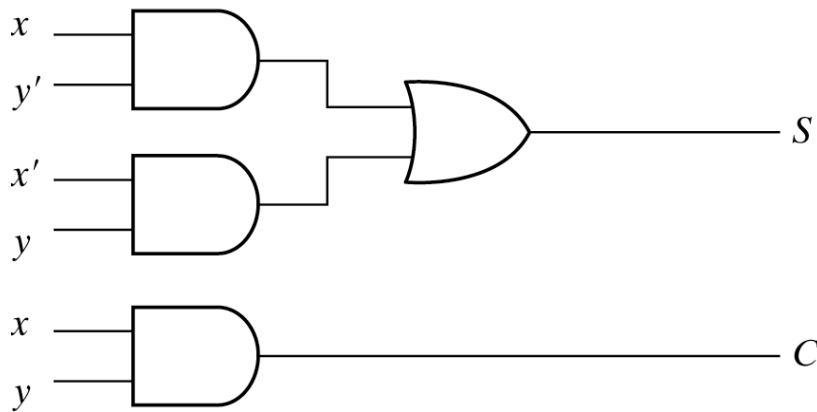


Some of products

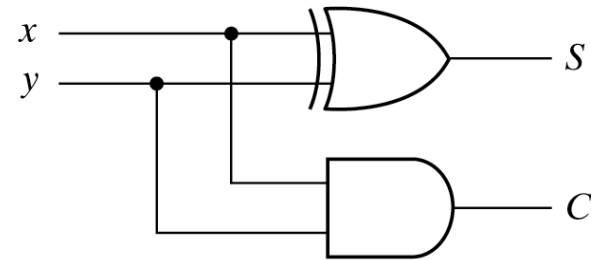
$$S = x'y + xy'$$

$$C = xy$$

Half-Adder-Implementation



$$(a) \begin{aligned} S &= xy' + x'y \\ C &= xy \end{aligned}$$



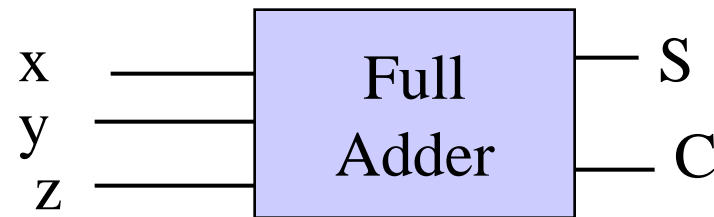
$$(b) \begin{aligned} S &= x \oplus y \\ C &= xy \end{aligned}$$

Fig. 4-5 Implementation of Half-Adder

Full Adder (see other implementations in Chap. 2)

Truth Table				
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

→ The Full-adder is combinational circuit



Full Adder (see other implementations in Chap. 2)

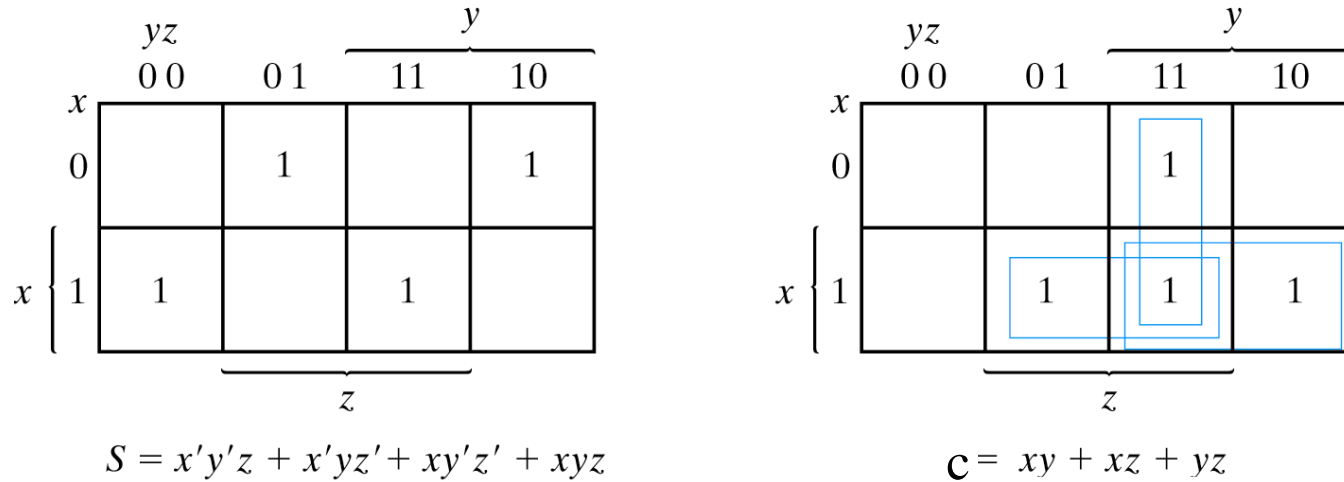


Fig. 4-6 Maps for Full Adder

Full Adder (see other implementations in Chap. 2)

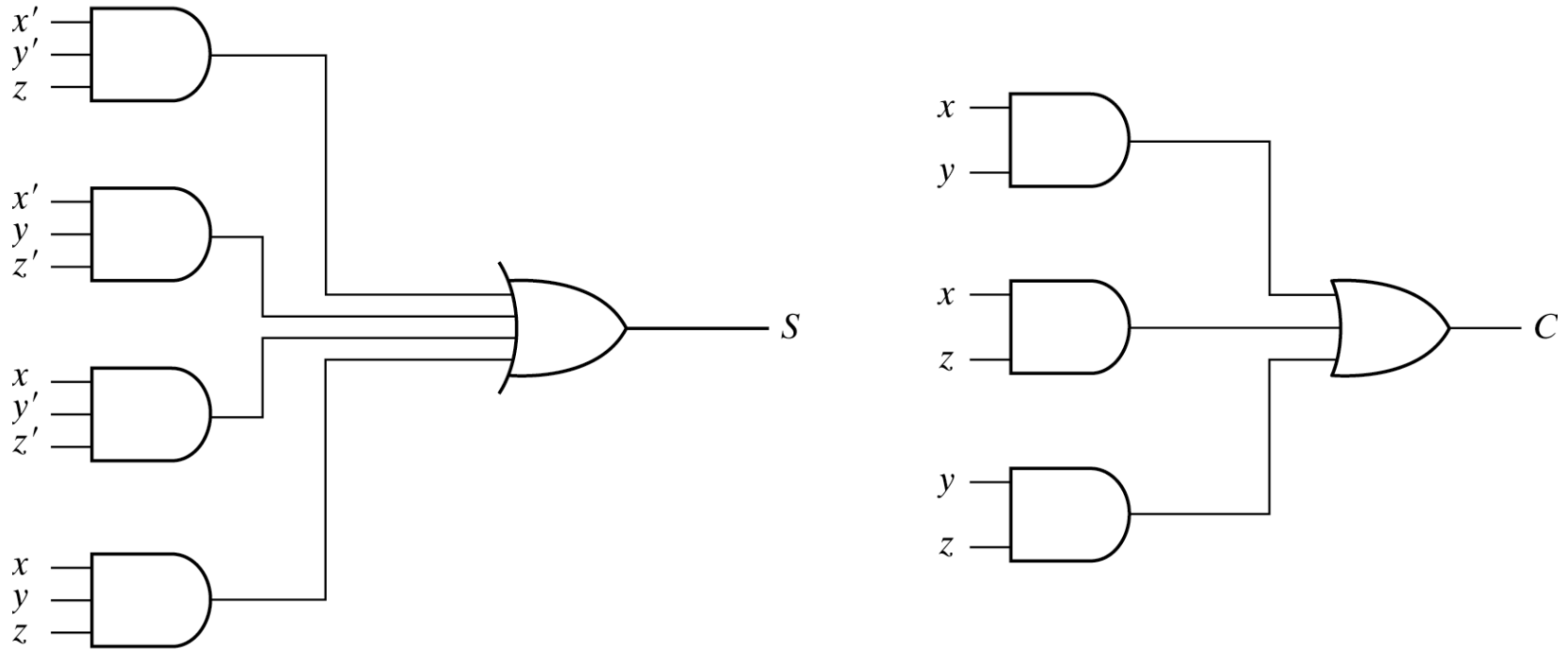


Fig. 4-7 Implementation of Full Adder in Sum of Products

Full Adder (same as in Chap. 2)

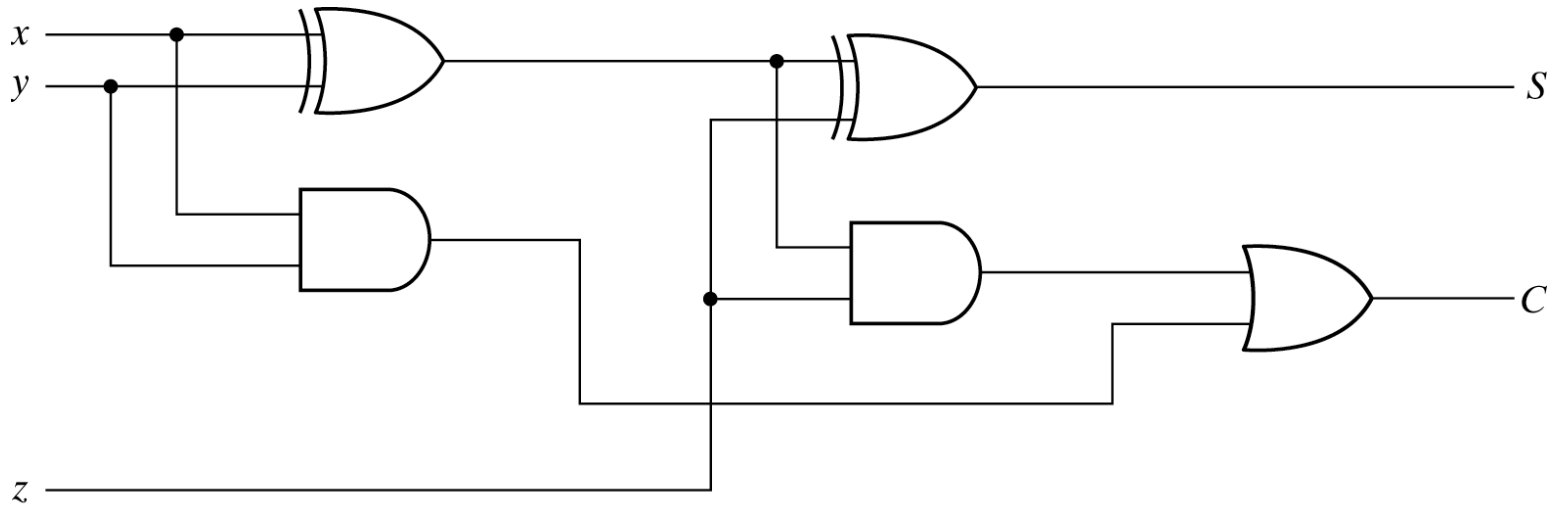
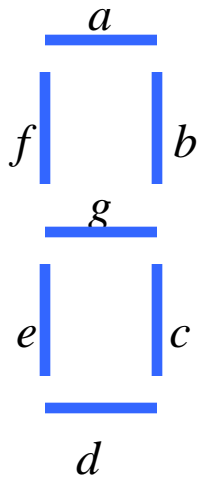


Fig. 4-8 Implementation of Full Adder with Two Half Adders and an OR Gate

Decoder -Example

a BCD to Seven Segment Decoder inputs data in BCD form and converts it to a seven segment output



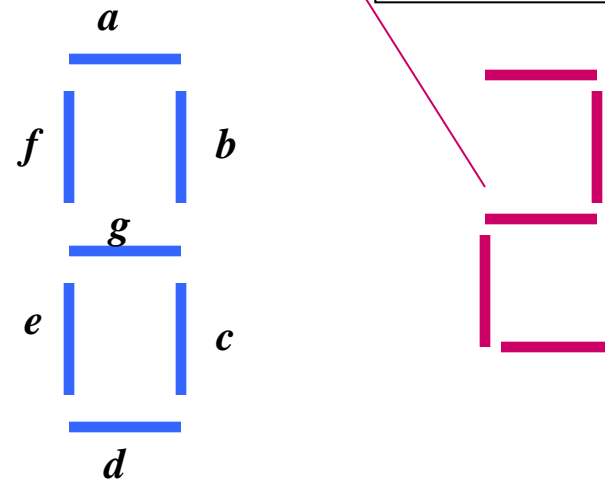
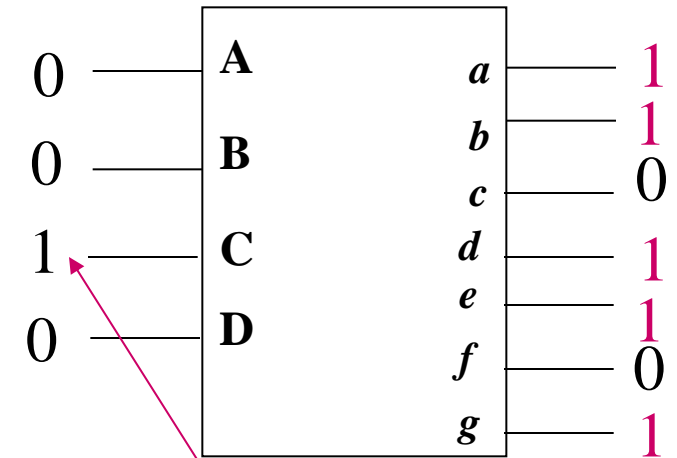
(a) Segment designation



