College of Computer and Information Sciences
Department of Computer Science

CSC 220: Computer Organization

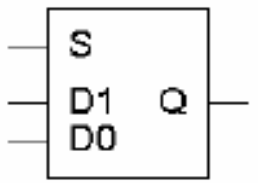
Unit 6
COMBINATIONAL CIRCUITS-2

Objectives

- Multiplexers
- DeMultiplexers
- Decoders
- Encoders

A 2-to-1 multiplexer

- Here is the circuit analog of that printer switch.



- This is a 2-to-1 multiplexer, or mux.
  - There are two data inputs D0 and D1, and a select input called S.
  - There is one output named Q.
- The multiplexer routes one of its data inputs (D0 or D1) to the output Q, based on the value of S.
  - If S=0, the output will be D0.
  - If S=1, the output will be D1.

Building a multiplexer

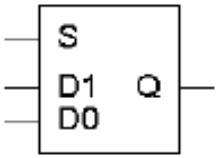
- Here is a truth table for the multiplexer, based on our description from the previous page:

> The multiplexer routes one of its data inputs (D0 or D1) to the output Q, based on the value of S.
> - If S=0, the output will be D0.
> - If S=1, the output will be 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

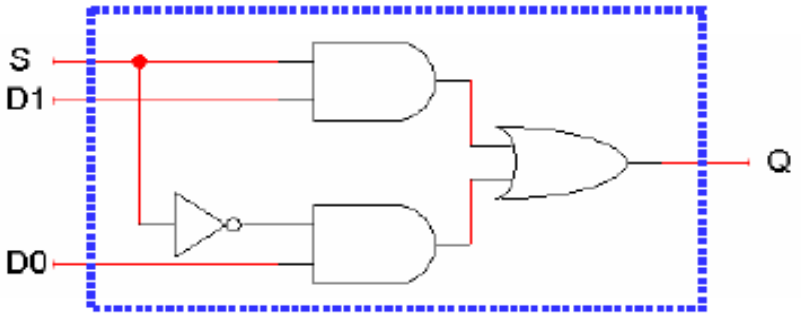
- You can then find an MSP for the mux output Q.

- Note that this corresponds closely to our English specification above—sometimes you can derive an expression without first making a truth table.



Multiplexer circuit diagram

- Here is an implementation of a 2-to-1 multiplexer.



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

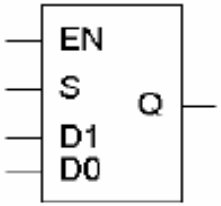
- Remember that a minimal sum of products expression leads to a minimal two-level circuit.

Blocks, abstraction and modularity

- Multiplexers are common enough that we often want to treat them as abstract units or black boxes, as symbolized by our block diagrams.
  - Block symbols make circuit diagrams simpler, by hiding the internal implementation details. You can use a device without knowing how it's designed, as long as you know what it does.
  - Different multiplexer implementations should be interchangeable.
  - Circuit blocks also aid hardware re-use, since you don't have to keep building a multiplexer from scratch every time you need one.
- These blocks are similar to functions in programming languages!

Enable inputs

- Many devices have an additional enable input, which "activates" or "deactivates" the device.
- We could design a 2-to-1 multiplexer with an enable input that's used as follows.
  - EN=0 disables the multiplexer, which forces the output to be 0. (It does *not* turn off the multiplexer.)
  - EN=1 enables the multiplexer, and it works as specified earlier.
- Enable inputs are especially useful in combining smaller muxes together to make larger ones, as we'll see later today.



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

Truth table abbreviations

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

- Notice that when EN=0, then Q is always 0, regardless of what S, D1 and D0 are set to.
- We can shorten the truth table by including Xs in the input variable columns, as shown on the bottom right.

EN	S	D1	D0	Q
0	x	x	x	0
1	0	0	0	0
1	0	0	1	1
1	0	1	0	0
1	0	1	1	1
1	1	0	0	0
1	1	0	1	0
1	1	1	0	1
1	1	1	1	1

Another abbr. 4 U

- Also, when EN=1 notice that if S=0 then Q=D0, but if S=1 then Q=D1.
- Another way to abbreviate a truth table is to list input variables in the output columns, as shown on the right.

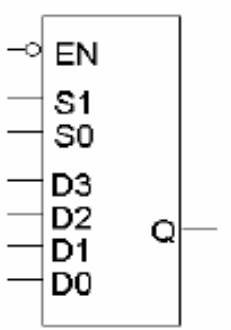
EN	S	D1	D0	Q
0	x	x	x	0
1	0	0	0	0
1	0	0	1	1
1	0	1	0	0
1	0	1	1	1
1	1	0	0	0
1	1	0	1	0
1	1	1	0	1
1	1	1	1	1

EN	S	Q
0	x	0
1	0	D0
1	1	D1

- This final version of the 2-to-1 multiplexer truth table is much clearer, and matches the equation Q = S'D0 + S D1 very closely.

A 4-to-1 multiplexer

- Here is a block diagram and abbreviated truth table for a 4-to-1 mux, which directs one of four different inputs to the single output line.
  - There are four data inputs, so we need *two* bits, S1 and S0, for the mux selection input.
  - LogicWorks multiplexers have active-low enable inputs, so the mux always outputs 1 when EN' = 1. This is denoted on the block symbol with a bubble in front of EN.

