Chapter 3 Ctd: Combinational Functions and Circuits

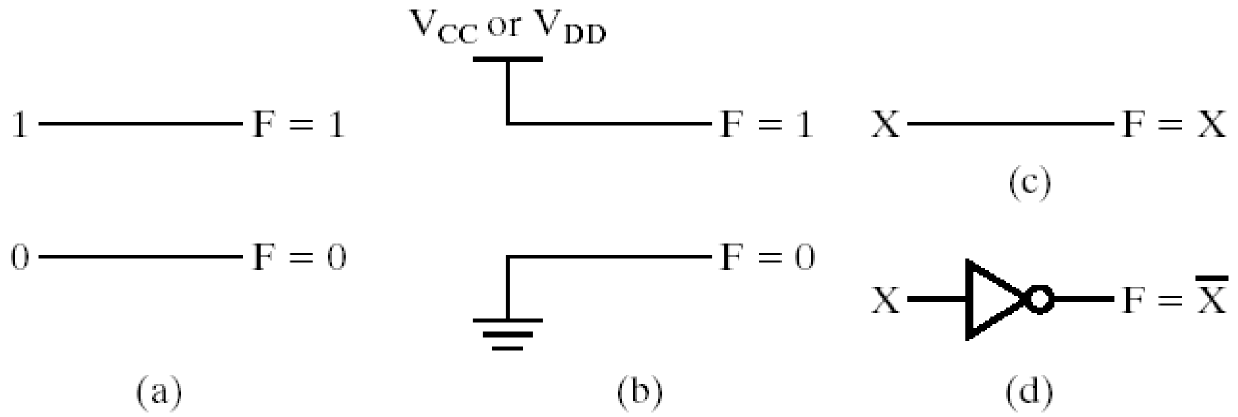
Value Fixing, Transferring, and Inverting

- Four different functions are possible as a function of single Boolean variable

TABLE 4-1
Functions of One Variable

X	F = 0	F = X	F = $\bar{X}$	F = 1
0	0	0	1	1
1	0	1	0	1

Transferring Inverting Value Fixing

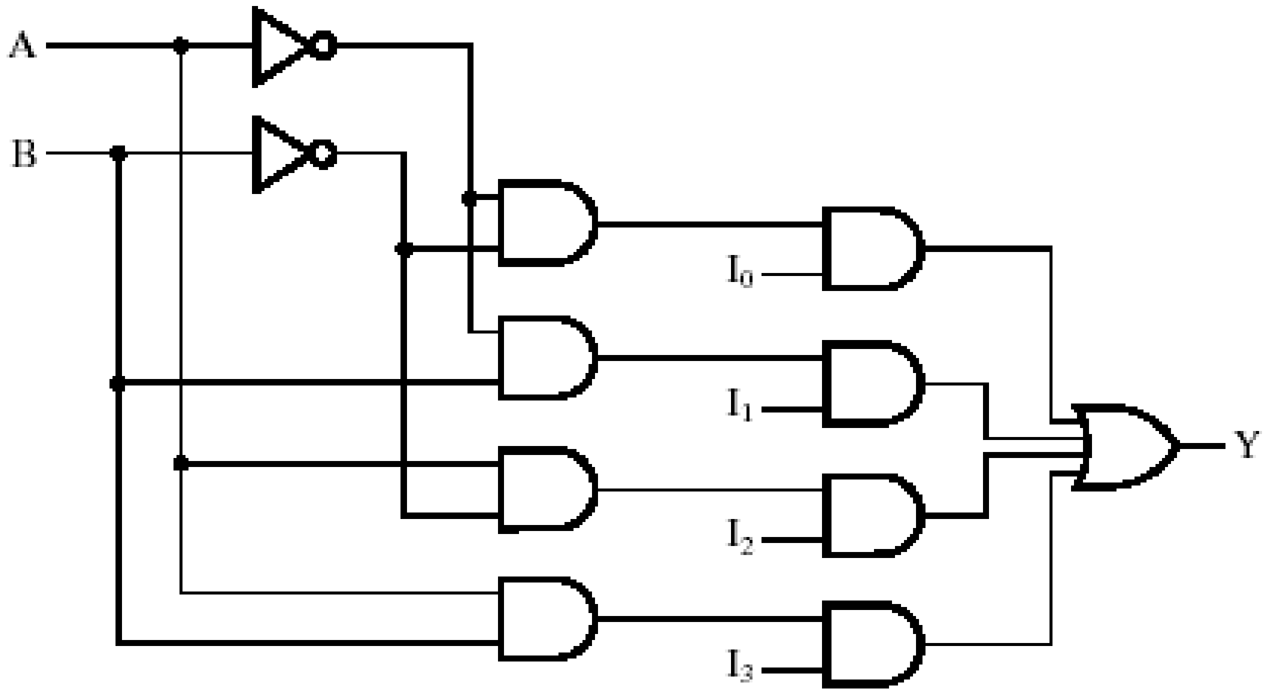


Value Fixing for Implementing a Function

- Y is actually a function of six varibles – truth table with 64 rows and 7 clmns
- Putting I_0 - I_3 in the output column reduces the truth table

A	B	Y
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3

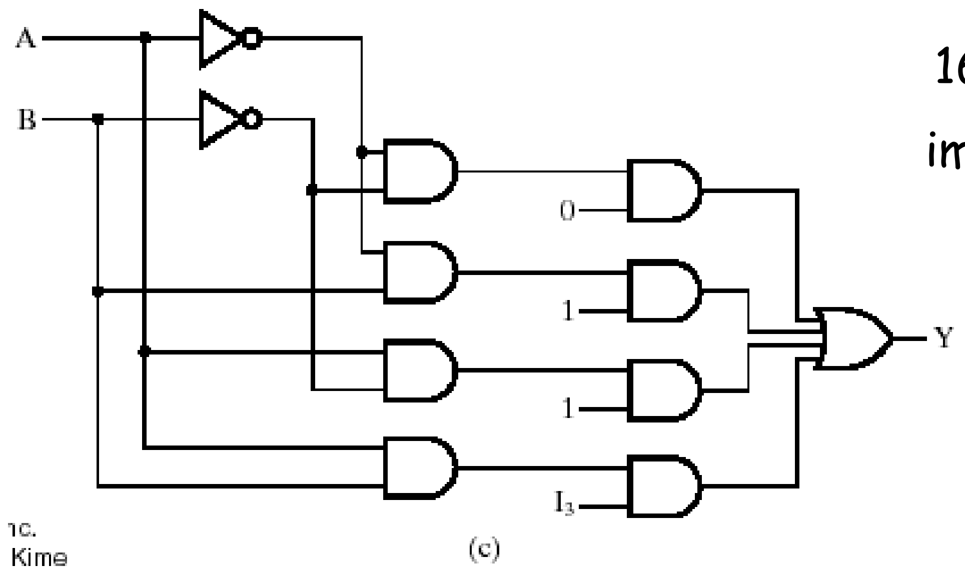
(a)



(b)

$$Y(A,B,I_0,I_1,I_2,I_3) = A'B'I_0 + A'BI_1 + AB'I_2 + ABI_3$$

Value Fixing for Implementing a Function

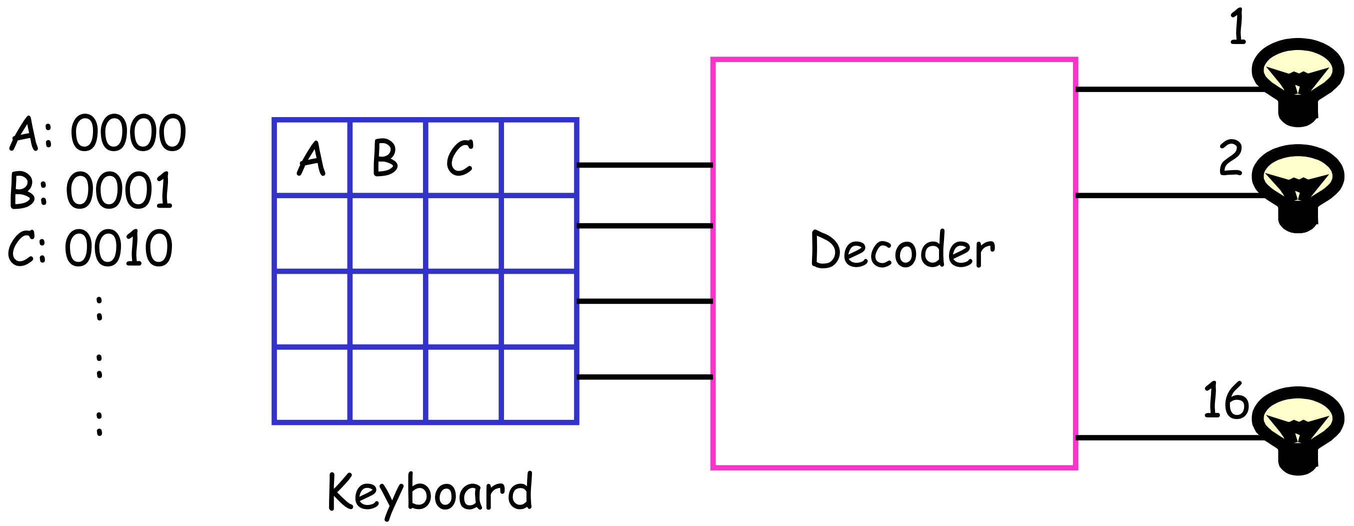


16 different functions can be implemented by setting I_0 - I_3 appropriately

TABLE 4-2
Function Implementation by Value-Fixing

A	B	$Y = A + B$	$Y = A\bar{B} + \bar{A}B$	$Y = A + B$ ($I_3 = 1$) or $Y = A\bar{B} + \bar{A}B$ ($I_3 = 0$)
0	0	0	0	0
0	1	1	1	1
1	0	1	1	1
1	1	1	0	I_3

What is a decoder?