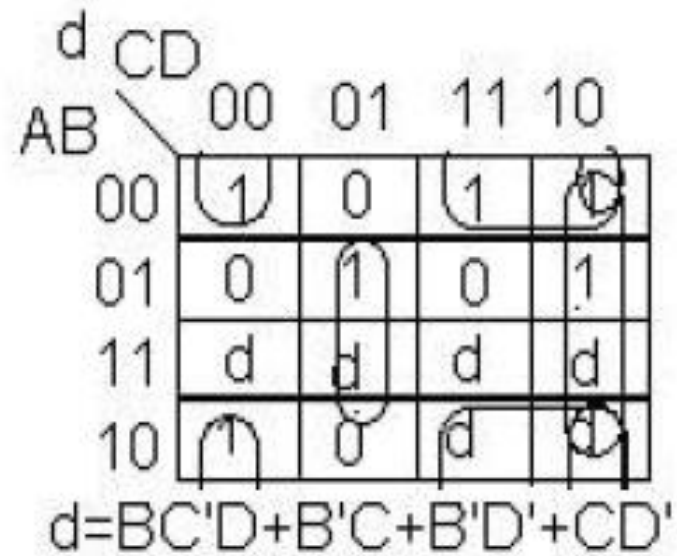
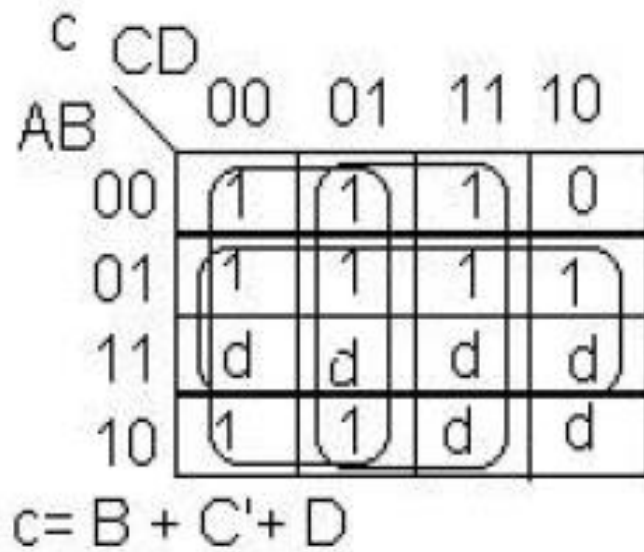
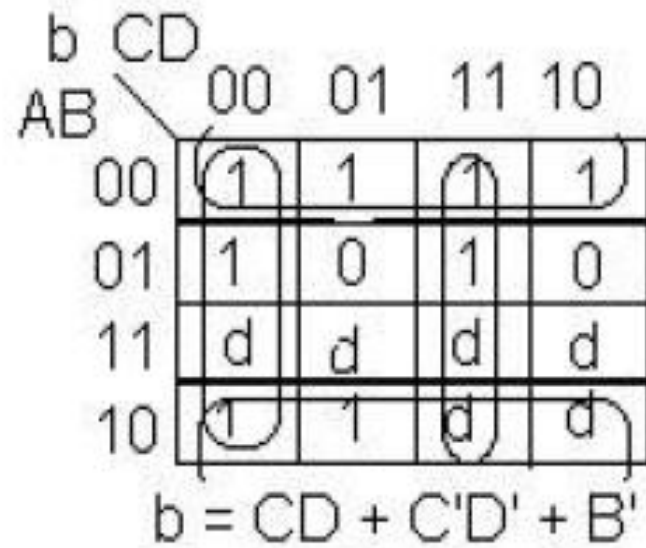
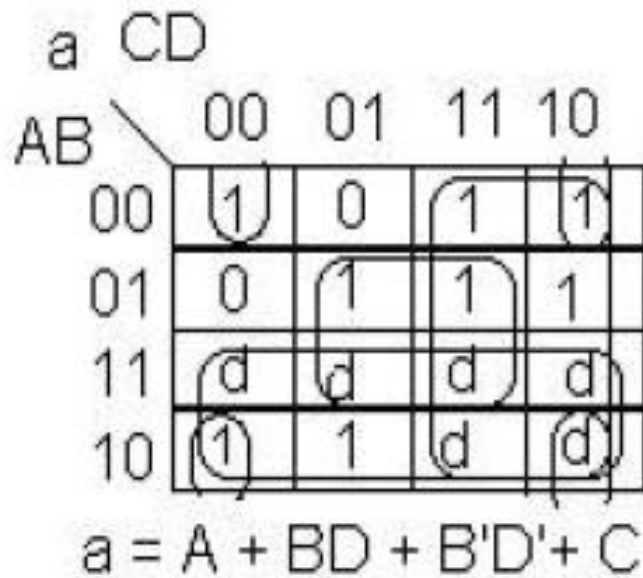
(b) Numerical designation for display

A- BCD to Seven Segment Decoder

A	B	C	D	a	b	c	d	e	f	g
0	0	0	0	1	1	1	1	1	1	0
0	0	0	1	0	1	1	0	0	0	0
0	0	1	0	1	1	0	1	1	0	1
0	0	1	1	1	1	1	1	0	0	1
0	1	0	0	0	1	1	0	0	1	1
0	1	0	1	1	0	1	1	0	1	1
0	1	1	0	1	0	1	1	1	1	1
0	1	1	1	1	1	1	0	0	0	0
1	0	0	0	1	1	1	1	1	1	1
1	0	0	1	1	1	1	1	0	1	1
1	0	1	0	x	x	x	x	x	x	x
1	0	1	1	x	x	x	x	x	x	x
1	1	0	0	x	x	x	x	x	x	x
1	1	0	1	x	x	x	x	x	x	x
1	1	1	0	x	x	x	x	x	x	x
1	1	1	1	x	x	x	x	x	x	x



Don't care terms

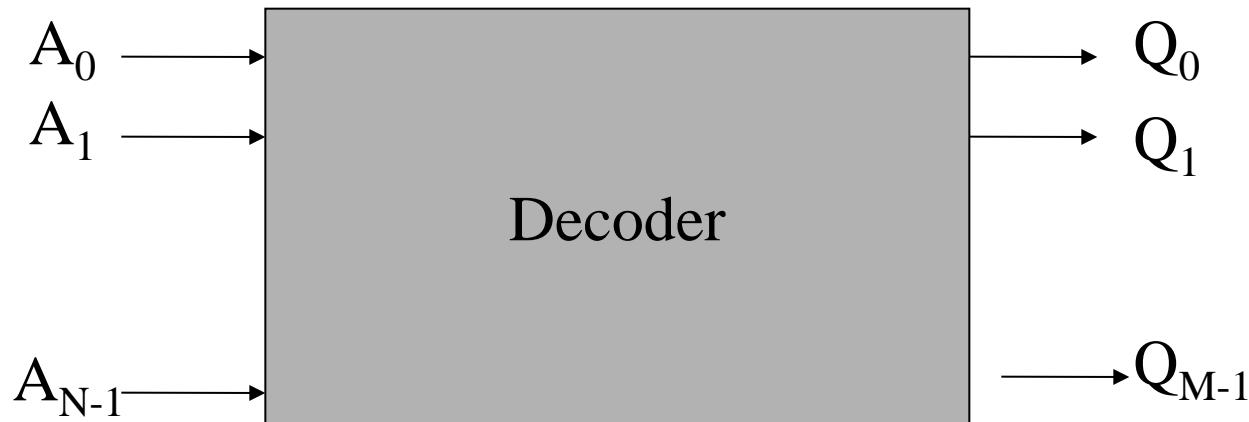


g		CD			
		00	01	11	10
AB	00	0	0	1	1
	01	1	1	0	1
	11	d	d	d	d
	10	1	1	d	d

$$g = BC' + B'C + CD' + A$$

Decoders

- A decoder is a combinational circuit that converts binary information from n input lines to a maximum of 2^n unique output lines. \rightarrow *n -to- 2^n decoder*
- If the n -bit coded information has unused combinations \rightarrow *less than 2^n outputs.*
 \rightarrow *n -to- m decoder, $m \leq 2^n$* , Example: *BCD-to-7-segment decoder, where $n=4$ and $m=7$*



2-to-4 Decoder

→ A 2-to-4 decoder operates according to the following truth table.

- The 2-bit input is called S_1S_0 , and the four outputs are Q_0 - Q_3 .
- If the input is the binary number i , then output Q_i is uniquely true.

S_1	S_0	Q_0	Q_1	Q_2	Q_3
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1

- For example, if the input $S_1 S_0 = 10$ (decimal 2), then output Q_2 is true, and Q_0 , Q_1 , Q_3 are all false.
- This logic circuit “decodes” a binary number into a “one-of-four” code.

Building a 2-to-4 decoder?

- Same design procedure as for the combinational logic circuit (see previous slides). From the truth table, we can derive equations for each of the four outputs (Q0-Q3), based on the two inputs (S0-S1).

S1	S0	Q0	Q1	Q2	Q3
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1

- There is no much to be simplified. the equations are:

$$Q0 = S1' S0'$$

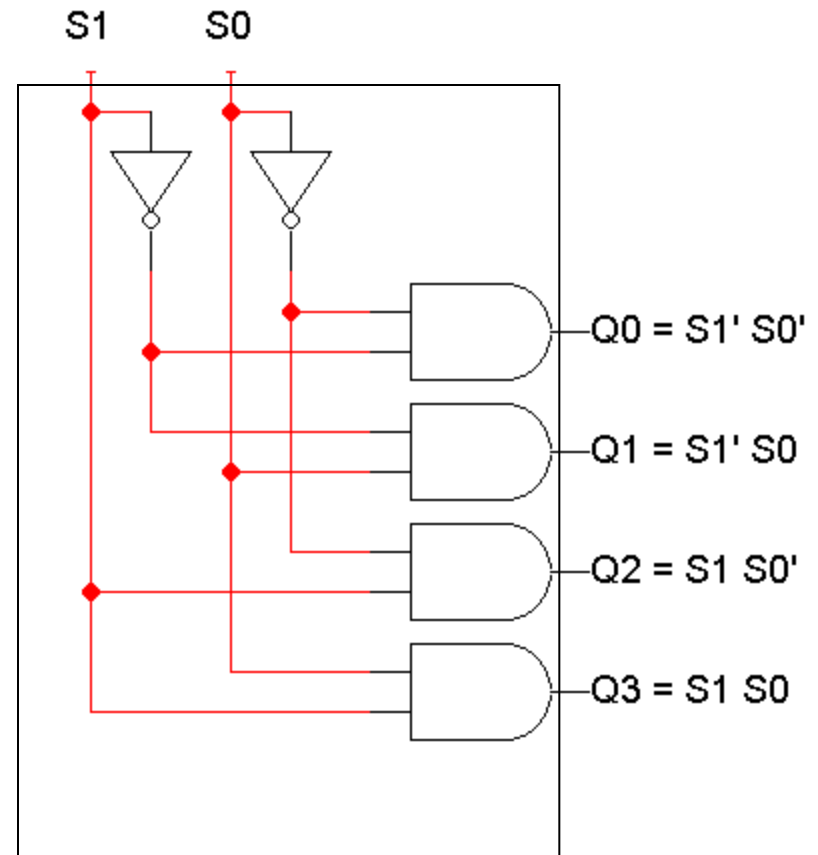
$$Q1 = S1' S0$$

$$Q2 = S1 S0'$$

$$Q3 = S1 S0$$

Implementation of 2-to-4 decoder

S1	S0	Q0	Q1	Q2	Q3
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1



Enable input

- Many devices have an additional **enable input**, which is used to “activate” or “deactivate” the device.
- For a decoder,
 - EN=1 activates the decoder, so it behaves as specified earlier. Exactly one of the outputs will be 1.
 - EN=0 “deactivates” the decoder. **By convention**, that means *all* of the decoder’s outputs are 0.
- We can include this additional input in the decoder’s truth table:

EN	S1	S0	Q0	Q1	Q2	Q3
0	0	0	0	0	0	0
0	0	1	0	0	0	0
0	1	0	0	0	0	0
0	1	1	0	0	0	0
1	0	0	1	0	0	0
1	0	1	0	1	0	0
1	1	0	0	0	1	0
1	1	1	0	0	0	1

abbreviated truth tables

- In this table, note that whenever $EN=0$, the outputs are always 0, *regardless* of inputs $S1$ and $S0$.

EN	S1	S0	Q0	Q1	Q2	Q3
0	0	0	0	0	0	0
0	0	1	0	0	0	0
0	1	0	0	0	0	0
0	1	1	0	0	0	0
1	0	0	1	0	0	0
1	0	1	0	1	0	0
1	1	0	0	0	1	0
1	1	1	0	0	0	1

- We can abbreviate the table by writing x's in the input columns for $S1$ and $S0$.