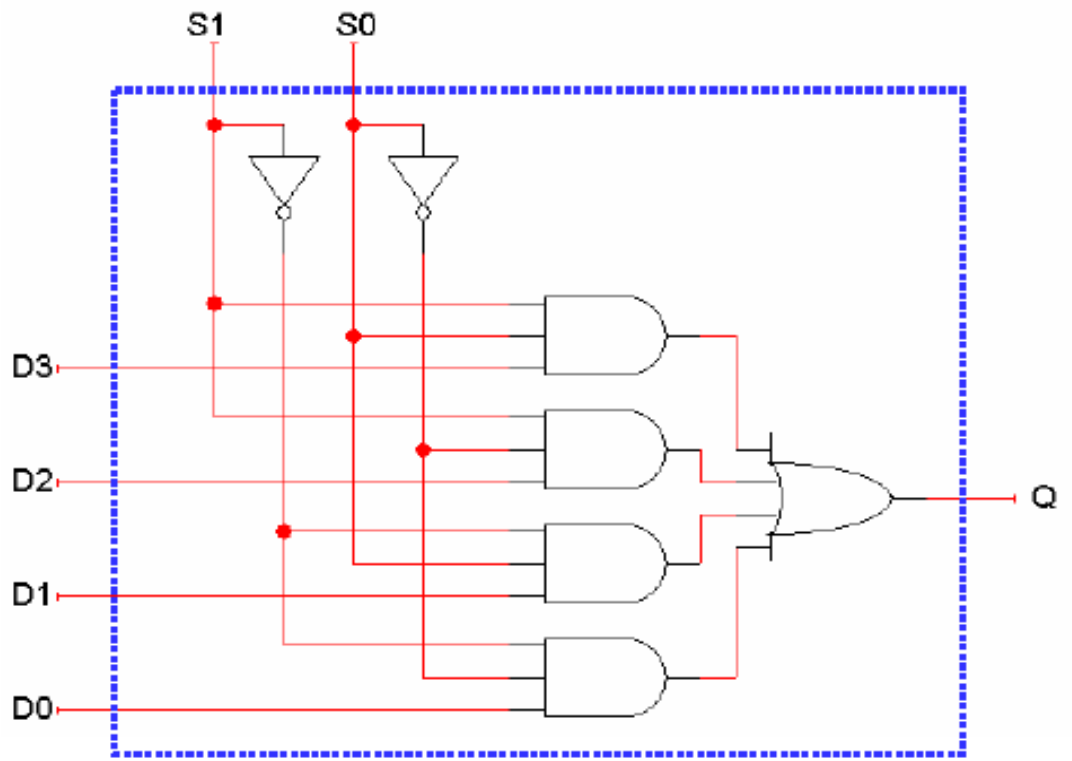


EN'	S1	S0	Q
0	0	0	D0
0	0	1	D1
0	1	0	D2
0	1	1	D3
1	x	x	1

$$Q = S1'S0'D0 + S1'S0\,D1 + S1\,S0'D2 + S1\,S0\,D3$$

A 4-to-1 multiplexer implementation

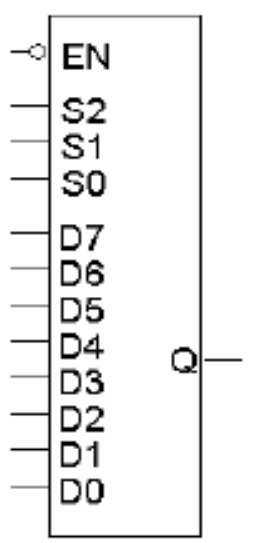
- Again we have a minimal sum of products expression, which leads to a minimal two-level circuit implementation.



$Q = S1'S0'D0 + S1'S0\,D1 + S1\,S0'D2 + S1\,S0\,D3$

2^n-to-1 multiplexers

- You can make even larger multiplexers, following the same pattern.
- A 2^n-to-1 multiplexer routes one of 2^n input lines to the output line.
  - There are 2^n data inputs, so there must also be *n* select inputs.
  - The output is a single bit.
- Here is an 8-to-1 multiplexer, probably the biggest we'll see in this class.



Implementing Functions with Multiplexers

Example: addition

- Multiplexers can sometimes make circuit design easier.
- As an example, let's make a circuit to add three 1-bit inputs X, Y and Z.
- We'll need two bits to represent the total.
  - The bits will be called C and S, standing for "carry" and "sum."
  - These are two separate functions of the inputs X, Y and Z.
- A truth table and sum of minterm equations for C and S are shown below.

	X	Y	Z	C	S
	0	0	0	0	0
	0	0	1	0	1
	0	1	0	0	1
0 + 1 + 1 = 10 →	0	1	1	1	0
	1	0	0	0	1
	1	0	1	1	0
	1	1	0	1	0
1 + 1 + 1 = 11 →	1	1	1	1	1

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

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

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

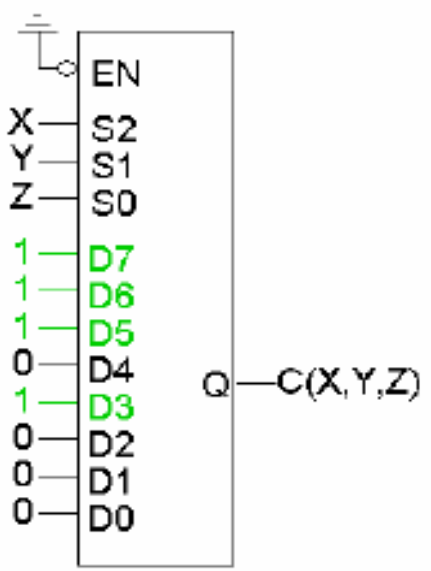
0 + 1 + 1 = 10 →

1 + 1 + 1 = 11 →

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

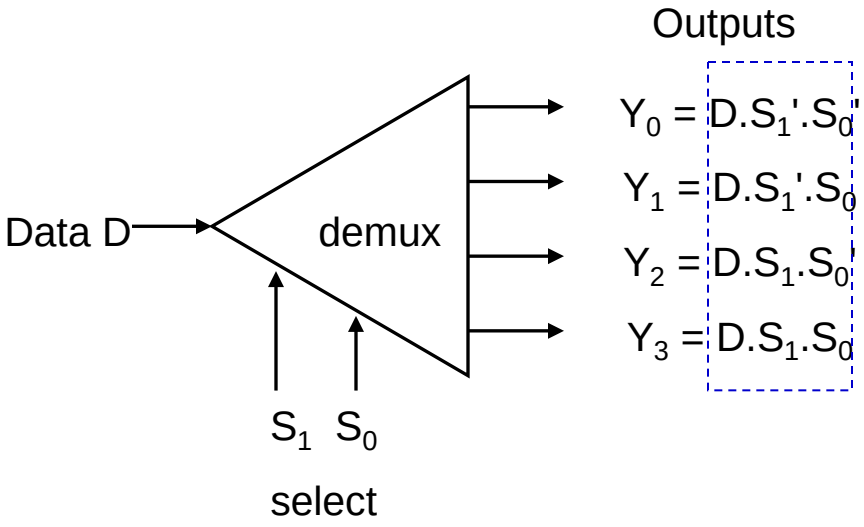
$S(X,Y,Z) = \Sigma m(1,2,4,7)$

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



Demultiplexer

- Given an input line and a set of selection lines, the demultiplexer will direct data from input to a selected output line.
- An example of a 1-to-4 demultiplexer:



S_1	S_0	Y_0	Y_1	Y_2	Y_3
0	0	D	0	0	0
0	1	0	D	0	0
1	0	0	0	D	0
1	1	0	0	0	D

Demultiplexer

- Takes one input
- Out to one of 2^n possible outputs

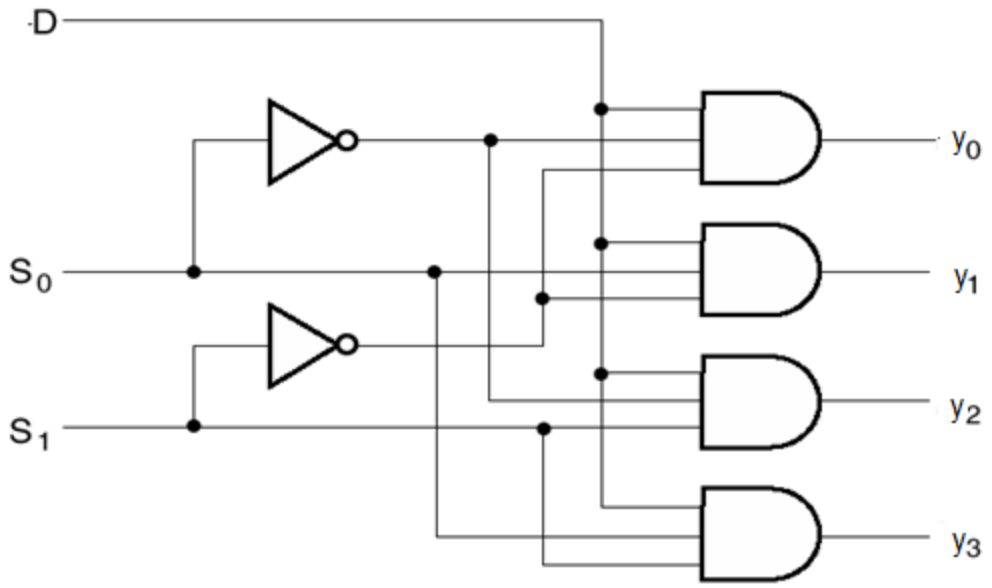
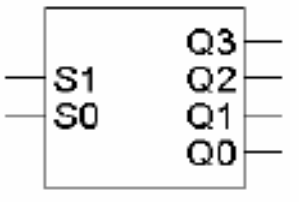


Fig. 3-24 1-to-4-Line Demultiplexer

Decoder

What a decoder does

- A *n*-to-2^n decoder uses its *n*-bit input to determine which of 2^n outputs will be uniquely activated.



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

- Here is a block diagram and truth table for a 2-to-4 decoder.
  - The two-bit input is called S1S0, and the four outputs are Q0-Q3.
  - If the input is the binary number i, then output Qi alone will be true.
- This circuit "decodes" a binary number into a "one-of-four" code.

Building a decoder

- We can use the truth table to derive minimal sum of products equations for each of the four outputs (Q0-Q3), based on the two inputs (S0-S1).

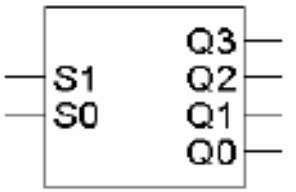
S1	S0	Q0	Q1	Q2	Q3
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1

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

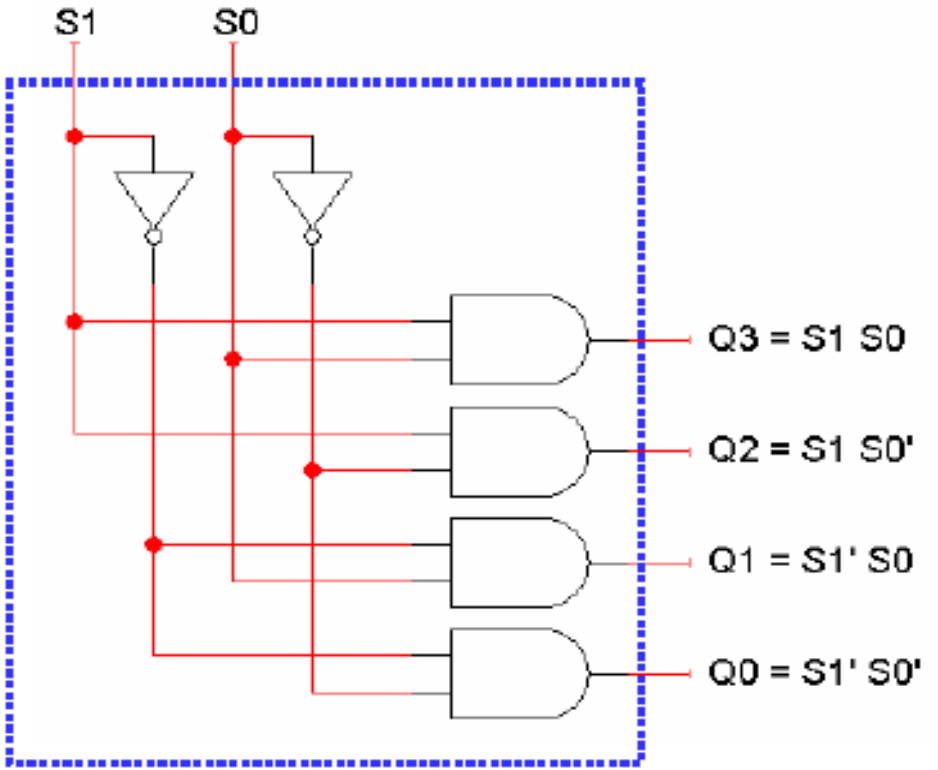
Q0 = S1'S0'
Q1 = S1'S0
Q2 = S1 S0'
Q3 = S1 S0

Decoder circuit diagram

- Here is an implementation of a 2-to-4 decoder.