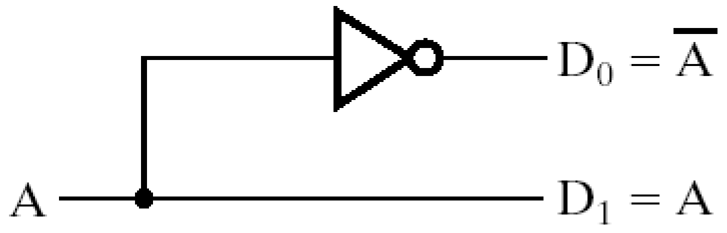


- Decoder: A combinational circuit with an n-bit binary code applied to its inputs and m-bit binary code appearing at the outputs
  - n ≤ m ≤ 2^n
  - Detect which of the 2^n combinations is represented at the inputs
  - Produce m outputs, only one of which is “1”

1-to-2 (Line) Decoder

A	D_0	D_1
0	1	0
1	0	1

(a)



(b)

2-to-4 Decoder

- A 2-to-4 decoder operates according to the following truth table
  - The 2-bit input is called S1S0, and the four outputs are Q0-Q3
  - If the input is the binary number i, then output Qi is uniquely true

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

- For instance, if the input S1 S0 = 10 (decimal 2), then output Q2 is true, and Q0, Q1, Q3 are all false

- This circuit "decodes" a binary number into a "one-of-four" code

How can you build a 2-to-4 decoder?

- Follow the design procedures from last time! We have a truth table, so we can write equations for each of the four outputs (Q0-Q3)

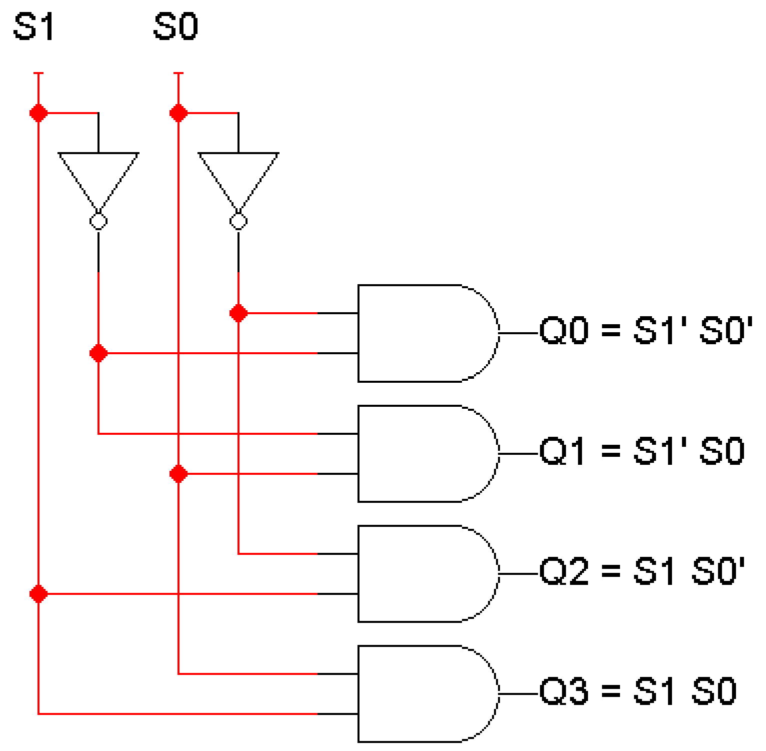
S1	S0	Q0	Q1	Q2	Q3
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1

- In this case there's not much to be simplified. Here are the equations:

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

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 Inputs

- 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

An aside: 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

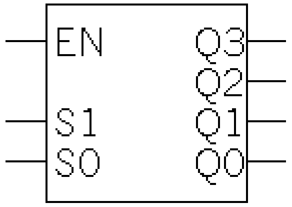
- We can abbreviate the table by writing x's in the input columns for S1 and S0

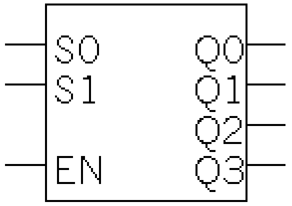
EN	S1	S0	Q0	Q1	Q2	Q3
0	x	x	0	0	0	0
1	0	0	1	0	0	0
1	0	1	0	1	0	0
1	1	0	0	0	1	0
1	1	1	0	0	0	1

Blocks and Abstraction

- Decoders are common enough that we want to encapsulate them and treat them as an individual entity

- Block diagrams for 2-to-4 decoders are shown here. The *names* of the inputs and outputs, not their order, is what matters.





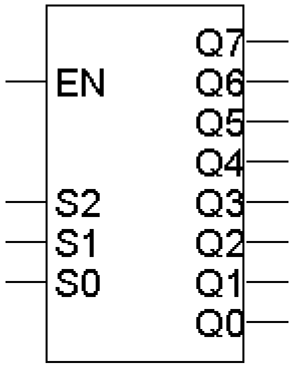
Q0 = S1′ S0′
Q1 = S1′ S0
Q2 = S1 S0′
Q3 = S1 S0

- A decoder block provides abstraction:
  - You can use the decoder as long as you know its truth table or equations, without knowing exactly what's inside
  - It makes diagrams simpler by hiding the internal circuitry
  - It simplifies hardware reuse. You don't have to keep rebuilding the decoder from scratch every time you need it

- These blocks are like functions in programming!

3-to-8 decoder

- Larger decoders are similar. Here is a 3-to-8 decoder
  - The block symbol is on the right
  - A truth table (without EN) is below
  - Output equations are at the bottom right
- Again, only one output is true for any input combination



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

$Q0 = S2' \, S1' \, S0'$
$Q1 = S2' \, S1' \, S0$
$Q2 = S2' \, S1 \, S0'$
$Q3 = S2' \, S1 \, S0$
$Q4 = S2 \, S1' \, S0'$
$Q5 = S2 \, S1' \, S0$
$Q6 = S2 \, S1 \, S0'$
$Q7 = S2 \, S1 \, S0$

So what good is a decoder?

- Do the truth table and equations look familiar?

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
- Arbitrary functions can be implemented with decoders. If a sum of minterms equation for a function is given, a decoder (a minterm generator) is used to implement that function

Design example: Addition

- Let's make a circuit that adds three 1-bit inputs X, Y and Z
- We will need two bits to represent the total; let's call them C and S, for "carry" and "sum." Note that C and S are two separate functions of the same inputs X, Y and Z
- Here are a truth table and sum-of-minterms equations for C and S

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

0 + 1 + 1 = 10 →

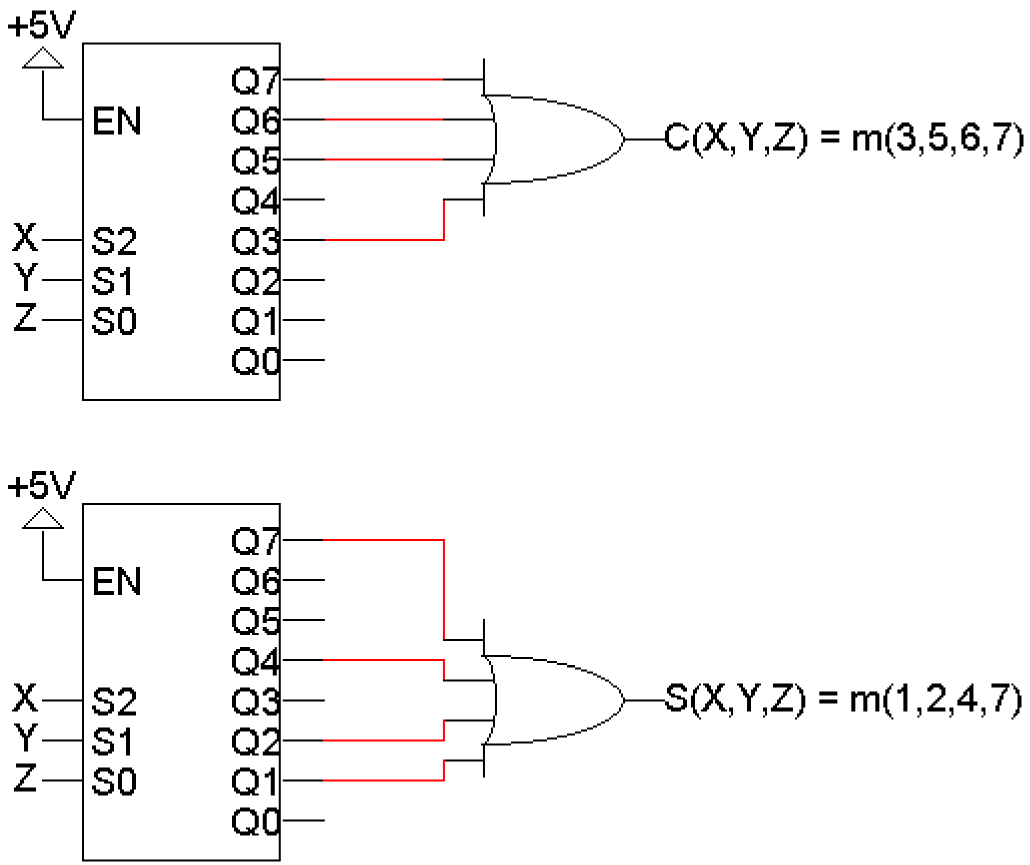
← 1 + 1 + 1 = 11

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

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

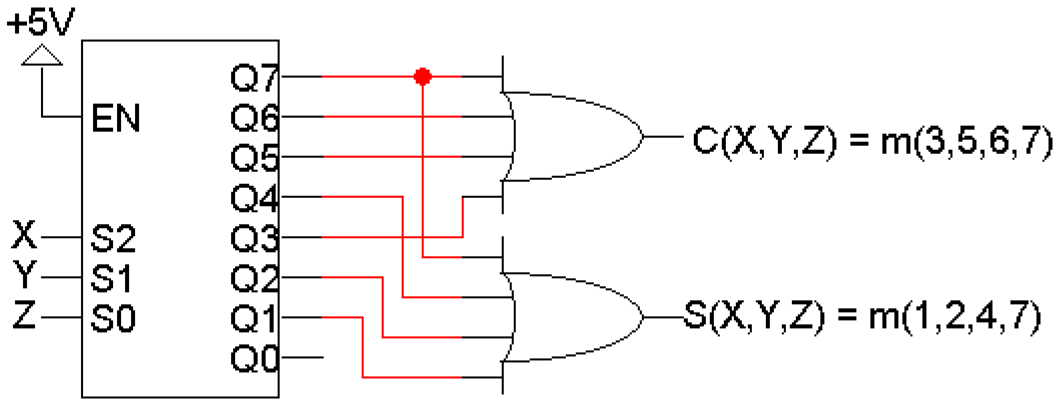
Decoder-based Adder

- Here, two 3-to-8 decoders implement *C* and *S* as sums of minterms.