EN	S1	S0	Q0	Q1	Q2	Q3
0	x	x	0	0	0	0
1	0	0	1	0	0	0
1	0	1	0	1	0	0
1	1	0	0	0	1	0
1	1	1	0	0	0	1

Decoder- a Minterm Generator

S1	S0	Q0	Q1	Q2	Q3
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1

$$Q0 = S1' S0'$$

$$Q1 = S1' S0$$

$$Q2 = S1 S0'$$

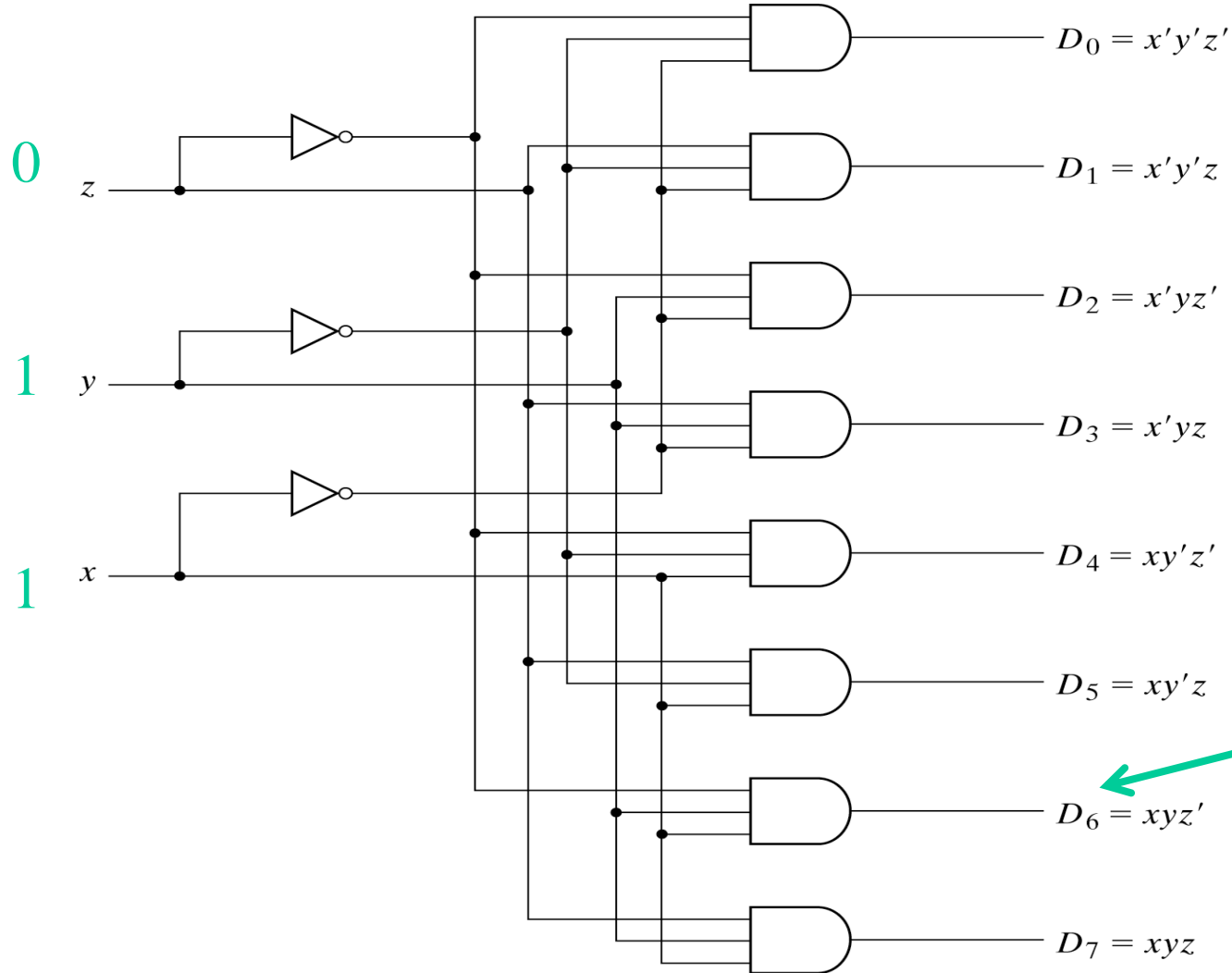
$$Q3 = S1 S0$$

- Decoders are sometimes called **minterm generators**.
 - For each of the input combinations, exactly one output is true.
 - Each output equation contains all of the input variables.
 - These properties hold for all sizes of decoders.
- Therefore we can implement arbitrary functions with decoders.
→ **from a sum of minterms equation** for a function, we can use a decoder (a **minterm generator**) to implement that function.

3- to- 8 line Decoder

- Three inputs, x , y , z , are decoded into eight outputs, D_0 through D_7
- Each output D_i represents one of the **minterms** of the 3 input variables.
- $D_i = 1$ when the binary number $xyz = i$
- Shorthand: $D_i = m_i$
- The output variables are mutually exclusive; exactly one output has the value 1 at any time, and the other seven are 0.

3- to- 8 line Decoder



D6 is selected
D6 = 1 all others = 0

Fig. 4-18 3-to-8-Line Decoder
ITI1100AA. Karmouch

Implementing Boolean Functions Using Decoders

- Any combinational circuit can be constructed using **decoders and OR gates!** Why?

→ **Here is an example:**

Implement a full adder circuit with a decoder and two OR gates.

Recall full adder equations, and let X, Y, and Z be the inputs:

$$S(X, Y, Z) = \Sigma m(1, 2, 4, 7)$$

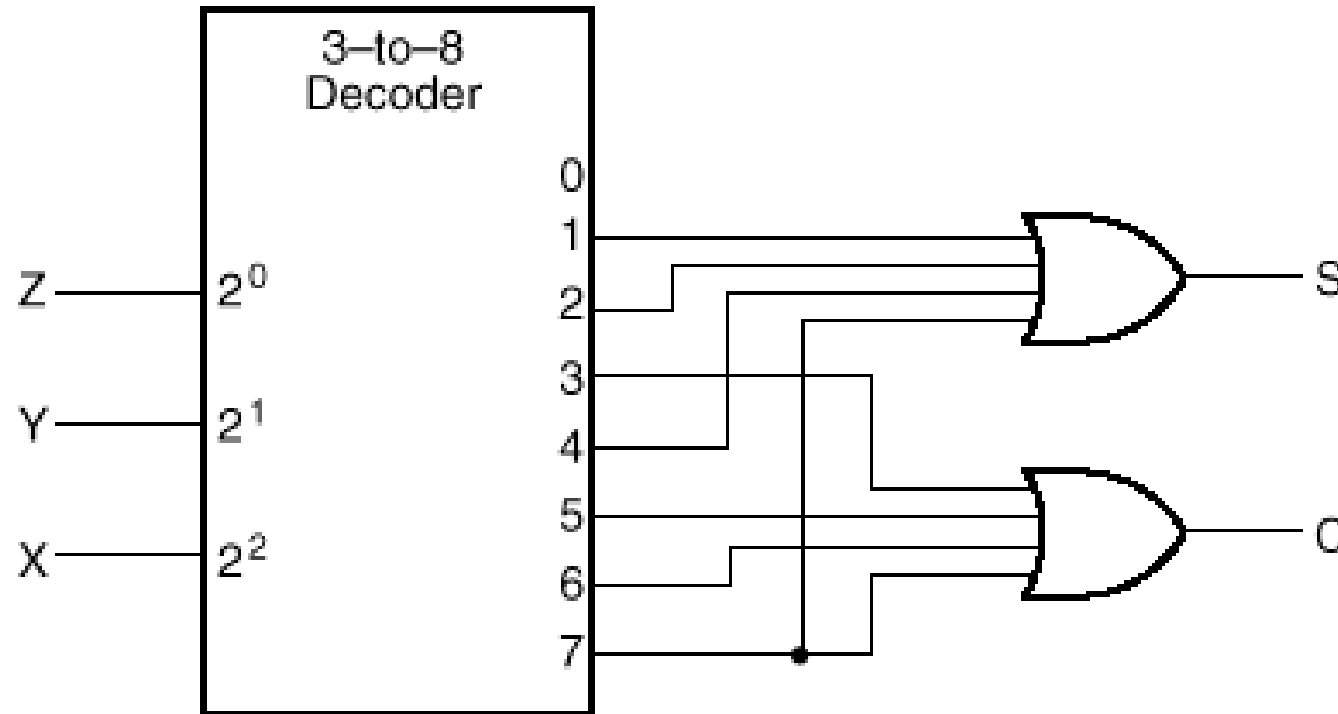
$$C(X, Y, Z) = \Sigma m(3, 5, 6, 7).$$

- **Since there are 3 inputs and a total of 8 minterms, we need a 3-to-8 decoder.**

Implementing Boolean Functions Using Decoders

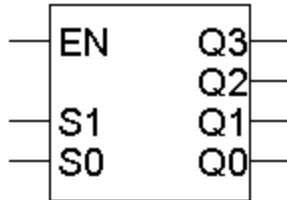
$$S(X,Y,Z) = \Sigma m(1,2,4,7)$$

$$C(X,Y,Z) = \Sigma m(3,5,6,7)$$



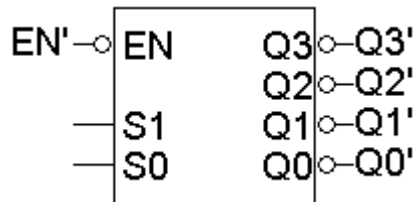
Implementing a decoder with NAND gates

- The decoders studied so far are **active-high** decoders (i.e. implemented **with AND gates**)



EN	S1	S0	Q0	Q1	Q2	Q3
0	x	x	0	0	0	0
1	0	0	1	0	0	0
1	0	1	0	1	0	0
1	1	0	0	0	1	0
1	1	1	0	0	0	1

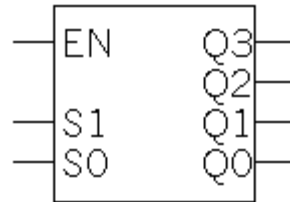
- Active-low decoders** are implemented **using NAND gates** (i.e. with an inverted EN input and inverted outputs).



EN	S1	S0	Q0	Q1	Q2	Q3
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
1	x	x	1	1	1	1

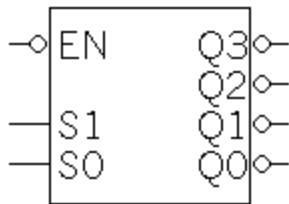
Active-High and Active-Low decoders

- Active-high decoders generate *minterms*, as we have already seen.



$$\begin{aligned}Q3 &= S1 S0 \\Q2 &= S1 S0' \\Q1 &= S1' S0 \\Q0 &= S1' S0'\end{aligned}$$

- Active-low decoders generate *Maxterms*.



$$\begin{aligned}Q3' &= (S1 S0)' = S1' + S0' \\Q2' &= (S1 S0')' = S1' + S0 \\Q1' &= (S1' S0)' = S1 + S0' \\Q0' &= (S1' S0')' = S1 + S0\end{aligned}$$

Building a 3-to-8 decoder with two 2-to-4 decoders

- Another way to design a 3-to-8 decoder is to use two 2-to-4 decoders.
- from the truth table of 3-8 decoder we can notice some patterns:
 - When $S_2 = 0$, outputs Q_0 - Q_3 are generated as in a 2-to-4 decoder.
 - When $S_2 = 1$, outputs Q_4 - Q_7 are generated as in a 2-to-4 decoder.
- S_2 can be used as an **Enable input** that activates/deactivates the 2-to-4 decoders.

S_2	S_1	S_0	Q_0	Q_1	Q_2	Q_3	Q_4	Q_5	Q_6	Q_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

$$Q_0 = S_2' S_1' S_0' = m_0$$

$$Q_1 = S_2' S_1' S_0 = m_1$$

$$Q_2 = S_2' S_1 S_0' = m_2$$

$$Q_3 = S_2' S_1 S_0 = m_3$$

$$Q_4 = S_2 S_1' S_0' = m_4$$

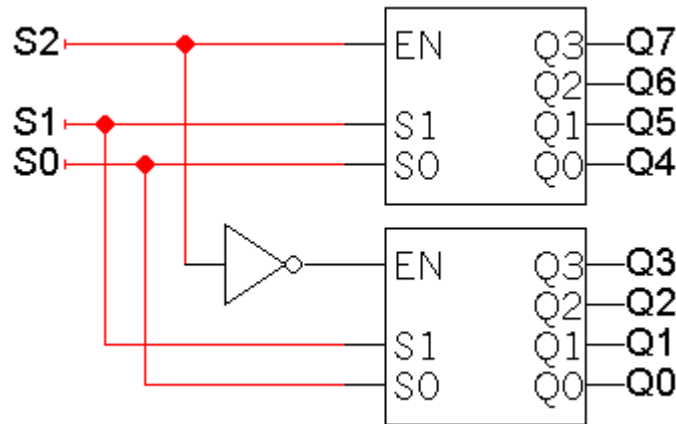
$$Q_5 = S_2 S_1' S_0 = m_5$$

$$Q_6 = S_2 S_1 S_0' = m_6$$

$$Q_7 = S_2 S_1 S_0 = m_7$$

Building a 3-to-8 decoder with two 2-to-4 decoders

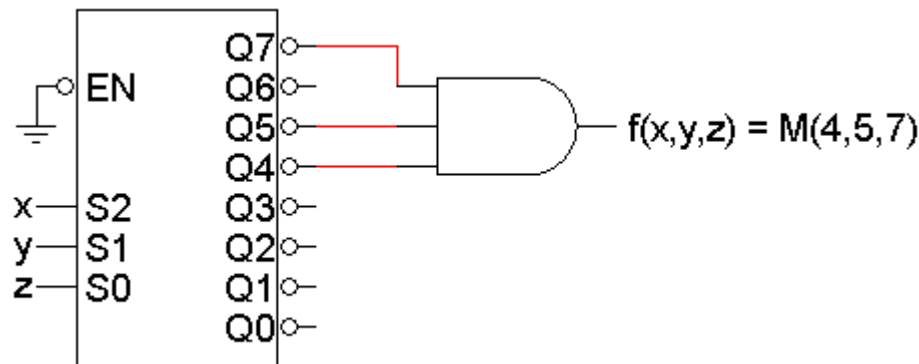
- We can use enable inputs to string decoders together. 3-to-8 decoder constructed from two 2-to-4 decoders:



S2	S1	S0	Q0	Q1	Q2	Q3	Q4	Q5	Q6	Q7
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

Active-low decoder example

- We can use active-low decoders to implement arbitrary functions as a product of maxterms.
- For example, here is an implementation of the function $f(x,y,z) = \prod M(4,5,7)$, using an active-low decoder.