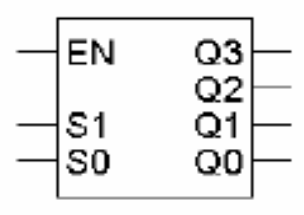


S1	S0	Q0	Q1	Q2	Q3
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1



Enable inputs

- Just as with multiplexers, decoders can include enable inputs.
  - EN=0 disables the decoder, which by convention means that all of the decoder's outputs are 0.
  - EN=1 enables the decoder so that it behaves as specified earlier, with exactly one of the outputs being 1.



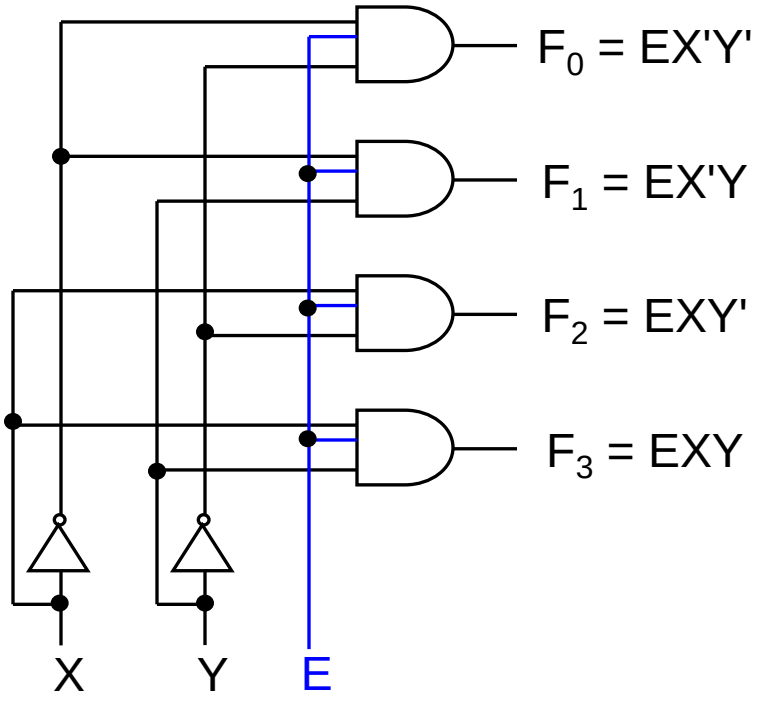
EN	S1	S0	Q0	Q1	Q2	Q3
0	x	x	0	0	0	0
1	0	0	1	0	0	0
1	0	1	0	1	0	0
1	1	0	0	0	1	0
1	1	1	0	0	0	1

Decoders with Enable (1/2)

- Decoders often come with an enable signal, so that the device is only activated when the enable, E=1.
- Truth table:

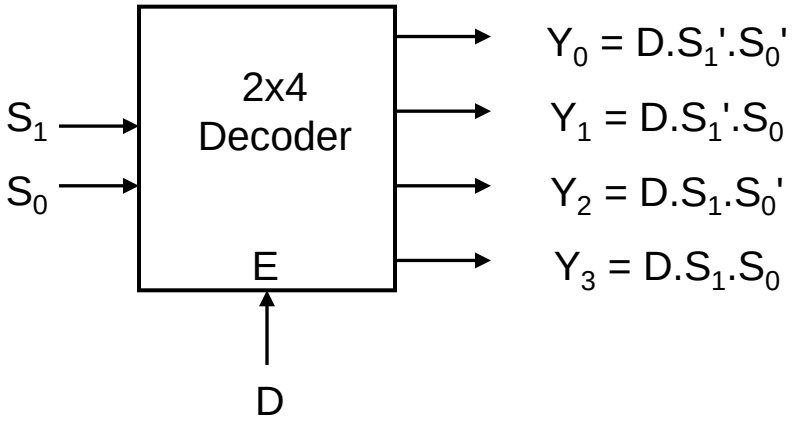
E	X	Y	F_0	F_1	F_2	F_3
1	0	0	1	0	0	0
1	0	1	0	1	0	0
1	1	0	0	0	1	0
1	1	1	0	0	0	1
0	X	X	0	0	0	0

- Circuit:



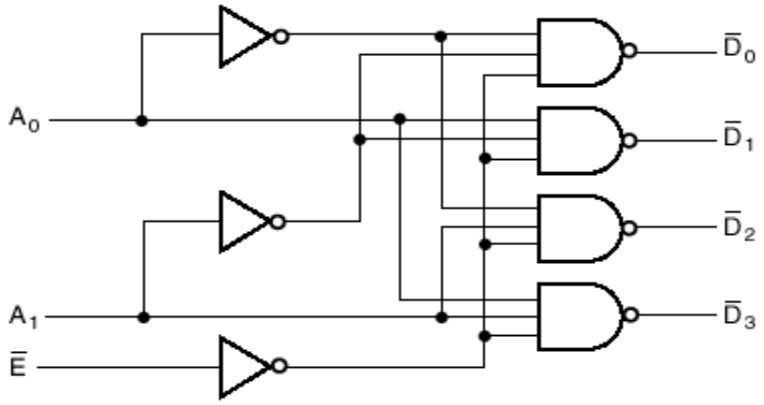
Demultiplexer Vs Decoder

- **The demultiplexer is actually identical to a decoder with enable, as illustrated below:**



Exercise: Provide the truth table for above demultiplexer.

A Demux Using NAND Gates: A Decoder with an Enable

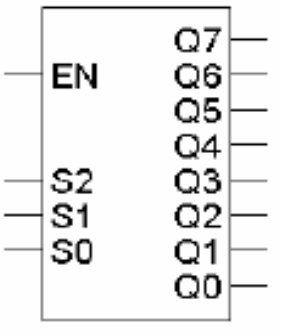


(a) Logic diagram

$\overline{E}$	A_1	A_0	$\overline{D}_0$	$\overline{D}_1$	$\overline{D}_2$	$\overline{D}_3$
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

A 3-to-8 decoder

- Larger decoders are similar. Here is a 3-to-8 decoder.
  - There are three selection inputs S2S1S0, which activate one of eight outputs, Q0-Q7.
  - Again, only one output will be true for any input combination.
- A truth table and output equations for a 3-to-8 decoder (without EN) are given below.