Using just one decoder

- Since the two functions C and S both have the same inputs, we could use just one decoder instead of two.



Building a 3-to-8 decoder

- You could build a 3-to-8 decoder directly from the truth table and equations below, just like how we built the 2-to-4 decoder
- Another way to design a decoder is to break it into smaller pieces
- Notice some patterns in the table below:
  - When S2 = 0, outputs Q0-Q3 are generated as in a 2-to-4 decoder
  - When S2 = 1, outputs Q4-Q7 are generated as in a 2-to-4 decoder

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

$Q0 = S2'\,S1'\,S0' = m_0$

$Q1 = S2'\,S1'\,S0 = m_1$

$Q2 = S2'\,S1\,S0' = m_2$

$Q3 = S2'\,S1\,S0 = m_3$

$Q4 = S2\,S1'\,S0' = m_4$

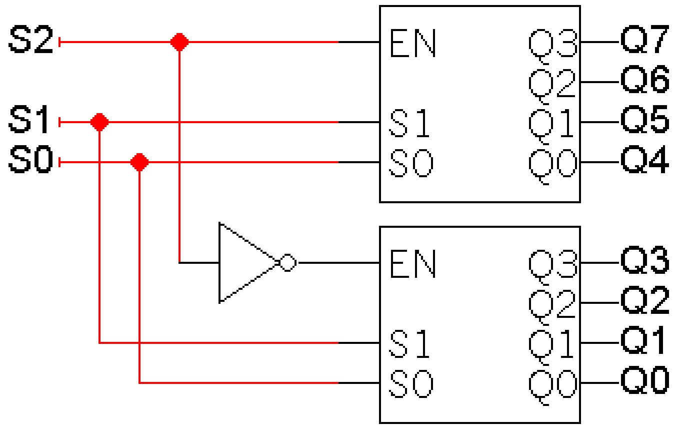
$Q5 = S2\,S1'\,S0 = m_5$

$Q6 = S2\,S1\,S0' = m_6$

$Q7 = S2\,S1\,S0 = m_7$

Decoder Expansion

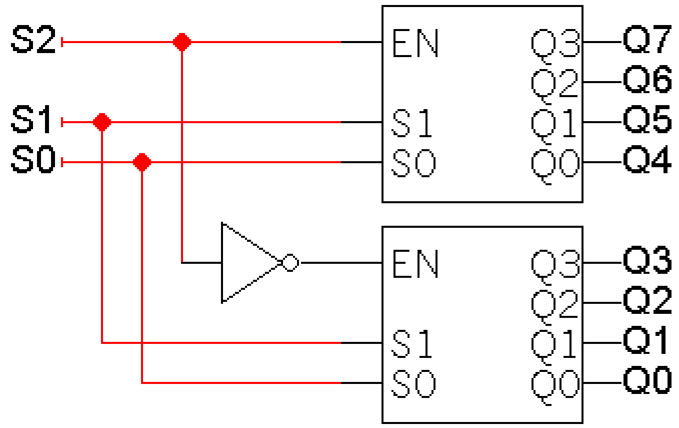
- You can use enable inputs to string decoders together. Here's a 3-to-8 decoder constructed from two 2-to-4 decoders:



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

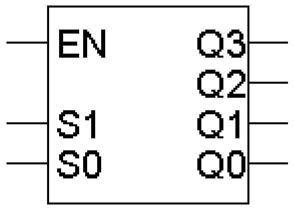
Modularity

- Be careful not to confuse the "inner" inputs and outputs of the 2-to-4 decoders with the "outer" inputs and outputs of the 3-to-8 decoder (which are in boldface)
- This is similar to having several functions in a program which all use a formal parameter "x"



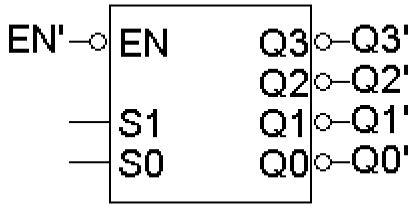
A variation of the standard decoder

- The decoders we've seen so far are active-high decoders



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

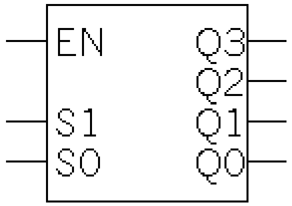
- An active-low decoder is the same thing, but with an inverted EN input and inverted outputs



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

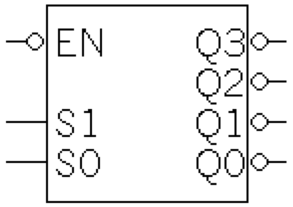
Separated at birth?

- Active-high decoders generate *minterms*, as we've already seen



Q3 = S1 S0
Q2 = S1 S0'
Q1 = S1' S0
Q0 = S1' S0'

- The output equations for an active-low decoder are mysteriously similar, yet somehow different

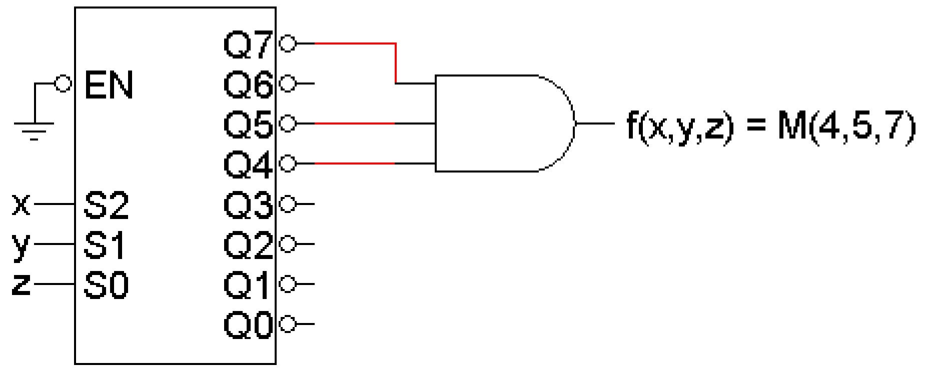


Q3' = (S1 S0)' = S1' + S0'
Q2' = (S1 S0')' = S1' + S0
Q1' = (S1' S0)' = S1 + S0'
Q0' = (S1' S0')' = S1 + S0

- It turns out that active-low decoders generate *maxterms*

Active-low Decoder

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



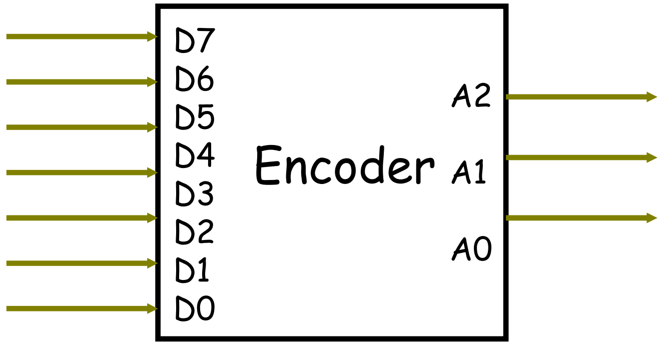
- The “ground” symbol connected to EN represents logical 0, so this decoder is always enabled