- The “ground” symbol connected to EN represents logical 0, so this decoder is always enabled.
- We need an AND gate for a product of sums.

Decoder Expansions

- Larger decoders can be constructed using a number of smaller ones.

→ HIERARCHICAL design

- *Example:*

A 6-to-64 decoder can be designed using **four 4-to-16** and **one 2-to-4** decoders. How? (**tip:** Use the 2-to-4 decoder to generate the enable signals to the four 4-to-16 decoders).

4-input tree decoder

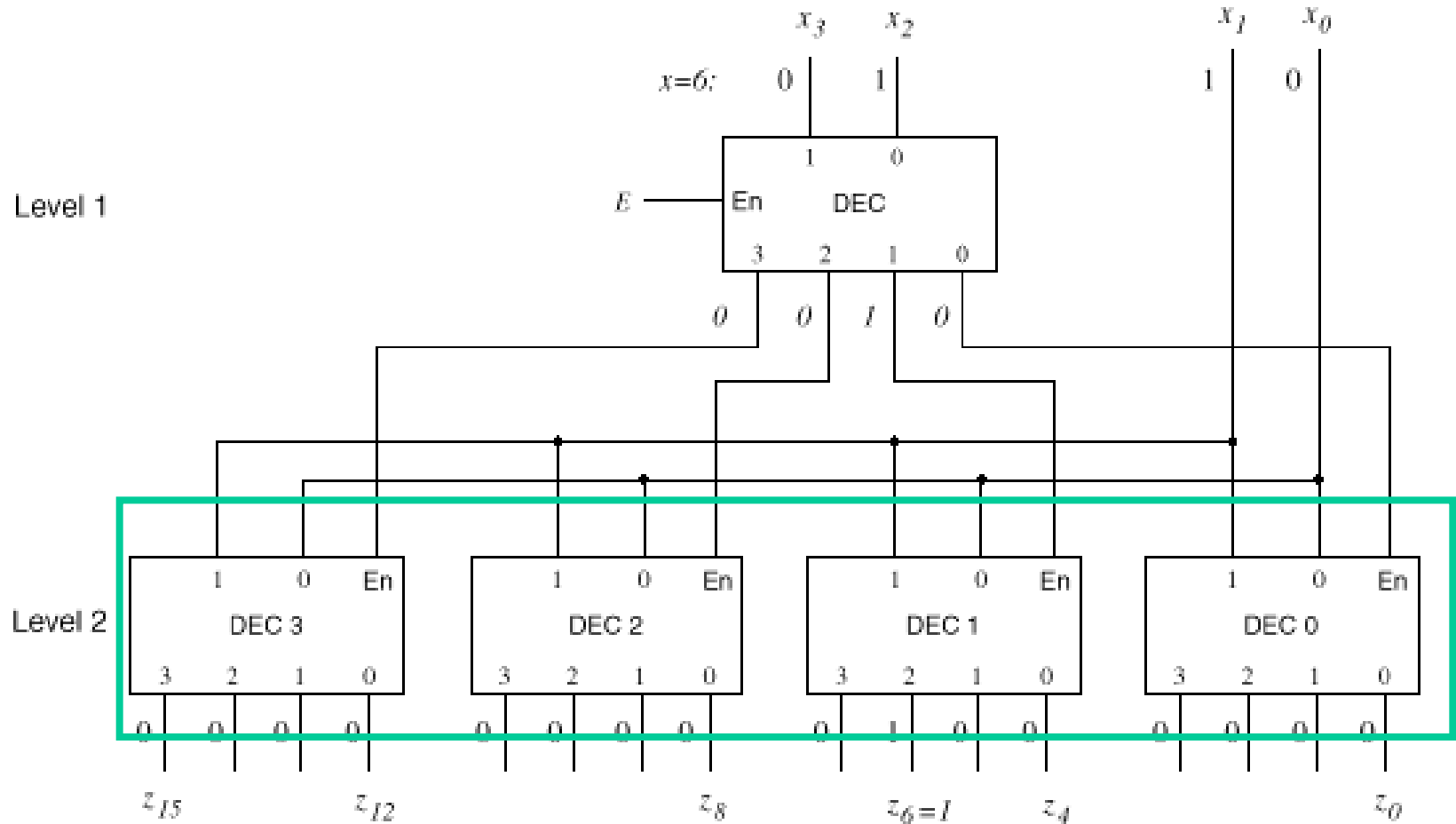


Figure 9.8: 4- nput tree decoder

Binary Encoder

- An encoder has a number of input lines, only one of which is activated at a given time.
- The opposite of the decoding process.
- Takes ALL its data inputs one at a time and then converts the one whose value is equal to "1" into a single encoded output

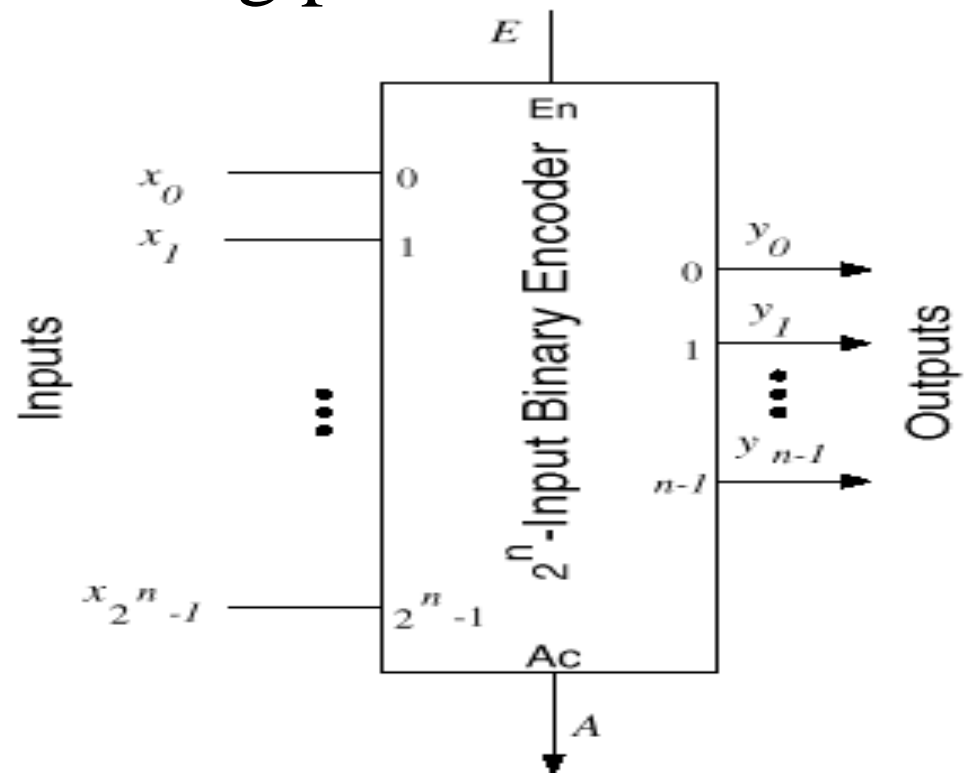


Figure 9.12: 2^n -input binary encoder.

Binary Encoder

Example: 8-to-3 Encoder

x_7	x_6	x_5	x_4	x_3	x_2	x_1	x_0	y_2	y_1	y_0
0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	1	0	0	0	1
0	0	0	0	0	1	0	0	0	1	0
0	0	0	0	1	0	0	0	0	1	1
0	0	0	1	0	0	0	0	1	0	0
0	0	1	0	0	0	0	0	1	0	1
0	1	0	0	0	0	0	0	1	1	0
1	0	0	0	0	0	0	0	1	1	1

Binary Encoder

- In the previous truth table each line selected (line 0 through 7) generates its own binary code such as a 1 is a 001, 5 is a 101 and so on.
- Boolean functions for Outputs

$$y_2 = x_4 + x_5 + x_6 + x_7$$

$$y_1 = x_2 + x_3 + x_6 + x_7$$

$$y_0 = x_1 + x_3 + x_5 + x_7$$

Simple Encoder Design Issues

- There are two ambiguities associated with the design of a simple encoder:
 - Only one input can be active at any given time. If two inputs are active simultaneously, the output produces an undefined combination (for example, if x_3 and x_6 are 1 simultaneously, the output of the encoder will be 111. (100 and 011))
 - An output with all 0's can be generated when all the inputs are 0's, or when x_0 is equal to 1.

Priority Encoders

- Solves the ambiguities multiple assigned inputs are allowed; **one** has **priority** over all others.
- **Separate** indication of **not** assigned inputs (**all** inputs are 0s).

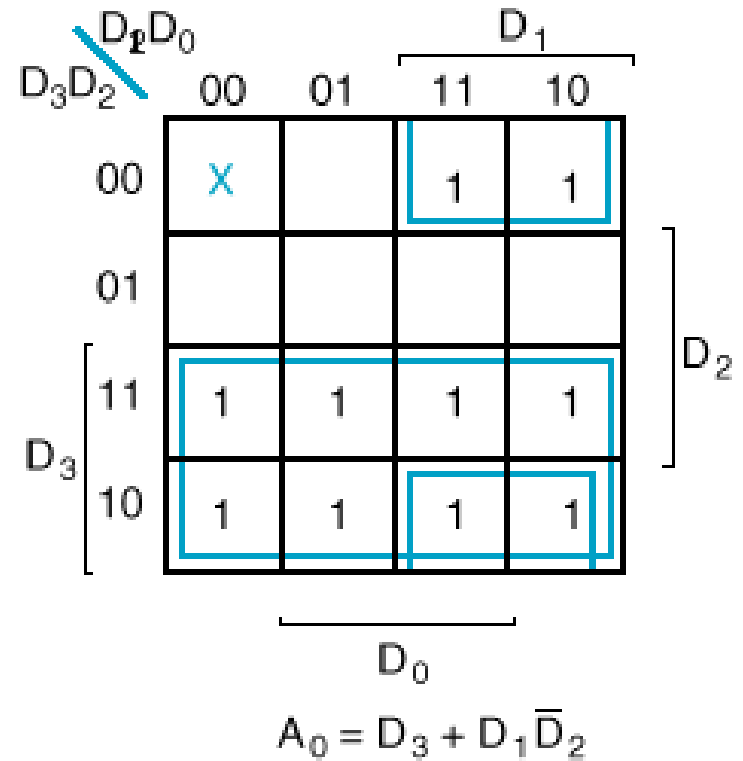
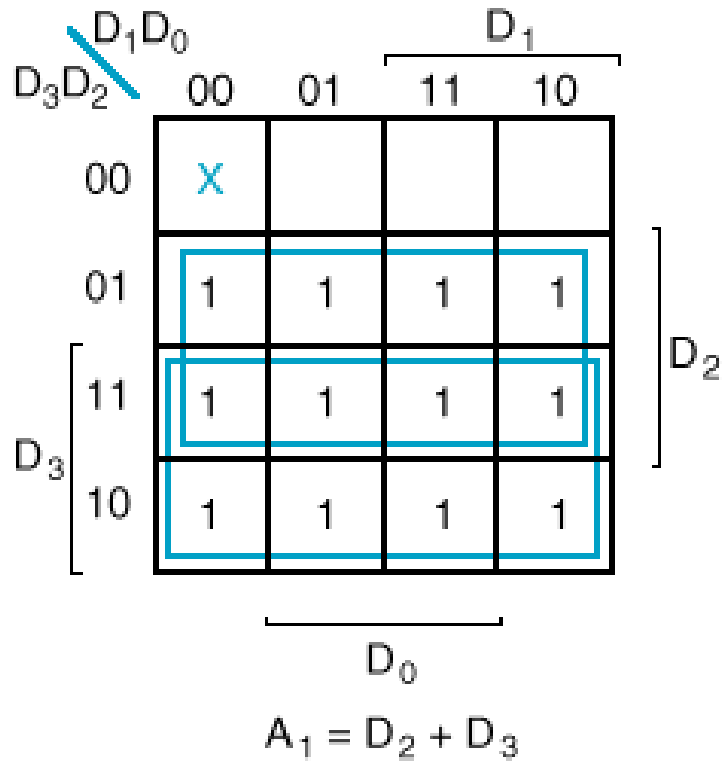
Example: 4-to-2 Priority Encoder Truth Table

Inputs				Outputs		
D_3	D_2	D_1	D_0	A_1	A_0	V
0	0	0	0	X	X	0
0	0	0	1	0	0	1
0	0	1	X	0	1	1
0	1	X	X	1	0	1
1	X	X	X	1	1	1