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

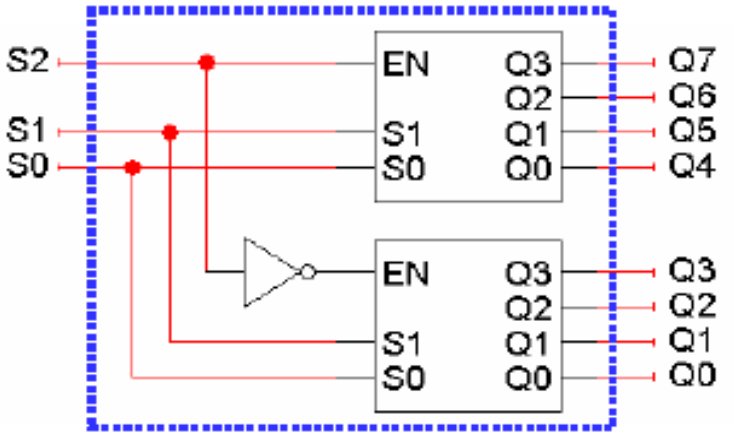
Building a 3-to-8 decoder

- You could build a 3-to-8 decoder from the truth table and MSP equations below, just like we built the 2-to-4 decoder earlier.
- 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

Decoder expansion

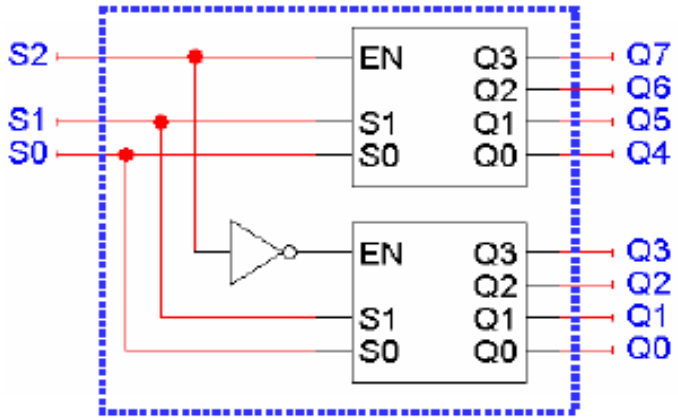
- Here's a 3-to-8 decoder built from two smaller 2-to-4 decoders.
- When S2=0, the bottom 2-to-4 decoder is enabled and generates a 1 for one of outputs Q0, Q1, Q2 or Q3.
- When S2=1, the top 2-to-4 decoder is enabled instead, and a 1 will be output for either Q4, Q5, Q6 or Q7.



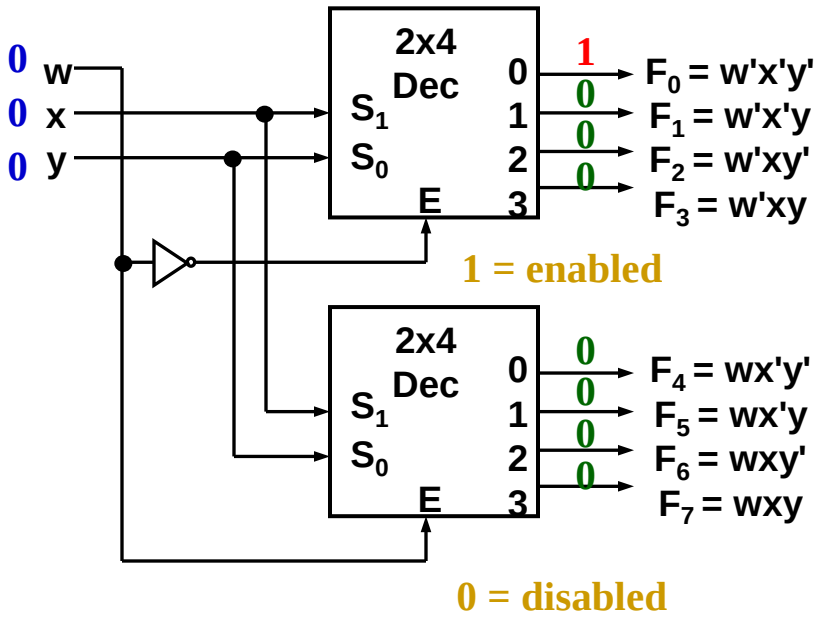
S2	S1	S0	Q0	Q1	Q2	Q3	Q4	Q5	Q6	Q7
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1

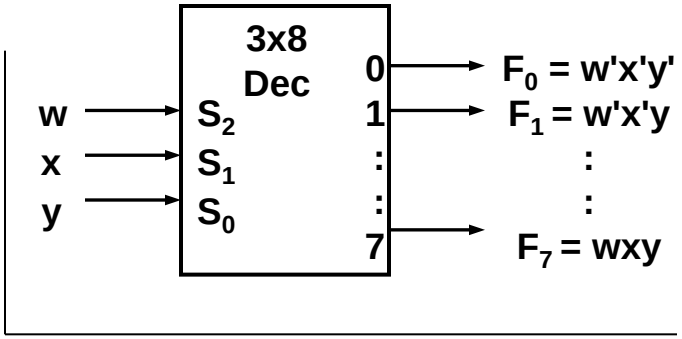
Modularity

- You could verify that this circuit is a 3-to-8 decoder, by using equations for the 2-to-4 decoders to derive equations for the 3-to-8.
- 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.
- This is similar to having several functions in a program which all use a formal parameter "x".