Summary of Decoders

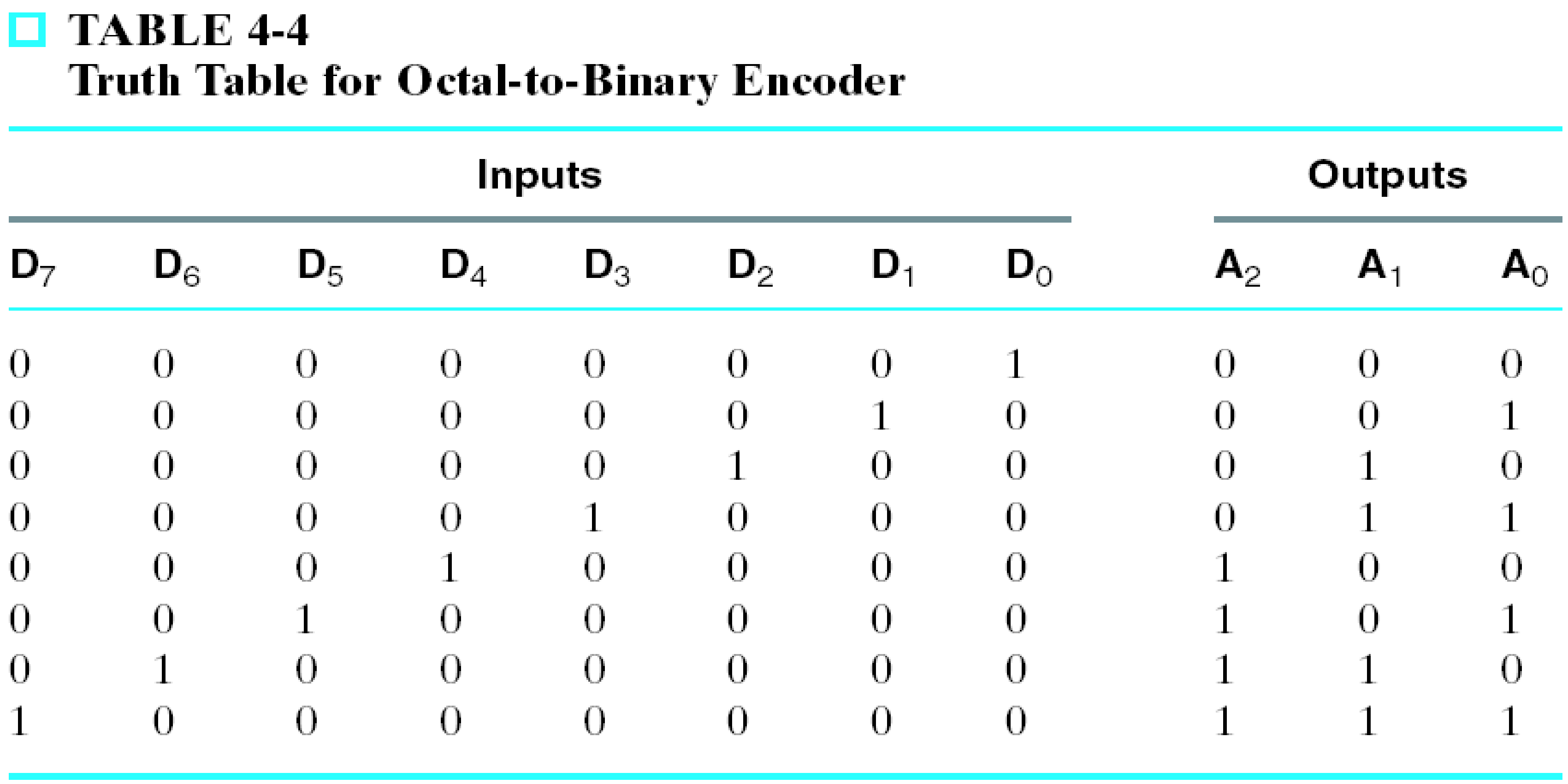
- A n-to-2^n decoder generates the minterms of an n-variable function
  - As such, decoders can be used to implement arbitrary functions
  - Later on we'll see other uses for decoders too
- Some variations of the basic decoder include:
  - Adding an enable input
  - Using active-low inputs and outputs to generate maxterms
- We also talked about:
  - Applying our circuit analysis and design techniques to understand and work with decoders
  - Using block symbols to encapsulate common circuits like decoders
  - Building larger decoders from smaller ones

Encoder

- An encoder is a digital function that performs the inverse operation of a decoder
- Octal-to-Binary Encoder: This encoder has eight inputs, one for each of the octal digits, and three outputs that generate the corresponding binary number
  - Only one input can have the value of 1 at any given time



Octal-to-Binary Encoder



- TABLE 4-4
  Truth Table for Octal-to-Binary Encoder

Inputs								Outputs		
D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	A_2	A_1	A_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

A0= D1 + D3 + D5 + D7,

A2= D4 + D5 + D6 + D7

A1= D2 + D3 + D6 + D7

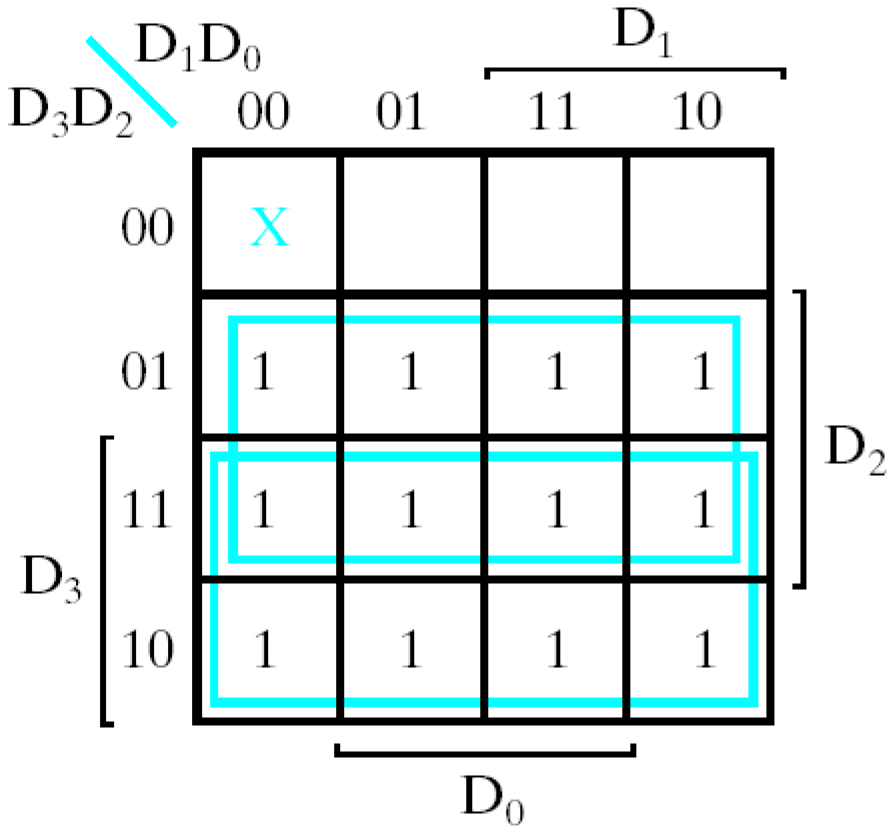
Priority Encoder

- A priority encoder is a combinational circuit that implements a priority function
- Octal-to-Binary: If more than one input is active simultaneously, the output produces an incorrect combination
- Priority Encoder: If two or more inputs are active simultaneously, the input having the highest priority takes precedence

TABLE 4-5
Truth Table of Priority Encoder

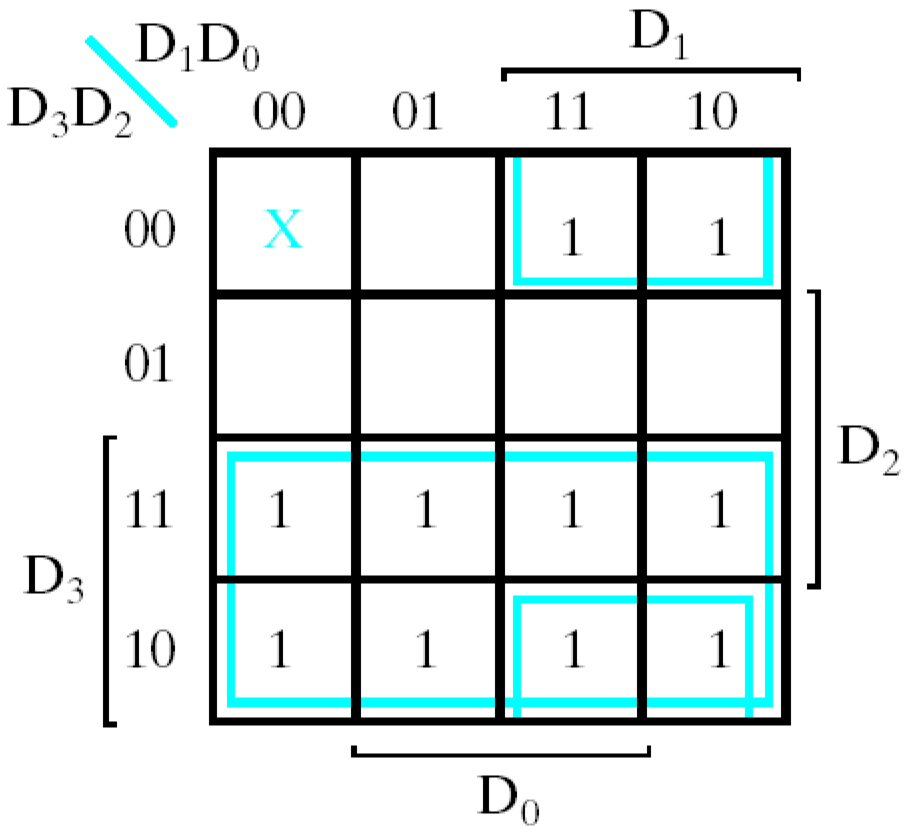
Inputs				Outputs		
D_3	D_2	D_1	D_0	A_1	A_0	V
0	0	0	0	X	X	0
0	0	0	1	0	0	1
0	0	1	X	0	1	1
0	1	X	X	1	0	1
1	X	X	X	1	1	1