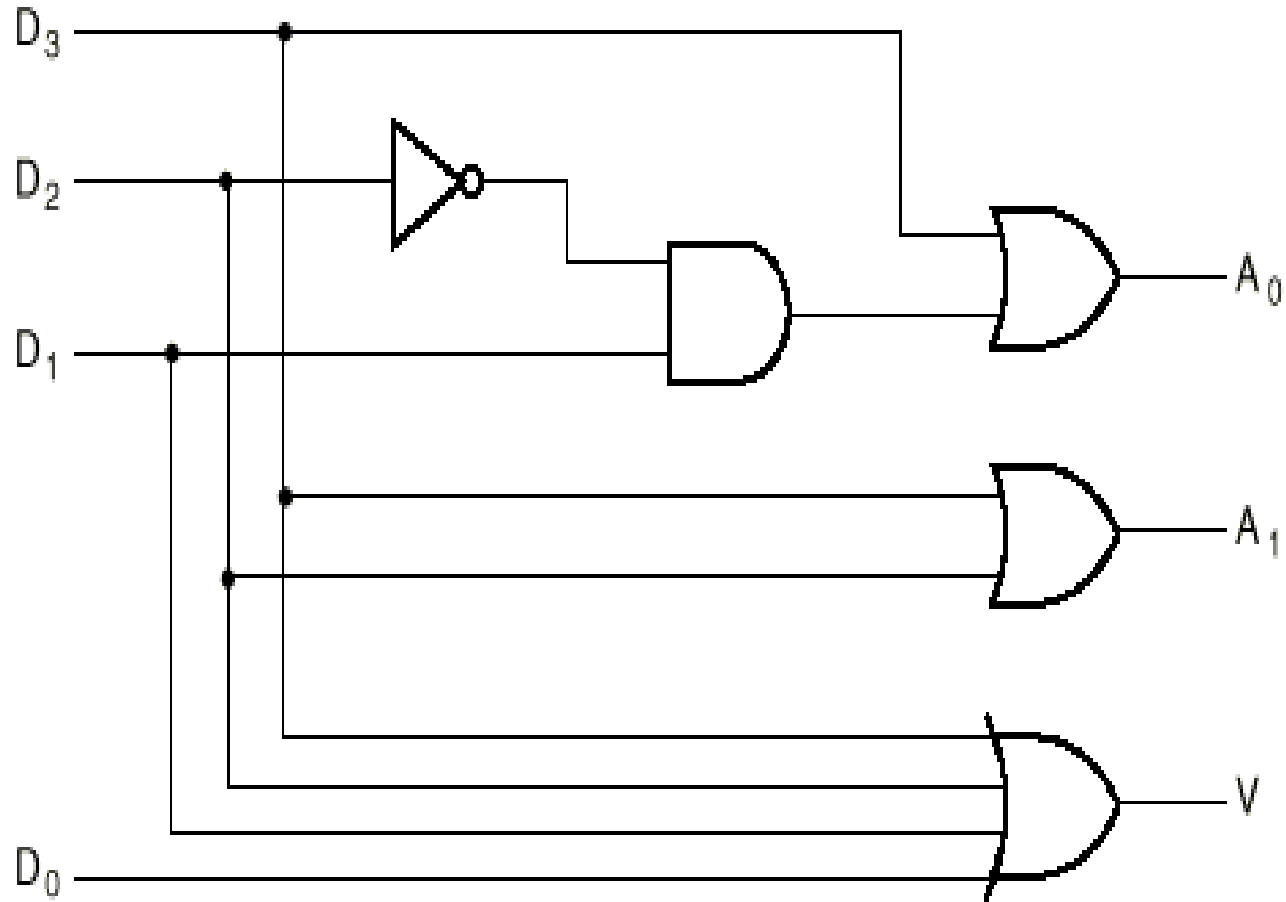
4-to-2 Priority Encoder (cont.)

- The operation of the priority encoder is such that:
 - If two or more inputs are equal to 1 at the same time, the input **in the highest-numbered position will take precedence.**
 - A *valid output indicator*, designated by V , is set to 1 only **when one or more inputs** are equal to 1. **$V = D_3 + D_2 + D_1 + D_0$ by inspection.**

4-to-2 Priority Encoder (cont.)

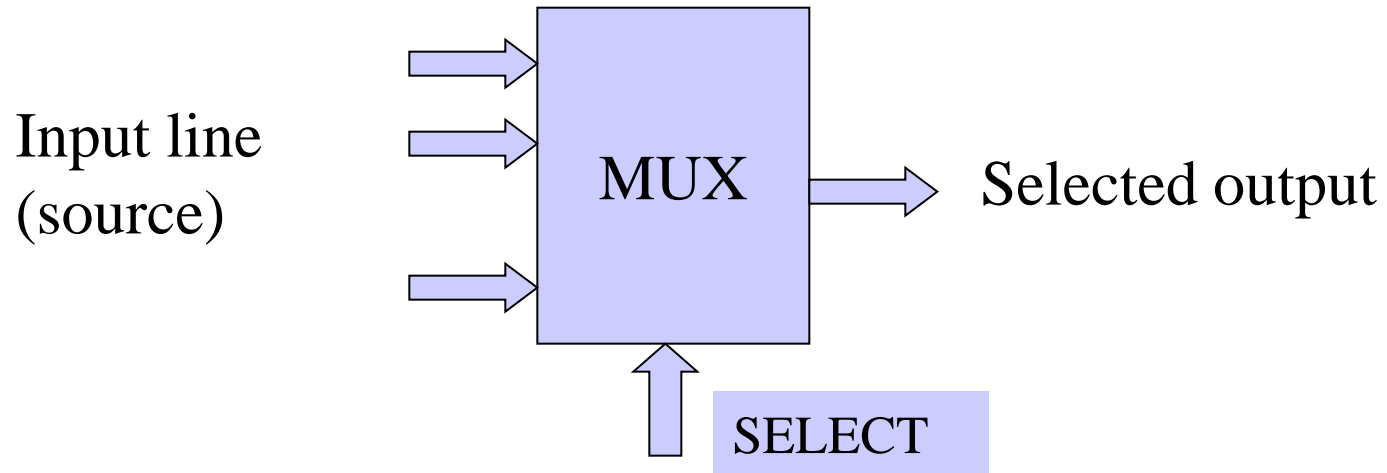


4-to-2 Priority Encoder (cont.)



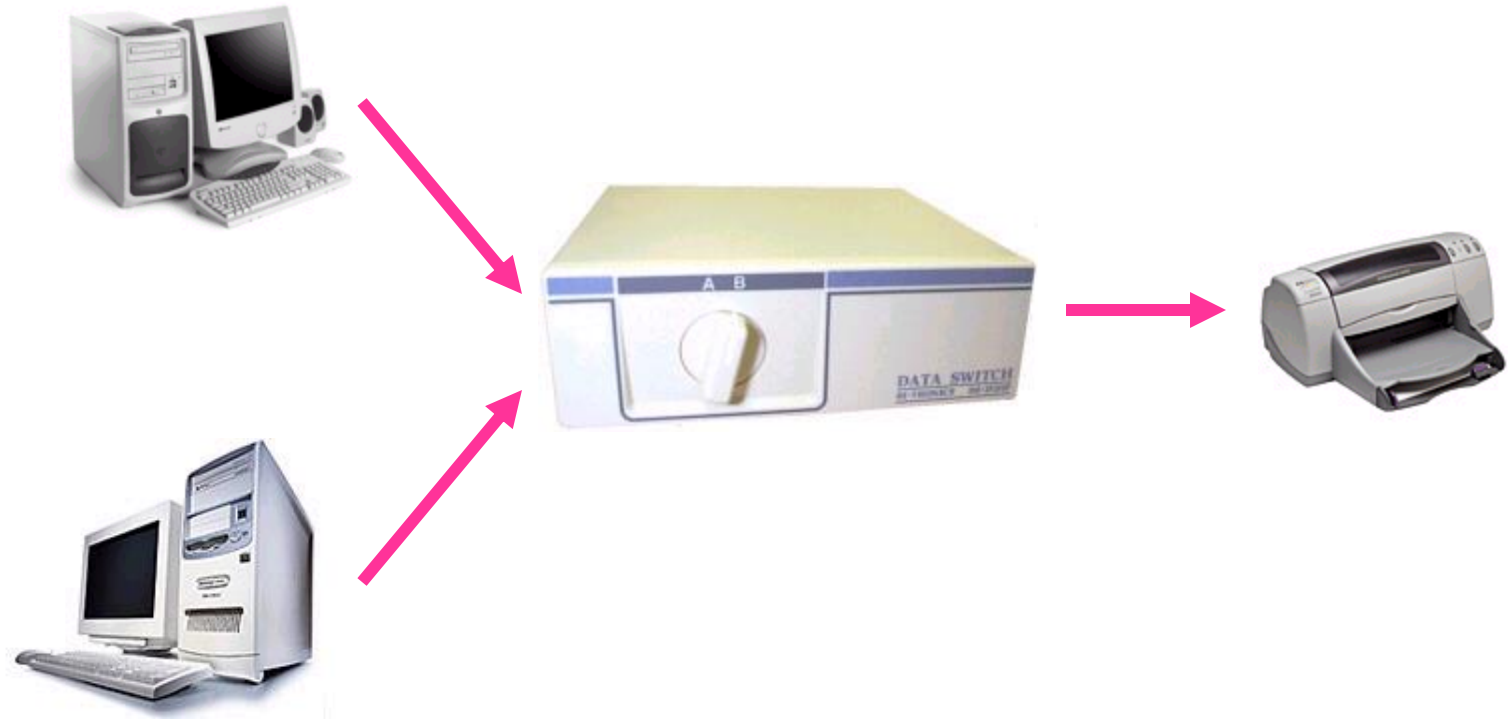
Multiplexer (MUX): Data Selectors

- A multiplexer selects one of several input signals and passes it on to the output.
- Routing of selected data input to the output is controlled by SELECT inputs.



Multiplexer

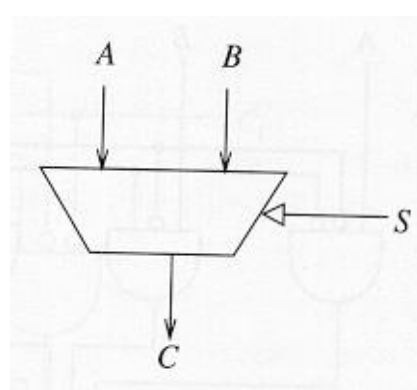
- Multiplexers, or muxes, are used to choose between resources.
- A real-life example: in the old days before networking, several computers could share one printer through the use of a switch.



Multiplexer (MUX): Data Selectors [2]

- A combinational circuit with 2^n *data inputs*, 1 data output and a number of bit *control input* that select one of the data inputs

C takes the value of A or B depending on the value of S



2-to-1 Multiplexer

S	A	B	C
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	0
1	1	1	1

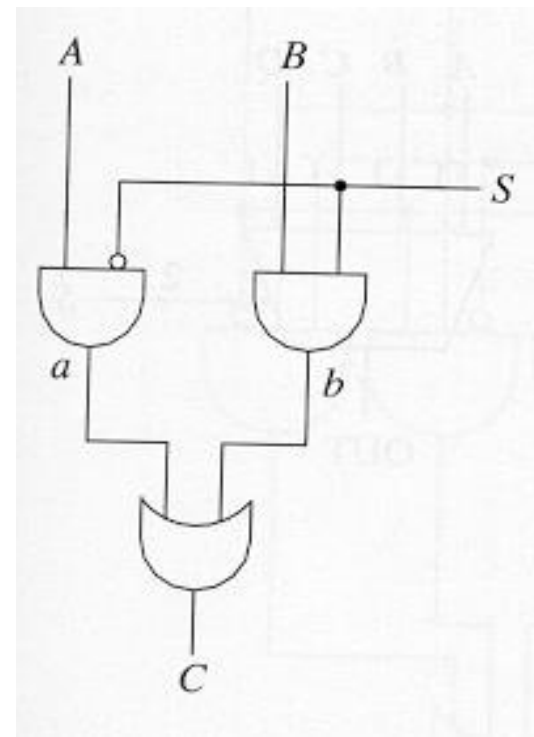
2-to-1 Multiplexer

When $S = 0 \rightarrow C = A$, when $S = 1 \rightarrow C = B$

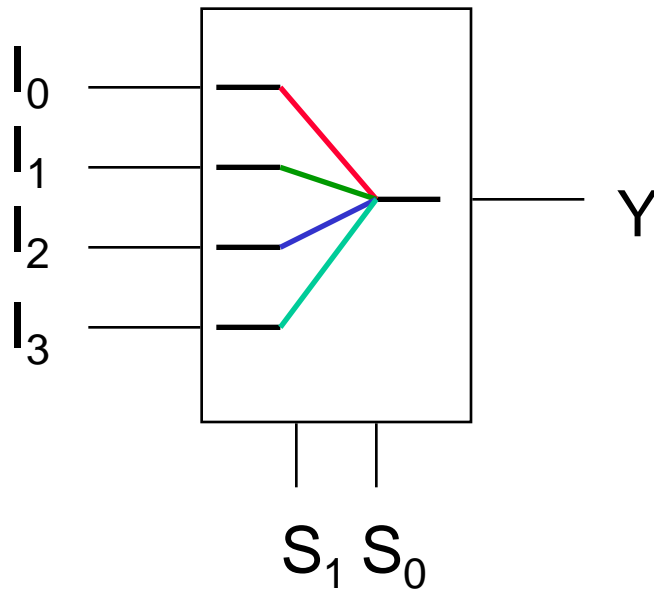
$$C = \bar{S}.A.\bar{B} + \bar{S}.A.B + S.\bar{A}.B + S.A.B$$
$$= \bar{S}.A + S.B$$

Inputs			Output
S	A	B	C
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	0
1	1	1	1