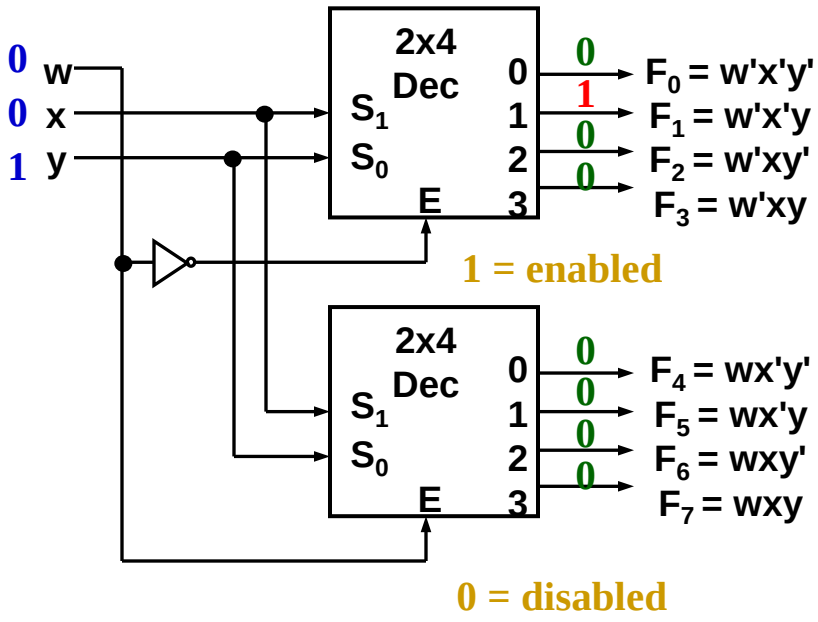


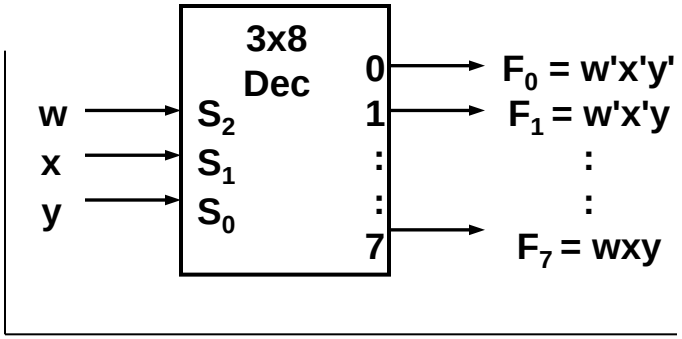
Larger Decoders (2/6)



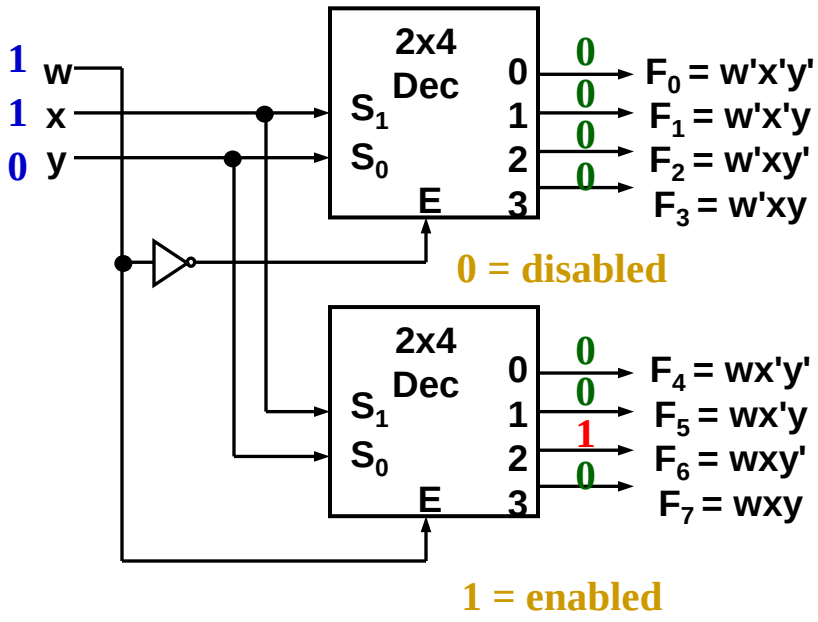


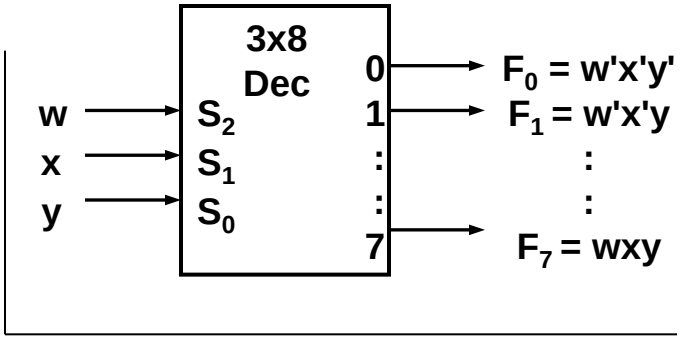
Larger Decoders (3/6)





Larger Decoders (4/6)

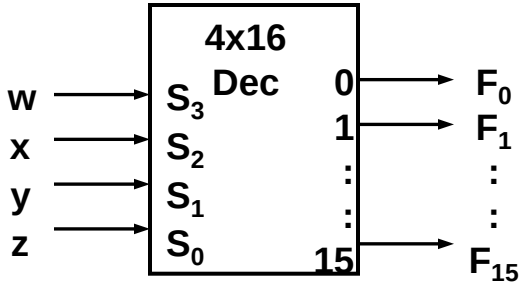


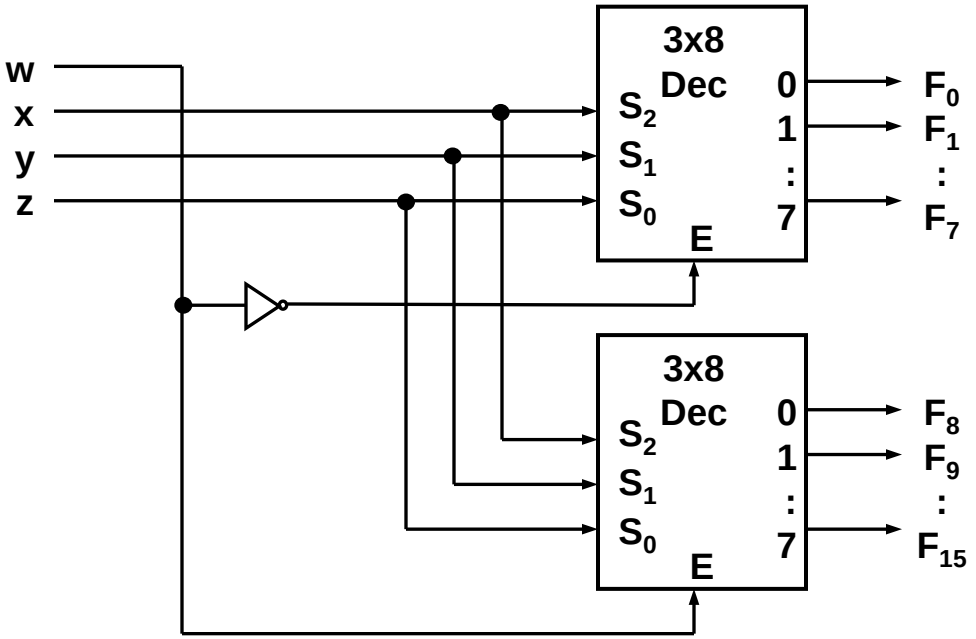


Larger Decoders (5/6)