4-Input Priority Encoder



$D_3D_2 \backslash D_1D_0$	00	01	11	10
00	X			
01	1	1	1	1
11	1	1	1	1
10	1	1	1	1

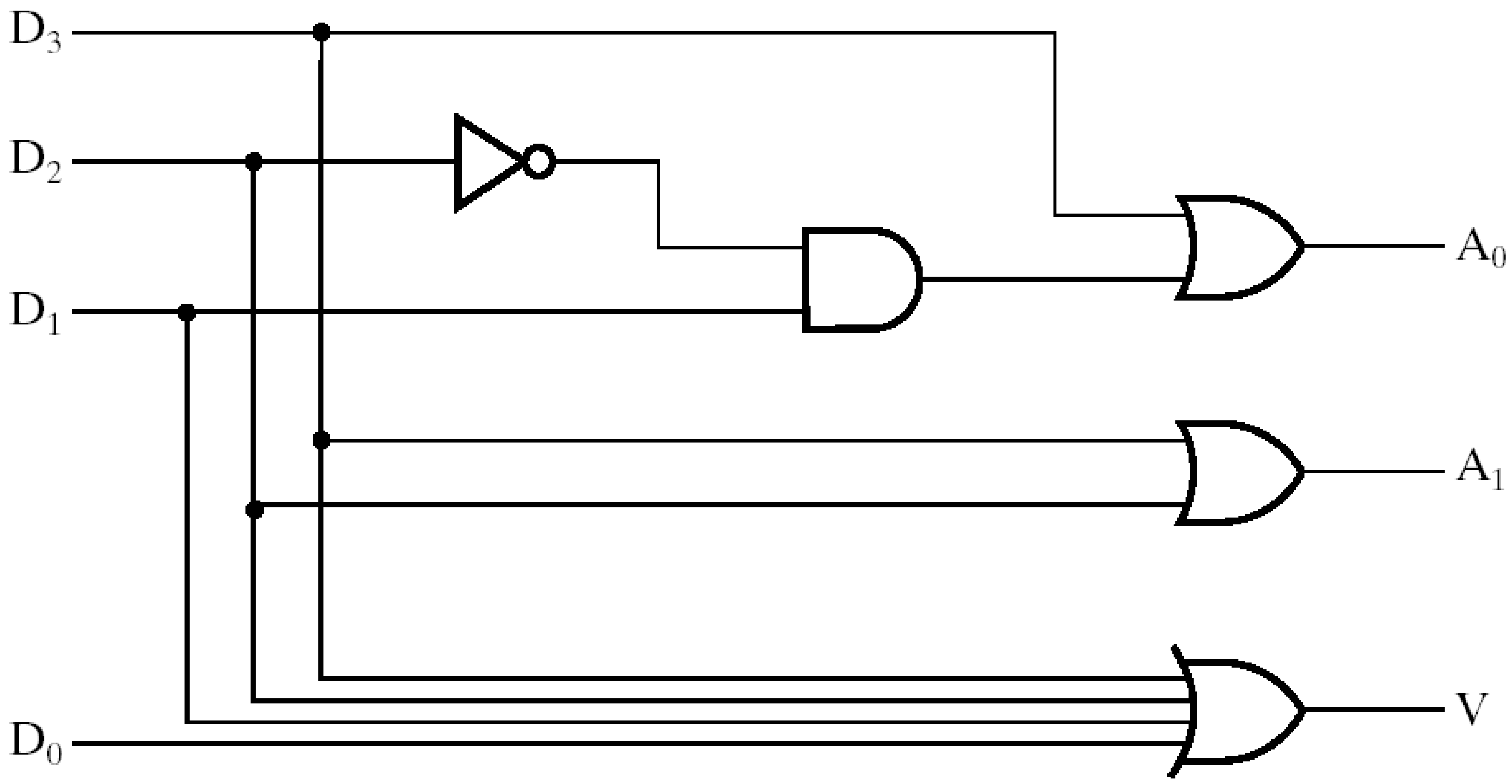
$A_1 = D_2 + D_3$



$D_3D_2 \backslash D_1D_0$	00	01	11	10
00	X		1	1
01				
11	1	1	1	1
10	1	1	1	1

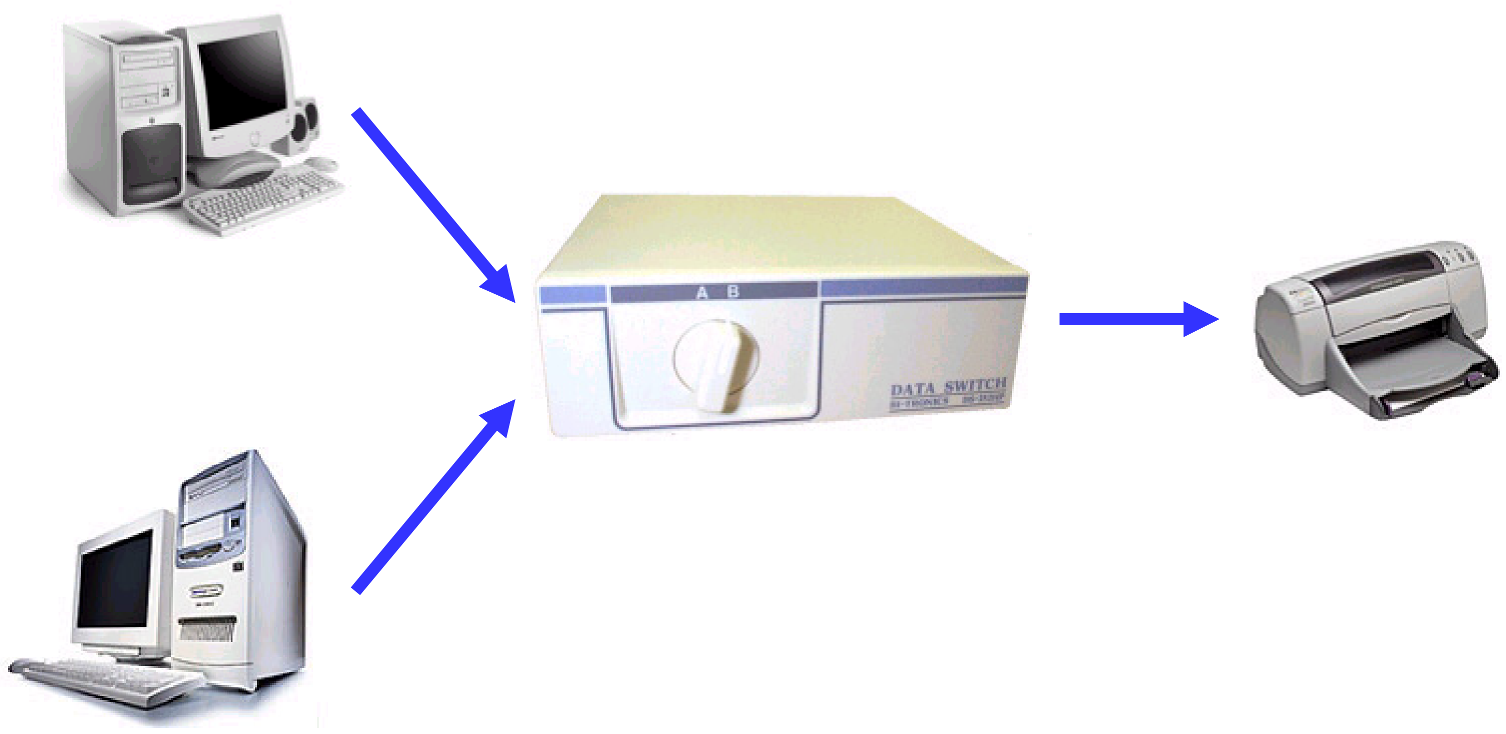
$A_0 = D_3 + D_1\overline{D}_2$

4-Input Priority Encoder



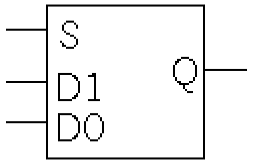
Multiplexers

- 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.



Multiplexers

- A 2^n-to-1 multiplexer sends one of 2^n input lines to a single output line
  - A multiplexer has two sets of *inputs*:
    - 2^n data input lines
    - n select lines, to pick one of the 2^n data inputs
  - The mux *output* is a single bit, which is one of the 2^n data inputs
- The simplest example is a 2-to-1 mux:

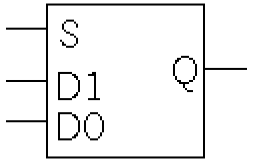


$Q = S' D0 + S D1$

- The select bit S controls which of the data bits D0-D1 is chosen:
  - If S=0, then D0 is the output (Q=D0).
  - If S=1, then D1 is the output (Q=D1).

More truth table abbreviations

- Here is a full truth table for this 2-to-1 mux, based on the equation:



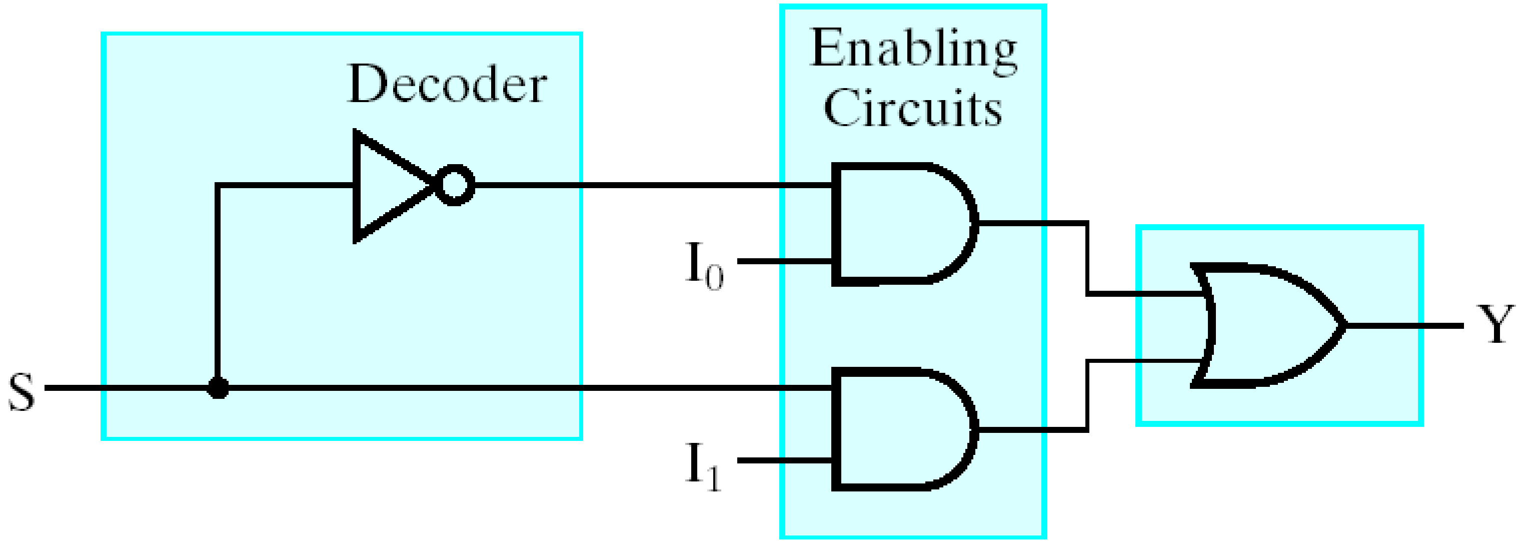
$Q = S'\,D0 + S\,D1$

S	D1	D0	Q
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

- Here is another kind of abbreviated truth table
  - Input variables appear in the output column
  - This table implies that when S=0, the output Q=D0, and when S=1 the output Q=D1
  - This is a pretty close match to the equation