Two level implementation



4 – 1 Multiplexer (MUX)



if $(S_1 S_0)_2 = 0$ Then $Y = I_0$

$$Q = \bar{S}_0 \cdot \bar{S}_1 \cdot I_0$$

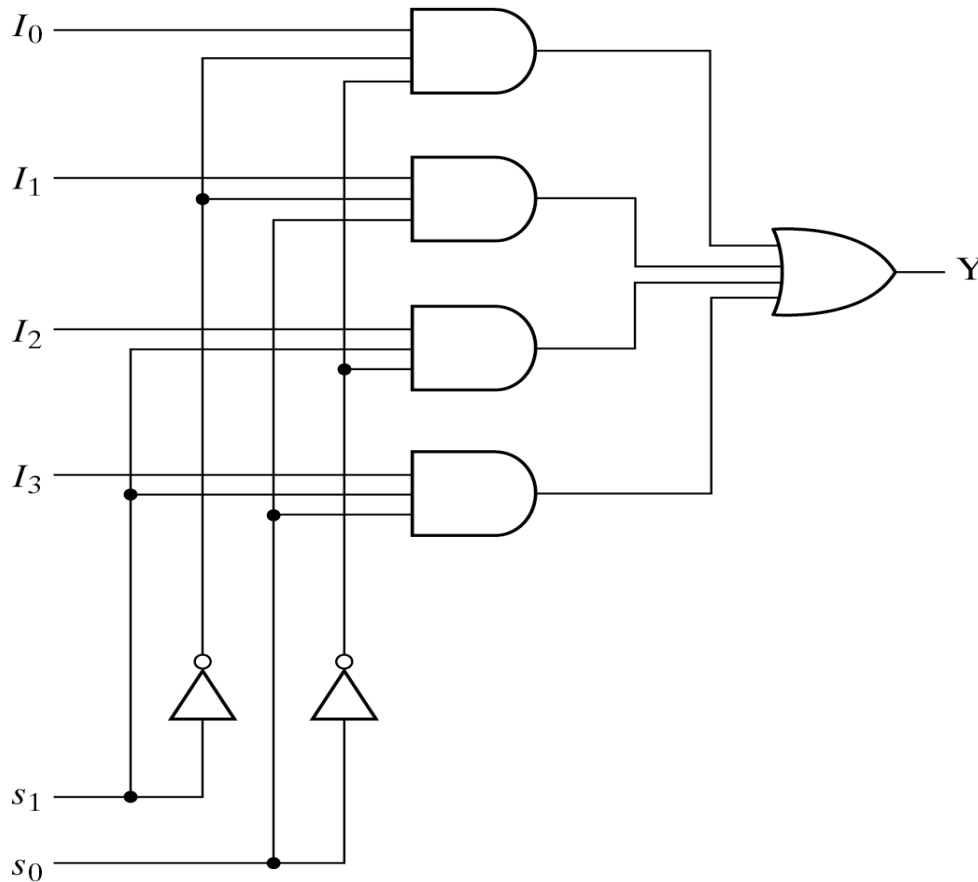
if $(S_1 S_0)_2 = 1$ Then $Y = I_1$

$$Q = S_0 \cdot \bar{S}_1 \cdot I_1$$

and so on, thus we have

$$Y = \bar{S}_1 \cdot \bar{S}_0 \cdot I_0 + \bar{S}_1 \cdot S_0 \cdot I_1 + S_1 \cdot \bar{S}_0 \cdot I_2 + S_1 \cdot S_0 \cdot I_3$$

4-1 MUX- Two level Implementation



(a) Logic diagram

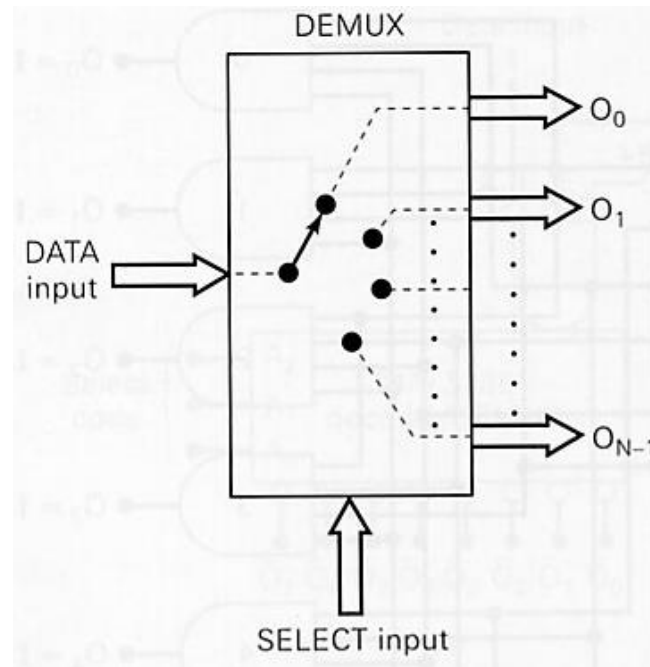
s_1	s_0	Y
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3

(b) Function table

FIG. 4.25 4-to-1 Data Multiplexer

Demultiplexer (DEMUX): Data Distributor

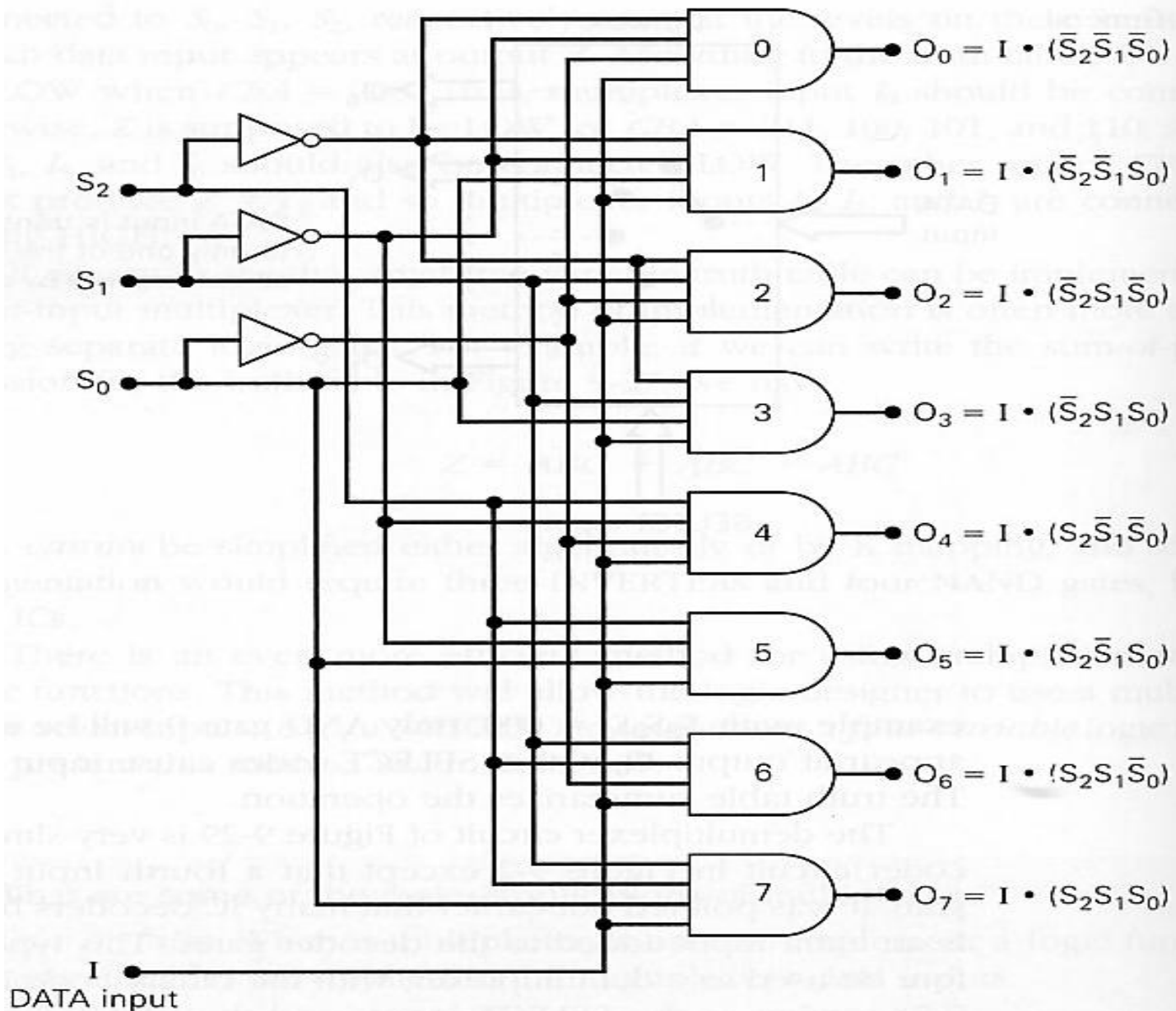
- Demultiplexer (DEMUX) takes a single input and distributes it over several outputs.



1-line-to-8-line Demultiplexer

SELECT code			OUTPUTS							
S_2	S_1	S_0	O_7	O_6	O_5	O_4	O_3	O_2	O_1	O_0
0	0	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	1	0	0	0	0	0	0	1	0	0
0	1	1	0	0	0	0	1	0	0	0
1	0	0	0	0	0	1	0	0	0	0
1	0	1	0	0	1	0	0	0	0	0
1	1	0	0	1	0	0	0	0	0	0
1	1	1	1	0	0	0	0	0	0	0

1-line-to-8-line Demultiplexer

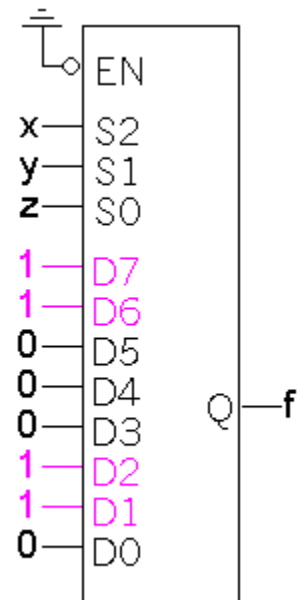


Implementing functions with multiplexers

- Multiplexers can be used to implement arbitrary functions.
- One way to implement a function of n variables is to use an n -to-1 multiplexer
 - For each minterm m_i of the function, connect 1 data input D_i . Each data input corresponds to one row of the truth table.
 - Connect the function's input variables to select inputs. These are used to indicate a particular input combination.

Example, $f(x,y,z) = \sum m(1,2,6,7)$ can be implemented as follows

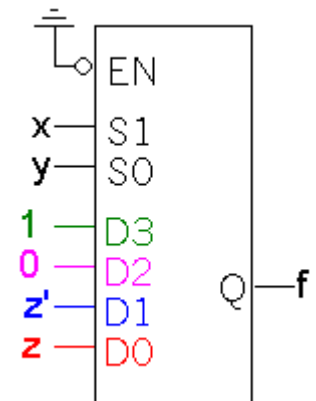
x	y	z	f
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1



Simplified Implementation (more efficient)

- We can actually implement $f(x,y,z) = \Sigma m(1,2,6,7)$ with just a 4-to-1 mux, instead of an 8-to-1.
- Step 1: Find the truth table for the function, and group the rows into pairs. Within each pair of rows, x and y are the same, so f is a function of z only.
 - When $xy=00$, $f=z$
 - When $xy=01$, $f=z'$
 - When $xy=10$, $f=0$
 - When $xy=11$, $f=1$
- Step 2: Connect the first two input variables of the truth table (here, x and y) to the select bits S1 S0 of the 4-to-1 mux.
- Step 3: Connect the equations above for f(z) to the data inputs D0-D3.

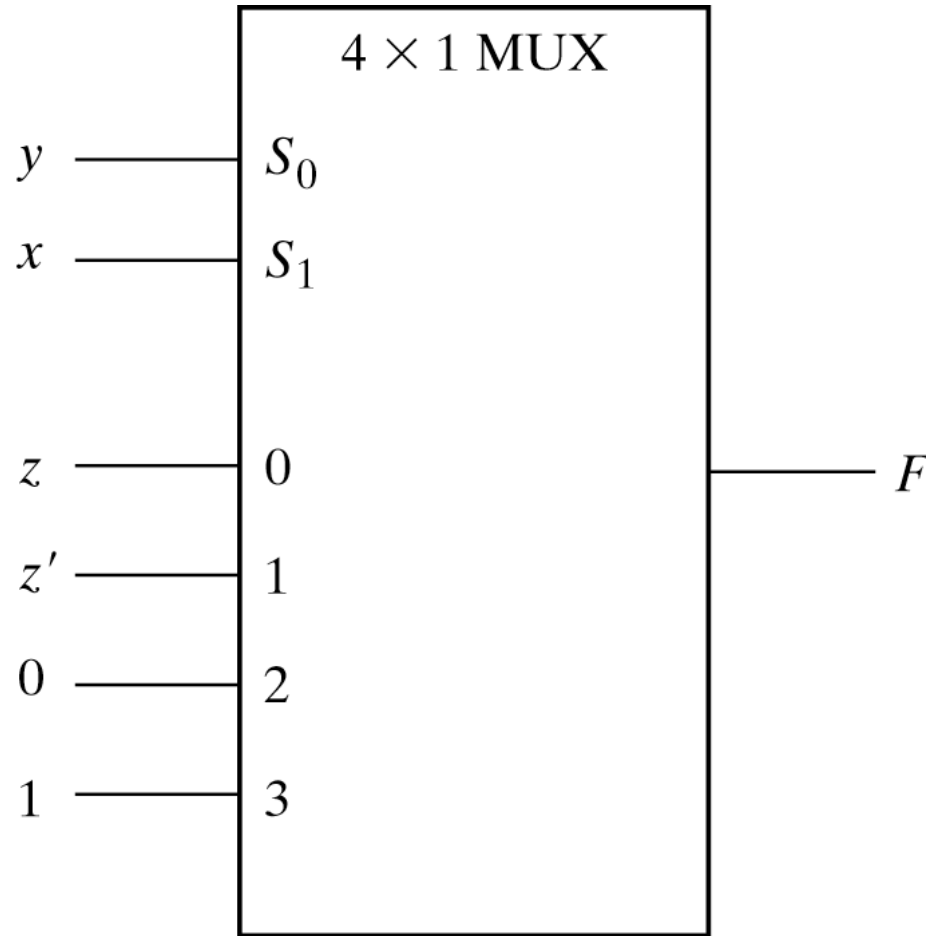
x	y	z	f
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1