- Question:

Construct a 4x16 decoder from two 3x8 decoders with 1-enable.





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 input combination, exactly one output is true.
  - Each output equation contains all of the input variables.
- This means that you can easily use a decoder, or a minterm generator, to implement any sum of minterms expression.

Implementing Functions with Decoder

Example: addition

- Yesterday we presented a simple circuit which added three 1-bit inputs X, Y and Z to produce a two-bit output, C ("carry") and S ("sum").
- A truth table and sum of minterm equations for C and S are shown below.

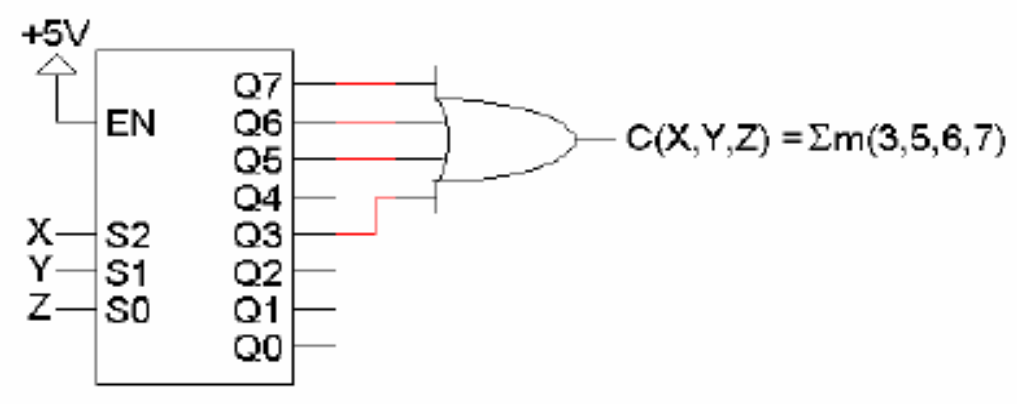
X	Y	Z	C	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

$C(X,Y,Z) = \Sigma m(3,5,6,7)$
$S(X,Y,Z) = \Sigma m(1,2,4,7)$

- Today we'll implement these two functions using 3-to-8 decoders.

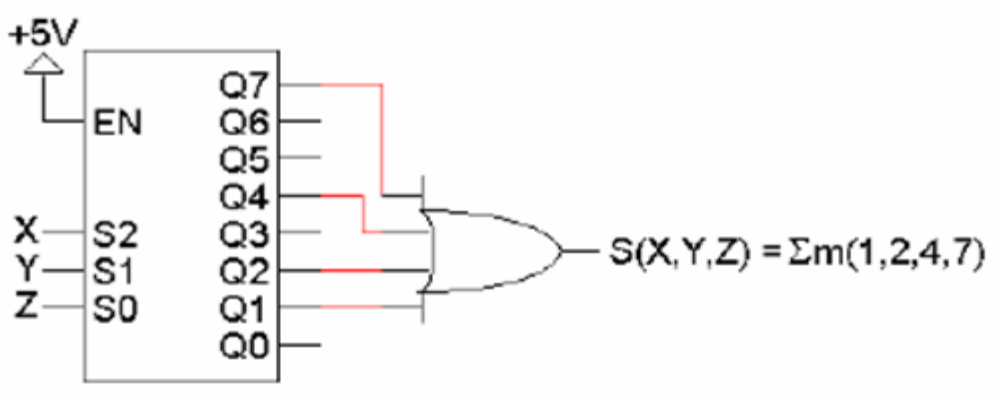
Implementing functions with decoders

- Here, a 3-to-8 decoder implements C as a sum of minterms.



- If XYZ is 011, 101, 110 or 111, then one of the decoder outputs Q3, Q5, Q6 or Q7 will be true, and the output C(X,Y,Z) will also be true.
- The "+5V" symbol ("5 volts") represents a 1 or true in LogicWorks.

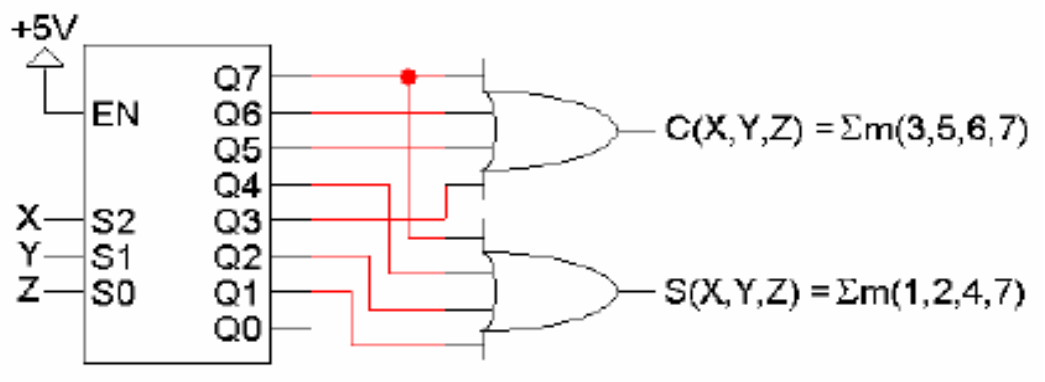
Decoder-based sum



- If XYZ is 001, 010, 100 or 111, then one of decoder outputs Q1, Q2, Q4 or Q7 will be true, and S(X,Y,Z) will be true as well.