S	Q
0	D0
1	D1

2-to-1 Mux

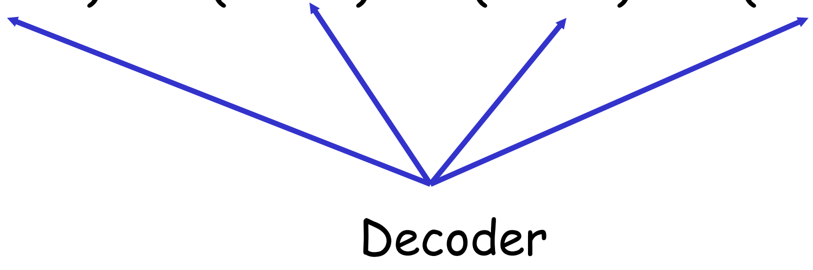


4-to-1 Mux

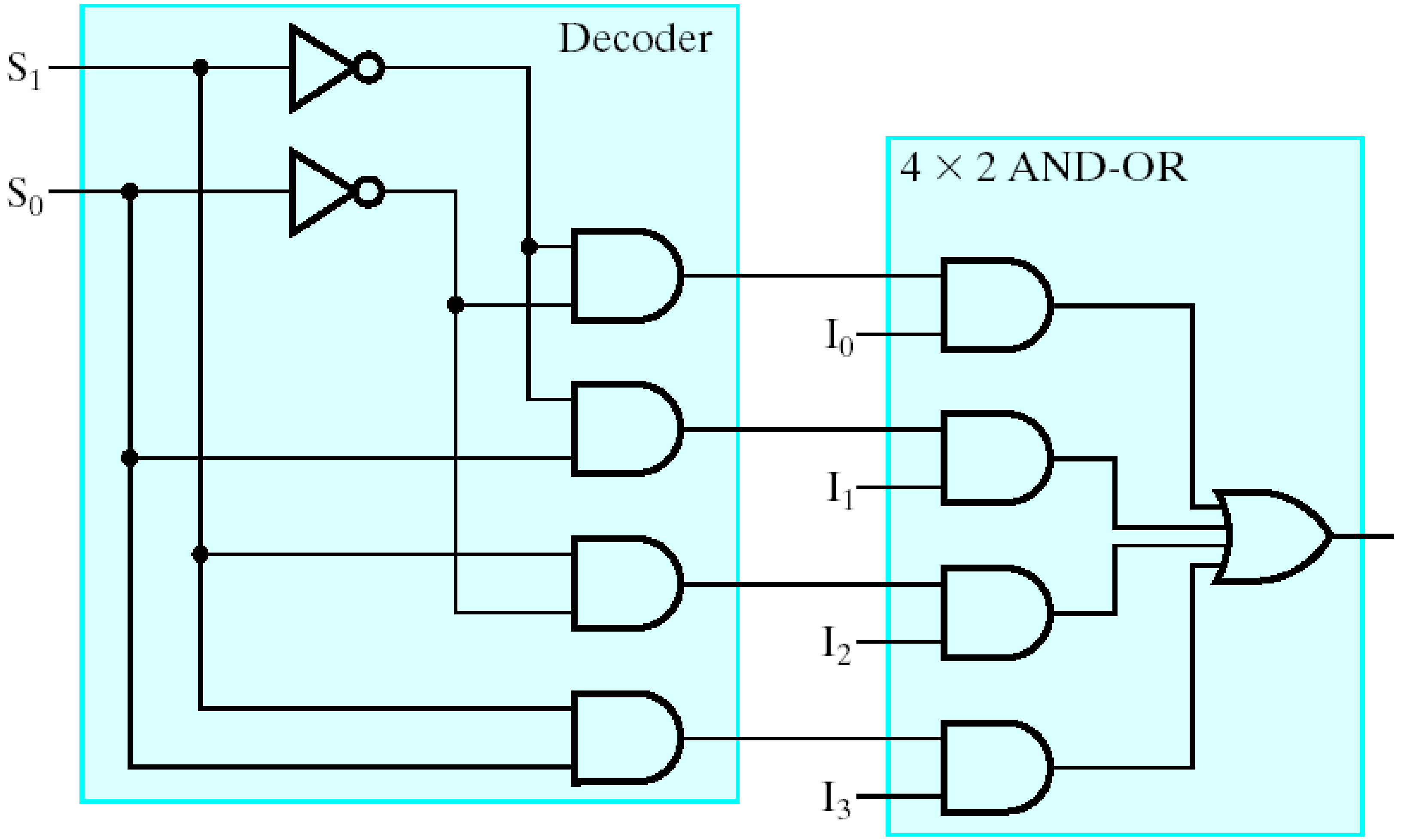
- **TABLE 4-7**
  Condensed Truth Table for 4-to-1-Line Multiplexer

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

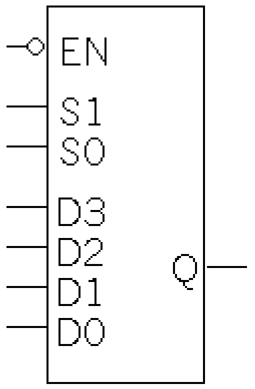
Y= (S1'S0')I0 + (S1'S0)I1 + (S1S0')I2 + (S1S0)I3