Use of multiplexer to implement Boolean functions

x	y	z	F	
0	0	0	0	
0	0	1	1	$F = z$
0	1	0	1	
0	1	1	0	$F = z'$
1	0	0	0	
1	0	1	0	$F = 0$
1	1	0	1	
1	1	1	1	$F = 1$

(a) Truth table



(b) Multiplexer implementation

Fig. 4-27 Implementing a Boolean Function with a Multiplexer

Use of multiplexer to implement Boolean functions

- Implement the following Boolean function with a multiplexer

- $f(A,B,C,D) = \Sigma m (1,3,4,11, 12,13, 14,15)$

Use of multiplexer to implement Boolean functions

- $f(A,B,C,D) = \sum m (1,3,4,11, 12,13, 14,15)$ $n-1$ selection and 2 power $n-1$

<i>A</i>	<i>B</i>	<i>C</i>	<i>D</i>	<i>F</i>	
0	0	0	0	0	
0	0	0	1	1	$F = D$
0	0	1	0	0	
0	0	1	1	1	$F = D$
0	1	0	0	1	
0	1	0	1	0	$F = D'$
0	1	1	0	0	
0	1	1	1	0	$F = 0$
1	0	0	0	0	
1	0	0	1	0	$F = 0$
1	0	1	0	0	
1	0	1	1	1	$F = D$
1	1	0	0	1	
1	1	0	1	1	$F = 1$
1	1	1	0	1	
1	1	1	1	1	$F = 1$

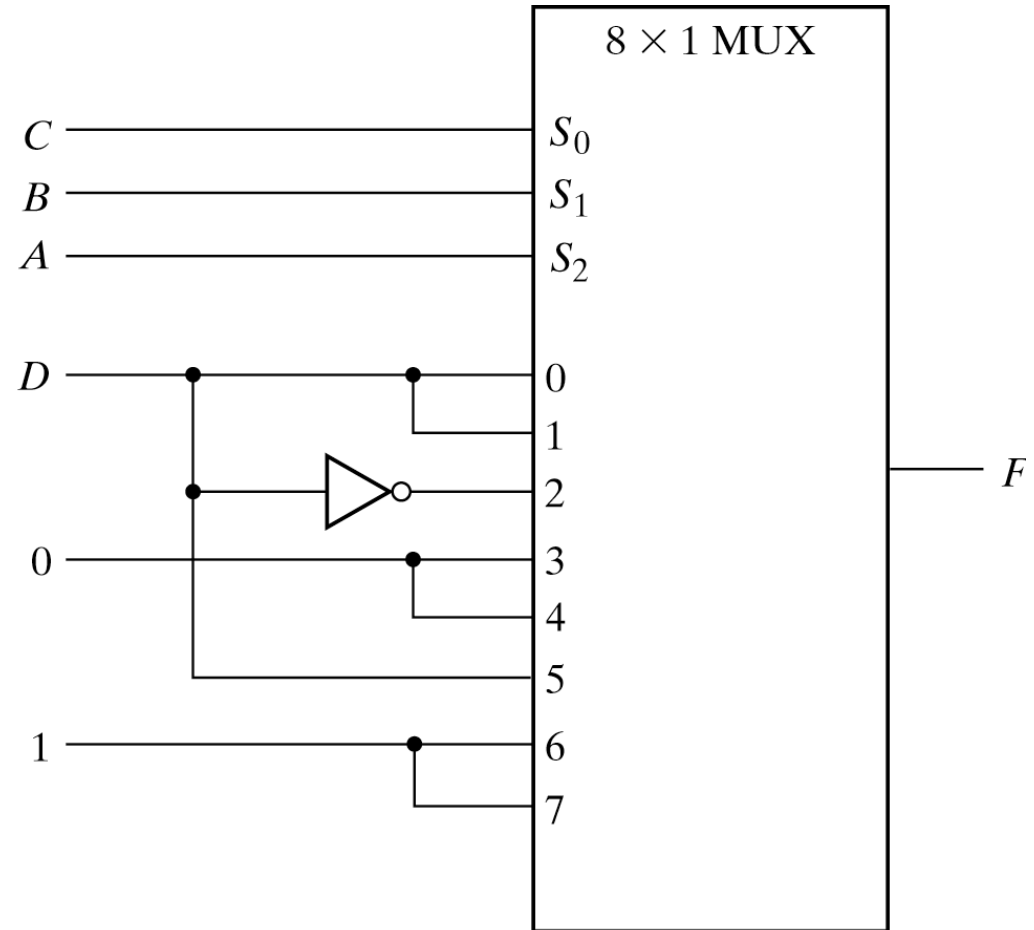


Fig. 4-28 Implementing a 4-Input Function with a Multiplexer

Other Combinational circuit examples

The following examples will be done in the DGD

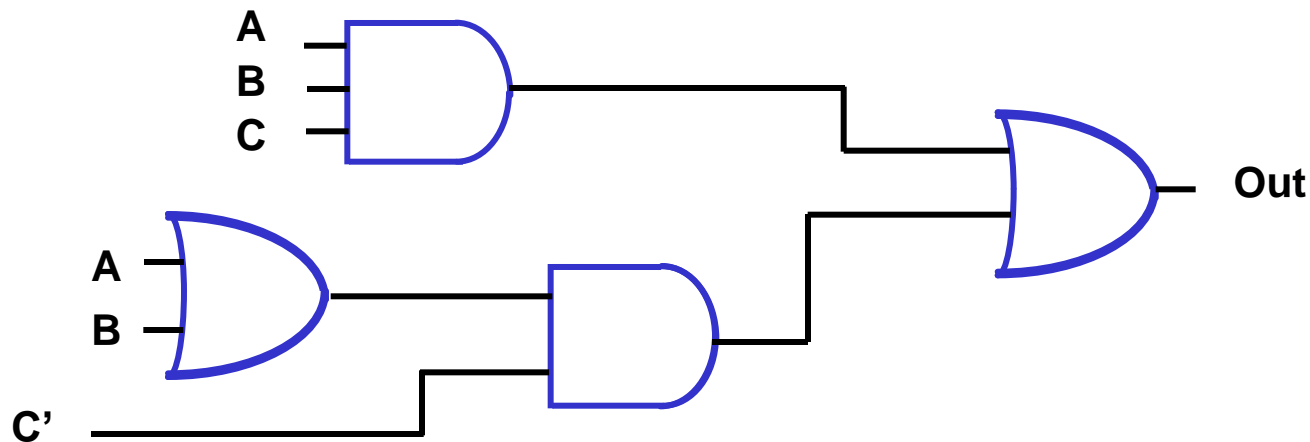
- 4 bit Magnitude Comparator
- 4 bit by 3 bit Binary Multiplier
- 7 segment decoder (started in the class) will be in the assignment #4

Analyzing digital circuits

- Important concept – analyzing digital circuits
 - Given a circuit
 - Create a truth table
 - Create a minimized circuit
- Approaches
 - Boolean expression approach
 - Truth table approach

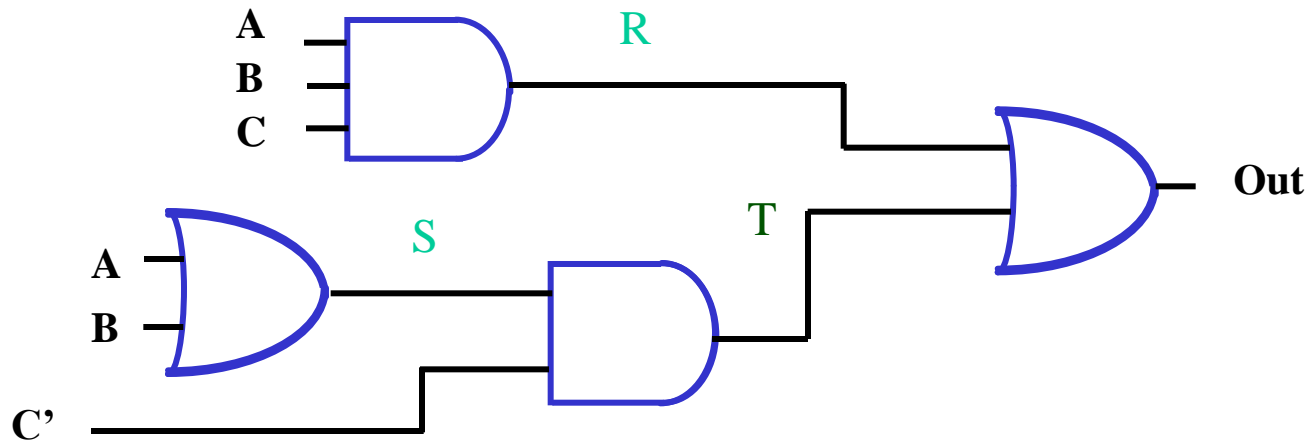
From a logic circuit to equation/ Truth table

- How can we convert from a circuit drawing to an equation or truth table?
- Two approaches
 - Create intermediate equations
 - Create intermediate truth tables



Label Gate Outputs

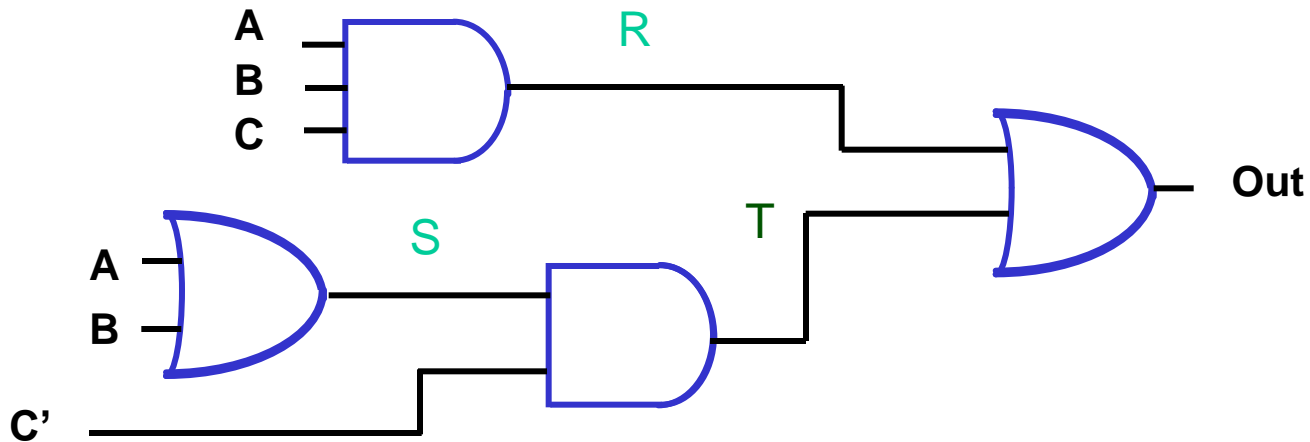
1. Label all gate outputs that are a function of input variables.
2. Label gates that are a function of input variables and previously labeled gates.
3. Repeat process until all outputs are labeled.



Approach 1: Create Intermediate Equations

➤ **Step 1: Create an equation for each gate output based on its input.**

- **$R = ABC$**
- **$S = A + B$**
- **$T = C'S$**
- **$Out = R + T$**

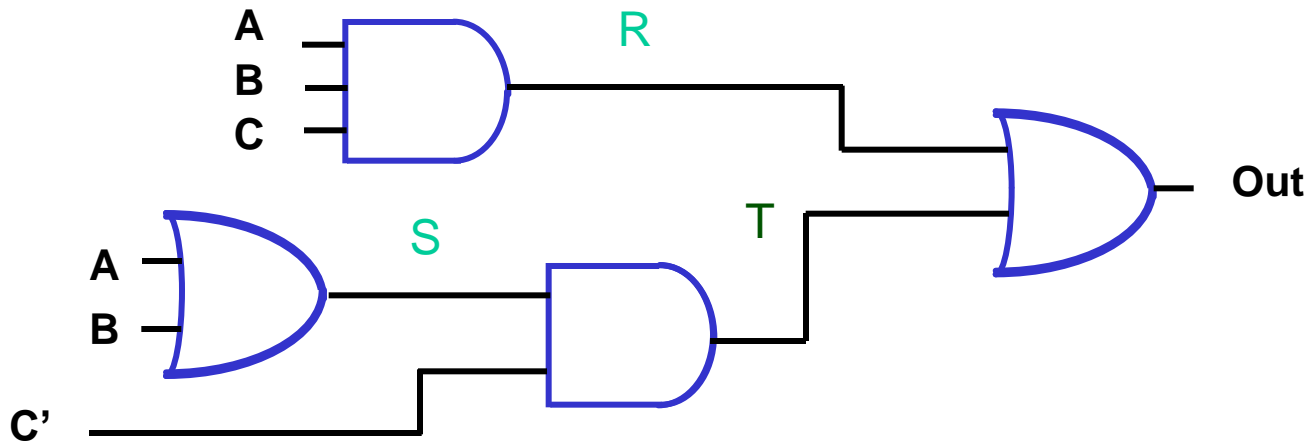


Approach 1: Substitute in sub expressions

➤ Step 2: Form a relationship based on input variables

(A, B, C)