Using just one decoder

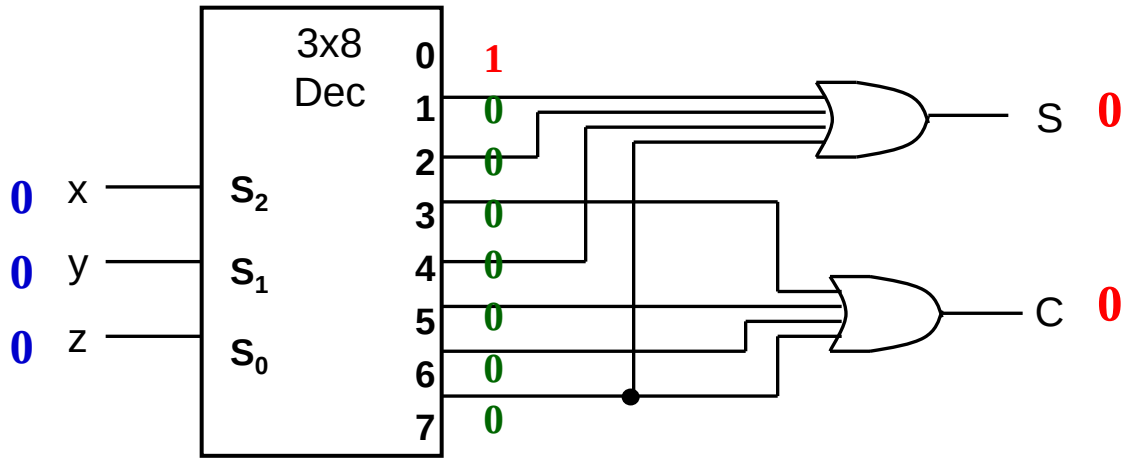
- Since the two functions C and S both have the same inputs, we could use just one decoder instead of two.



- Decoder output Q0 is unused, while Q7 is used multiple times. In general, you can always use circuit outputs as many or as few times as you need.

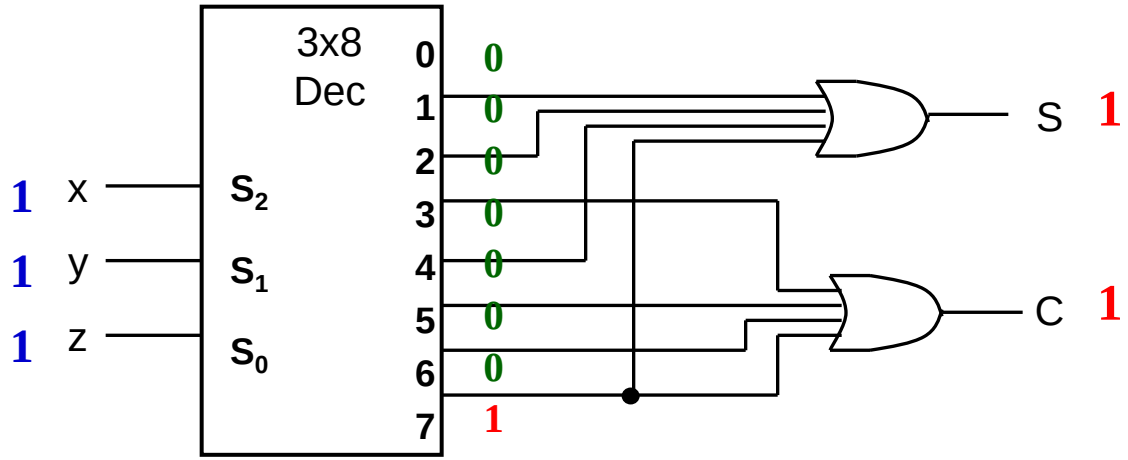
Decoders: Implementing Functions

Example: Full adder



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

Decoders: Implementing Functions

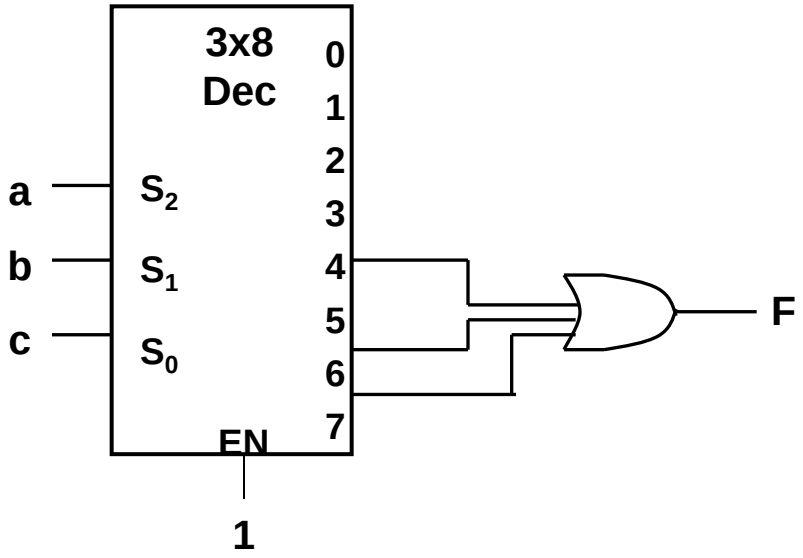


x	y	z	C	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

- Example:

$$F(a,b,c) = \sum m(4,6,7)$$

- Using a 3×8 decoder (assuming 1-enable and active-high outputs).



Encoder

- Encoder is the opposite of decoder
- 2^n inputs
  - or less – 10 inputs in "Decimal to BCD" encoder: I_0, I_1, I_2, I_3, …, I_9
- n outputs
  - 4 output lines "Decimal to BCD"encoder

Truth Table: 8-to-3 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

Inputs are Minterms

- Can OR the minterms appropriately to get each of the outputs A_0, A_1, A_2
- Example: $A_0 = D_1 + D_3 + D_5 + D_7$

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

Generating Outputs using OR of Minterms

- $A_0 = D_1 + D_3 + D_5 + D_7$
- $A_1 = D_2 + D_3 + D_6 + D_7$
- $A_2 = D_4 + D_5 + D_6 + D_7$

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