4-to-1 Mux



4-to-1 Mux with Enable Input

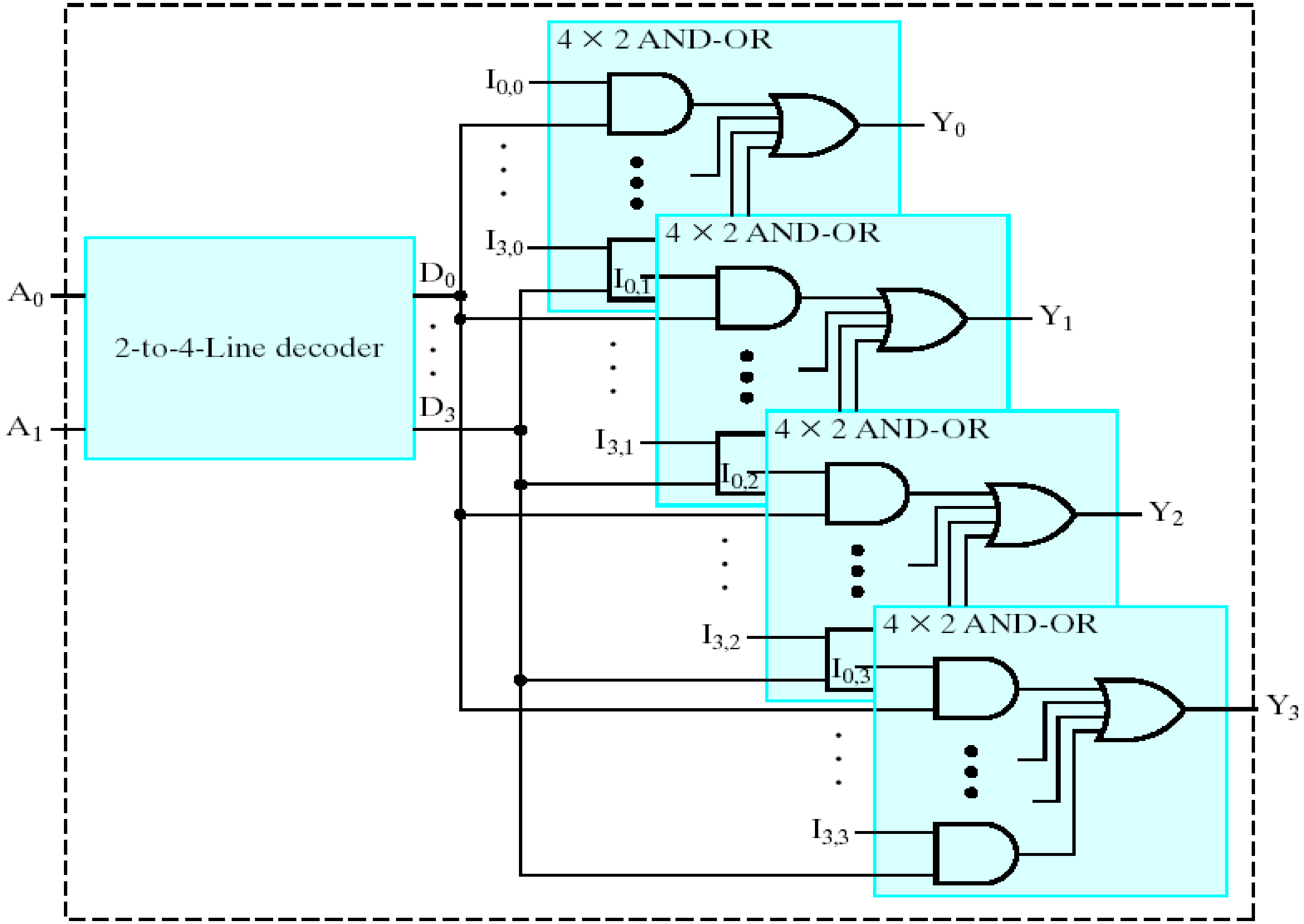


EN	S1	S0	Q
0	0	0	D0
0	0	1	D1
0	1	0	D2
0	1	1	D3
1	x	x	1

Q = EN'S1' S0' D0 + EN'S1' S0 D1 + EN'S1 S0' D2

+ EN'S1 S0 D3 + EN

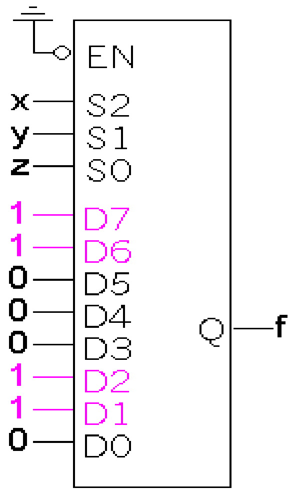
Quad 4-to-1 Mux



Implementing functions with multiplexers

- Muxes can be used to implement arbitrary functions
- One way to implement a function of *n* variables is to use an 2^*n-to-1* mux:
  - For each minterm m_i of the function, connect 1 to mux data input Di. Each data input corresponds to one row of the truth table
  - Connect the function's input variables to the mux select inputs. These are used to indicate a particular input combination
- For example, let's look at f(x,y,z) = Σm(1,2,6,7).

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



A more efficient way

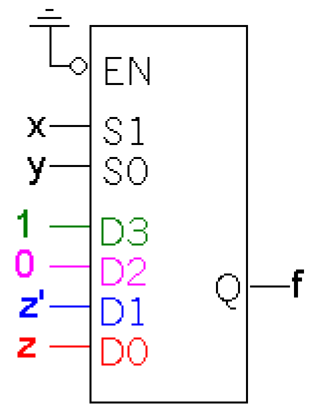
- We can actually implement f(x,y,z) = Σm(1,2,6,7) with just a 4-to-1 mux, instead of an 8-to-1

- 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.

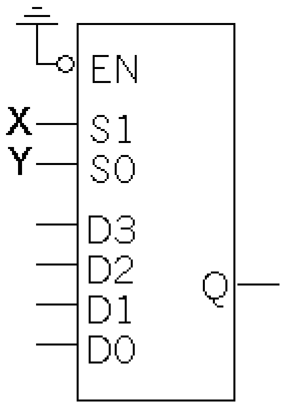
x	y	z	f
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1



Example: multiplexer-based adder

- Let's implement the adder carry function, C(X,Y,Z), with muxes
- There are three inputs, so we'll need a 4-to-1 mux
- The basic setup is to connect two of the input variables (usually the first two in the truth table) to the mux select inputs

X	Y	Z	C
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1



With S1=X and S0=Y, then

Q=X'Y'D0 + X'YD1 + XY'D2 + XYD3

Multiplexer-based carry

- We can set the multiplexer data inputs D0-D3, by fixing X and Y and finding equations for C in terms of just Z.

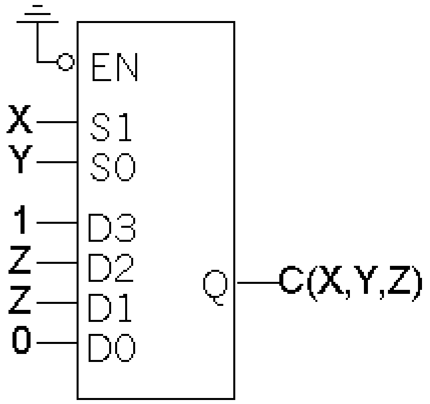
X	Y	Z	C
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

When XY=00, C=0

When XY=01, C=Z

When XY=10, C=Z

When XY=11, C=1



$$\begin{aligned} C &= X'\,Y'\,D0 + X'\,Y\,D1 + X\,Y'\,D2 + X\,Y\,D3 \\ &= X'\,Y'\,0 + X'\,Y\,Z + X\,Y'\,Z + X\,Y\,1 \\ &= X'\,Y\,Z + X\,Y'\,Z + XY \\ &= \Sigma m(3,5,6,7) \end{aligned}$$

Multiplexer-based sum

- Here's the same thing, but for the sum function S(X,Y,Z)

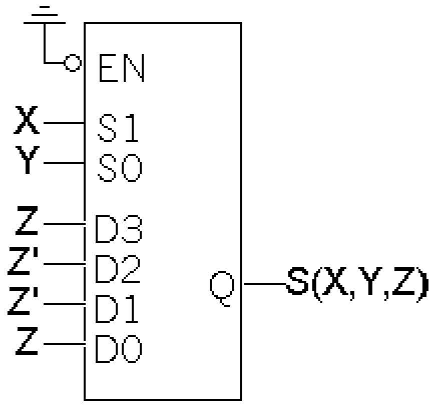
X	Y	Z	S
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

When XY=00, S=Z

When XY=01, S=Z'

When XY=10, S=Z'

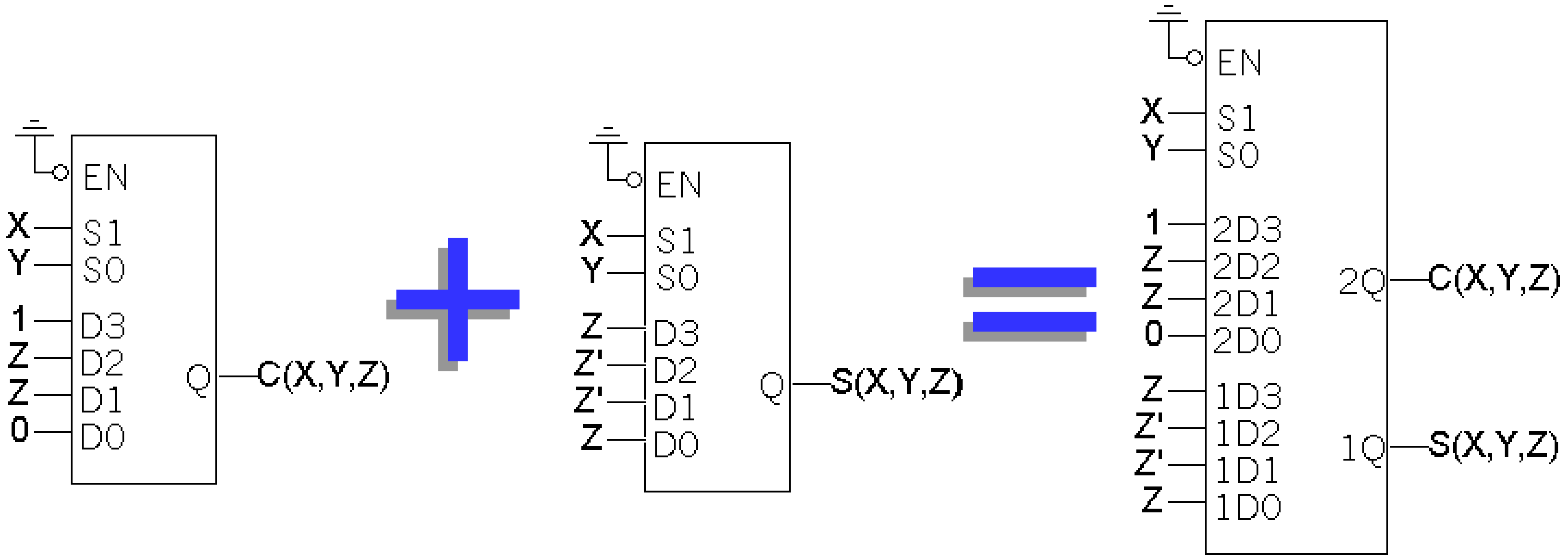
When XY=11, S=Z



$$\begin{aligned} S &= X'\,Y'\,D0 + X'\,Y\,D1 + X\,Y'\,D2 + X\,Y\,D3 \\ &= X'\,Y'\,Z + X'\,Y\,Z' + X\,Y'\,Z' + X\,Y\,Z \\ &= \Sigma m(1,2,4,7) \end{aligned}$$

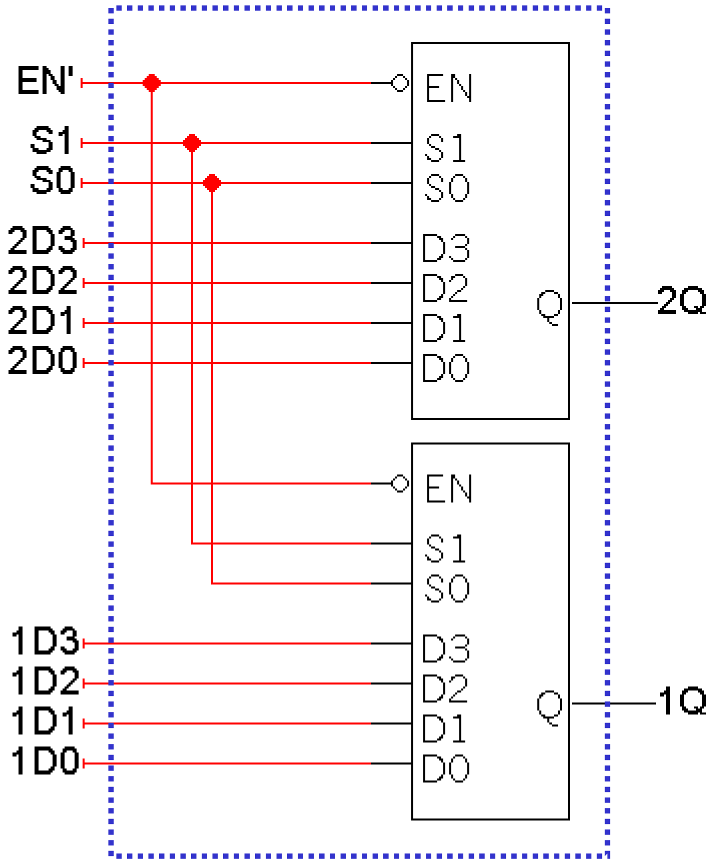
Dual multiplexer-based full adder

- We need *two* separate 4-to-1 muxes: one for C and one for S
- But sometimes it's convenient to think about the adder output as being a single 2-bit number, instead of as two separate functions
- A dual 4-to-1 mux gives the illusion of 2-bit data inputs and outputs
  - It's really just two 4-to-1 muxes connected together

