



CSC 220: Computer Organization

Unit 6

COMBINATIONAL CIRCUITS-2

Prepared by:

Md Saiful Islam, PhD

Department of Computer Science
College of Computer and Information Sciences

Overview

Multiplexers

DeMultiplexers

Decoders

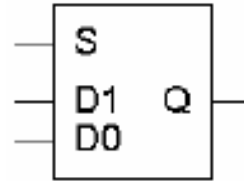
Encoders

Chapter-3

M. Morris Mano, Charles R. Kime and Tom Martin, **Logic and Computer Design Fundamentals**, Global (5th) Edition, Pearson Education Limited, 2016. ISBN: 9781292096124

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.

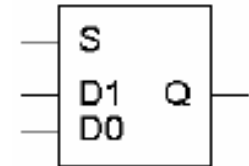
- If S=0, the output will be D0.
- If S=1, the output will be D1.

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

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

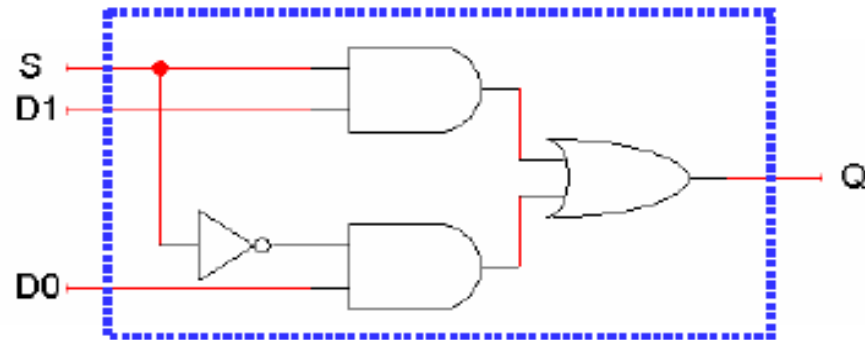
- Note that this corresponds closely to our English specification above—sometimes you can derive an expression without first making a truth table.

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



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.

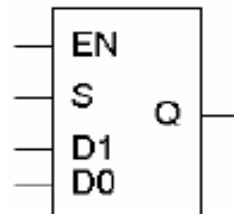
Blocks, abstraction and modularity

- 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



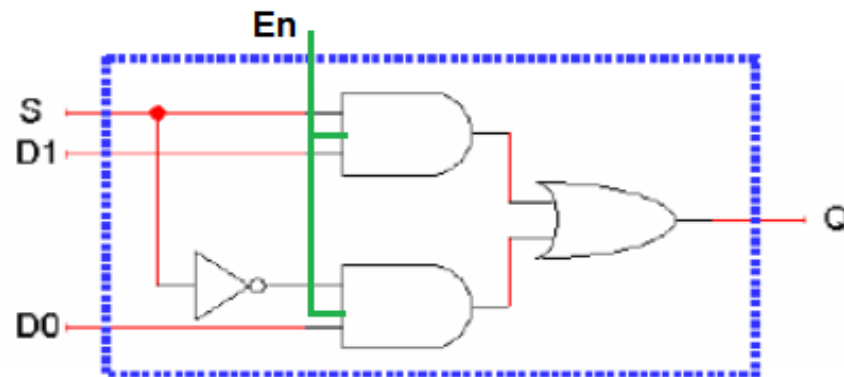
EN	S	Q
0	x	0
1	0	D0
1	1	D1

- 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.

$$Q = EnS'D0 + EnSD1$$

Multiplexer circuit diagram (With Enable)

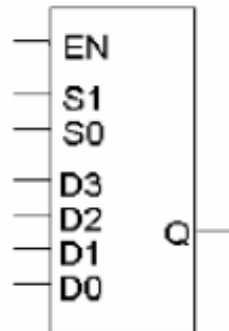
- Here is an implementation of a 2-to-1 multiplexer.



$$Q = EnS'D0 + EnSD1$$

A 4-to-1 multiplexer (Active High)

- 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.