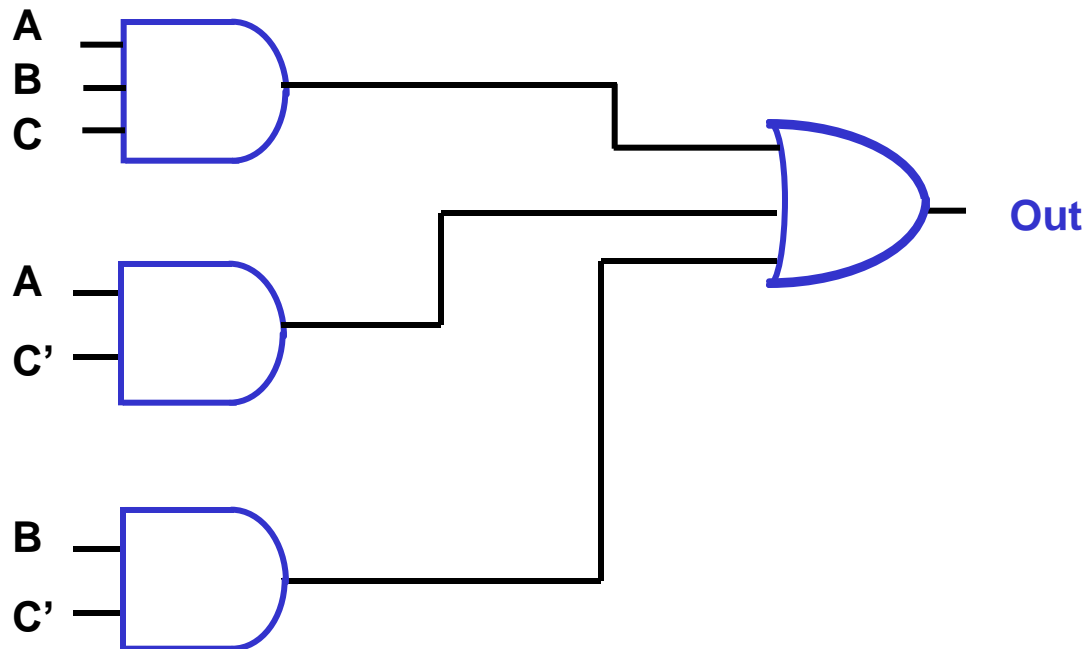
- $R = ABC$
- $S = A + B$
- $T = C'S = C'(A + B)$
- $Out = RT = ABC + C'(A+B)$



Approach 1: Substitute in sub expressions

➤ Step 3: Expand equation to SOP final result

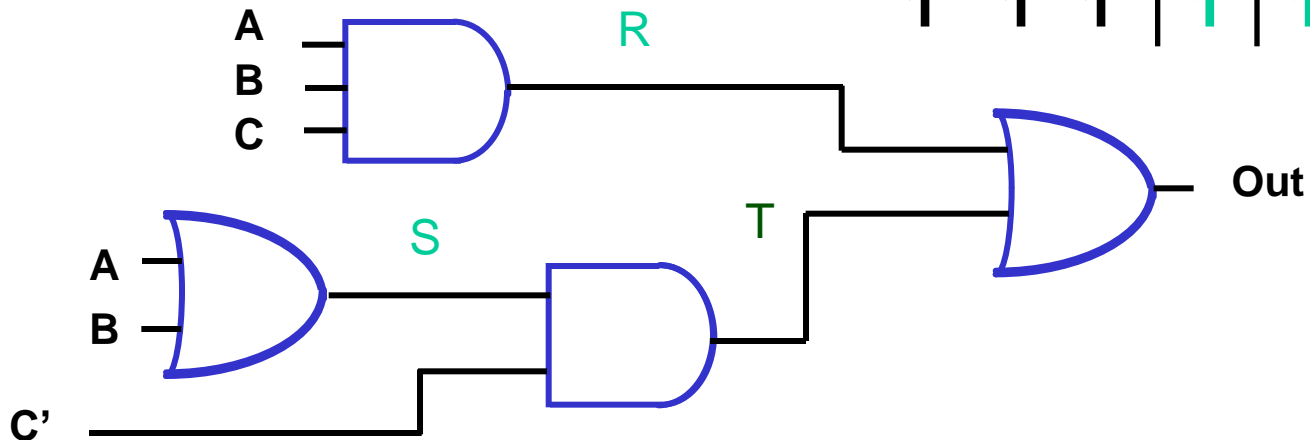
- **Out = ABC + C'(A+B) = ABC + AC' + BC'**



Approach 2: Truth Table

- **Step 1: Determine outputs for functions of input variables.**

A	B	C	R	S
0	0	0	0	0
0	0	1	0	0
0	1	0	0	1
0	1	1	0	1
1	0	0	0	1
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

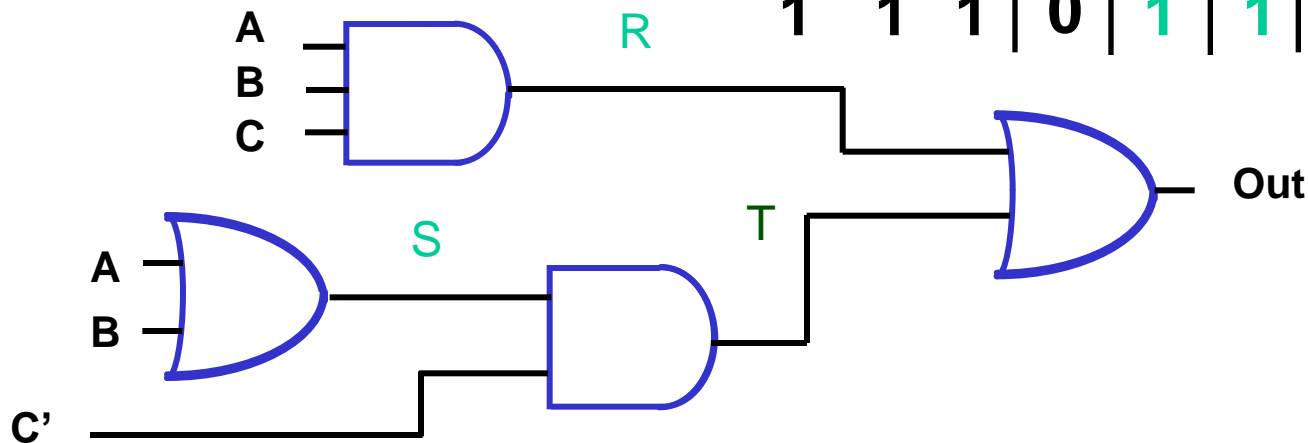


Approach 2: Truth Table

- **Step 2: Determine outputs for functions of intermediate variables.**

$$T = S * C'$$

A	B	C	C'	R	S	T
0	0	0	1	0	0	0
0	0	1	0	0	0	0
0	1	0	1	0	1	1
0	1	1	0	0	1	0
1	0	0	1	0	1	1
1	0	1	0	0	1	0
1	1	0	1	0	1	1
1	1	1	0	1	1	0

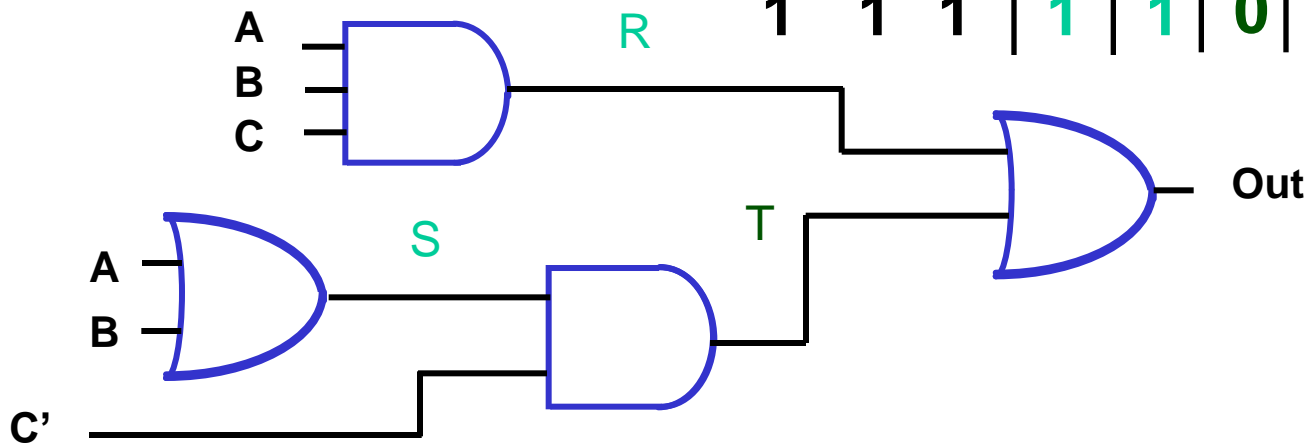


Approach 2: Truth Table

- Step 3: Determine outputs for function.

$$R + T = \text{Out}$$

A	B	C	R	S	T	Out
0	0	0	0	0	0	0
0	0	1	0	0	0	0
0	1	0	0	1	1	1
0	1	1	0	1	0	0
1	0	0	0	1	1	1
1	0	1	0	1	0	0
1	1	0	0	1	1	1
1	1	1	1	1	0	1



More Difficult Example

➤ Step 3: Note labels on interior nodes

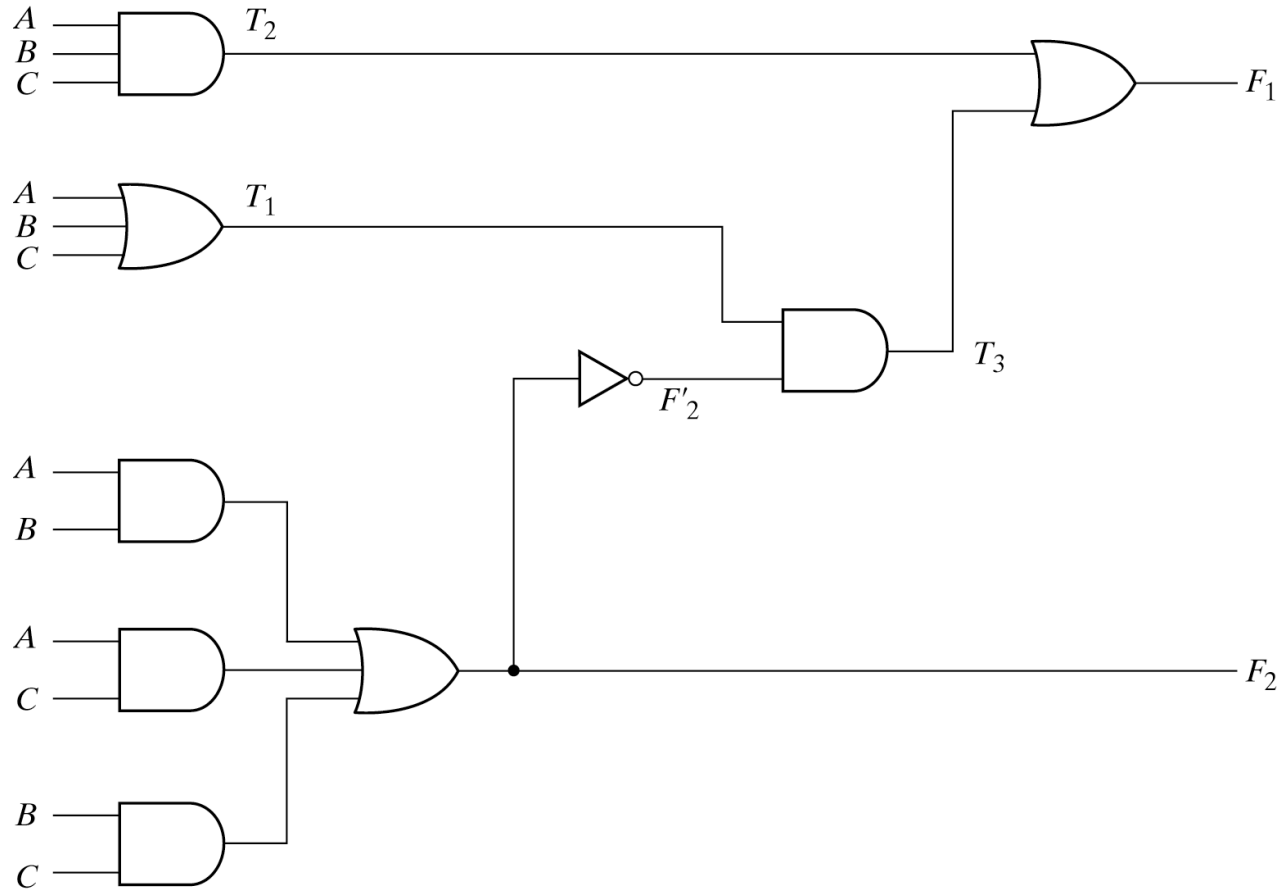


Fig. 4-2 Logic Diagram for Analysis Example

More Difficult Example: Truth Table

- Remember to determine intermediate variables starting from the inputs.
- When all inputs are determined for a gate, determine output.
- The truth table can be reduced using K-maps.

A	B	C	F_2	F'_2	T_1	T_2	T_3	F_1
0	0	0	0	1	0	0	0	0
0	0	1	0	1	1	0	1	1
0	1	0	0	1	1	0	1	1
0	1	1	1	0	1	0	0	0
1	0	0	0	1	1	0	1	1
1	0	1	1	0	1	0	0	0
1	1	0	1	0	1	0	0	0
1	1	1	1	0	1	1	0	1