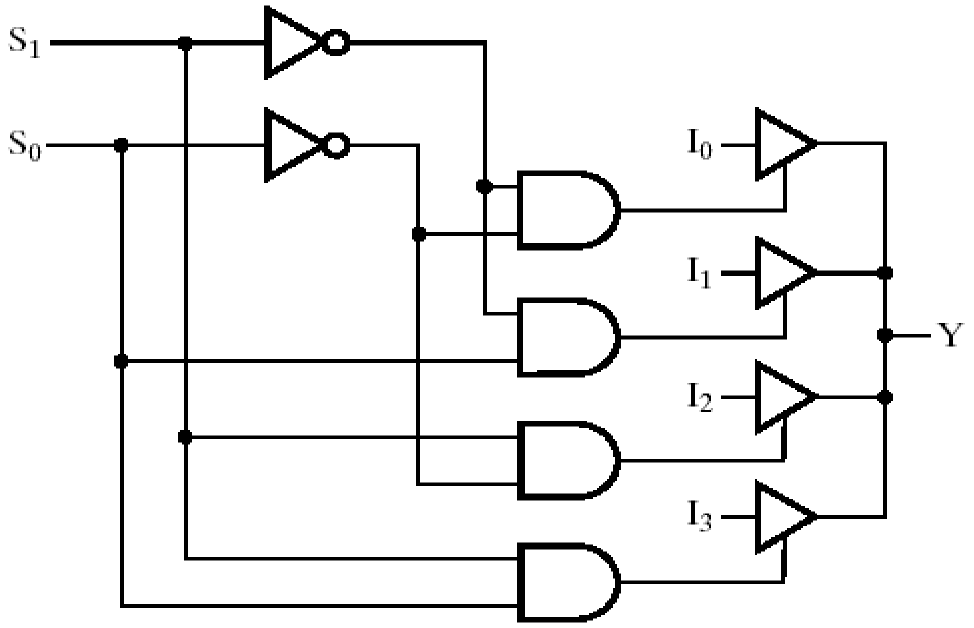


Dual muxes in more detail

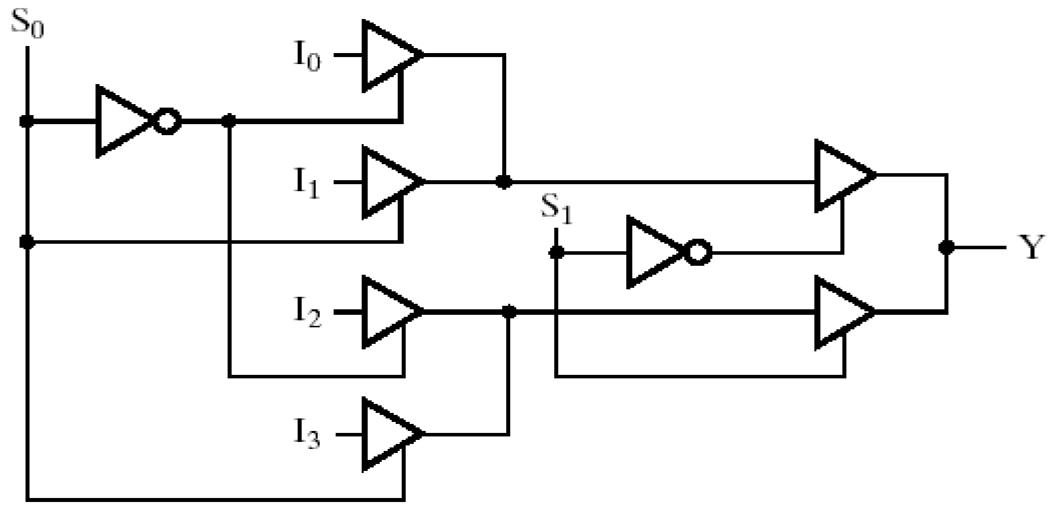
- You can make a dual 4-to-1 mux by connecting two 4-to-1 muxes. (“Dual” means “two-bit values.”)
- In the diagram on the right, we’re using S1-S0 to choose one of the following *pairs* of inputs:
  - 2D3 1D3, when S1 S0 = 11
  - 2D2 1D2, when S1 S0 = 10
  - 2D1 1D1, when S1 S0 = 01
  - 2D0 1D0, when S1 S0 = 00



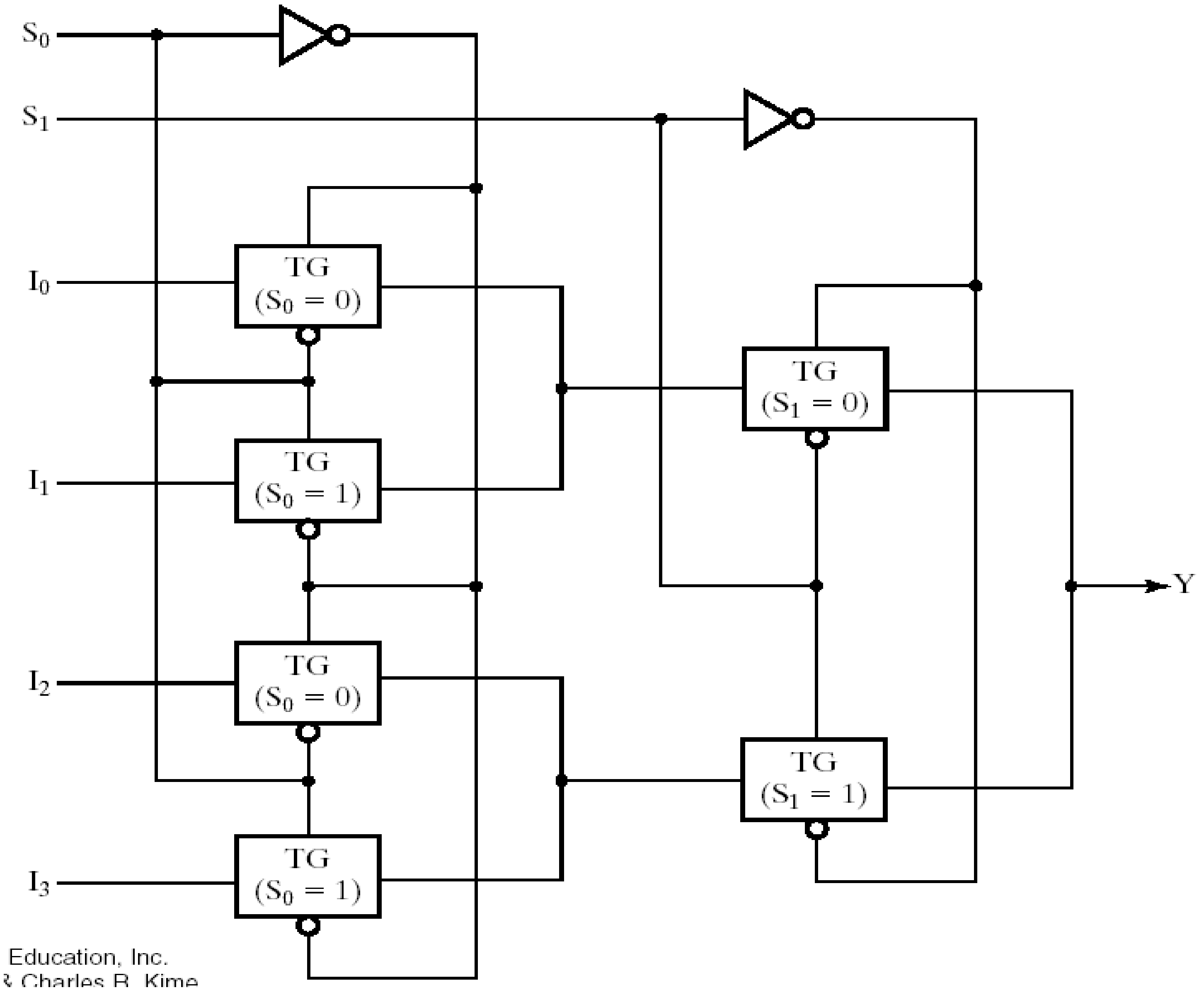
Selecting based on 3-state buffers



(a)



Selecting based on TGs



Summary of Muxes

- A 2^n-to-1 multiplexer routes one of 2^n input lines to a single output line
- Just like decoders,
  - Muxes are common enough to be supplied as stand-alone devices for use in modular designs.
  - Muxes can implement arbitrary functions
- We saw some variations of the standard multiplexer:
  - Smaller muxes can be combined to produce larger ones
  - We can add active-low or active-high enable inputs
- As always, we use truth tables and Boolean algebra to analyze things