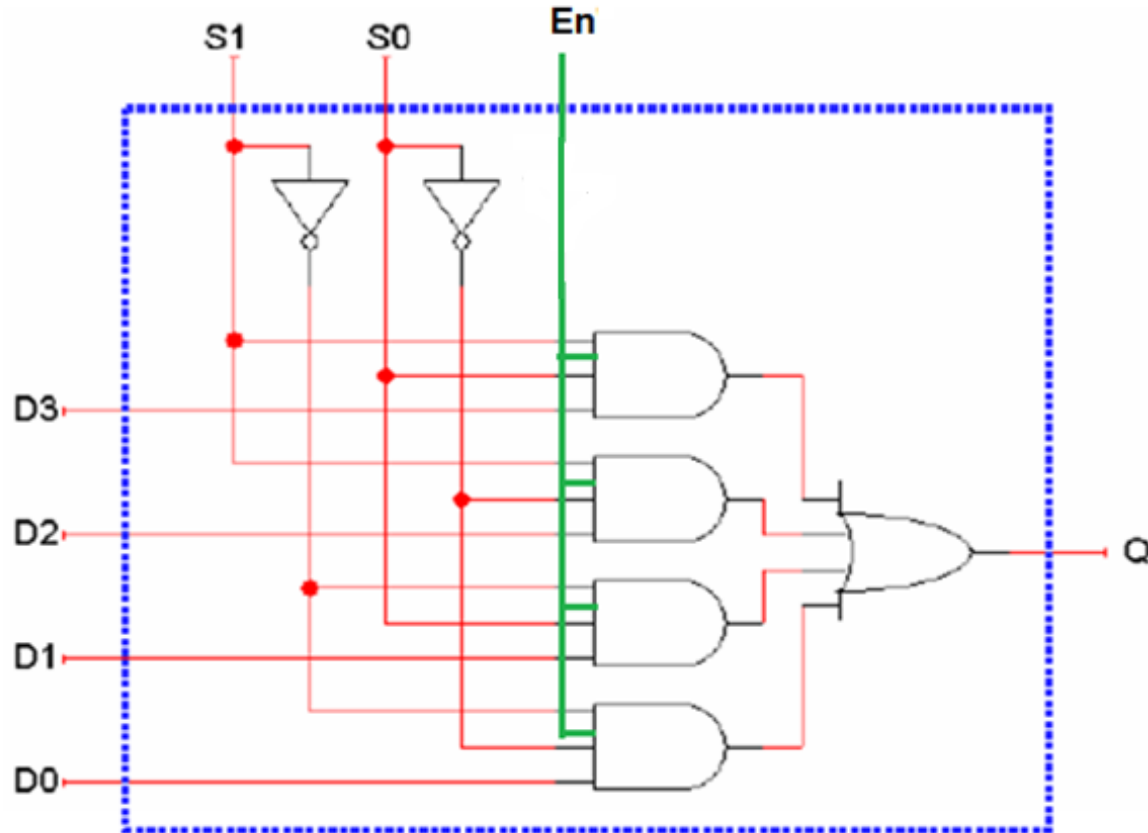
EN	S1	S0	Q
1	0	0	D0
1	0	1	D1
1	1	0	D2
1	1	1	D3
0	x	x	0

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

$$Q = EnS1'S0'D0 + EnS1'S0D1 + EnS1S0'D2 + EnS1S0D3$$

A 4-to-1 multiplexer implementation

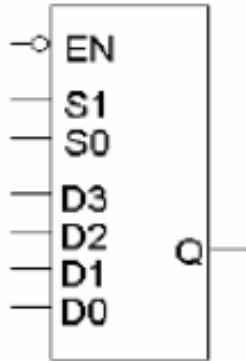
- Again we have a minimal sum of products expression, which leads to a minimal two-level circuit implementation.



$$Q = EnS1'S0'D0 + EnS1'S0D1 + EnS1S0'D2 + EnS1S0D3$$

A 4-to-1 multiplexer (Active Low)

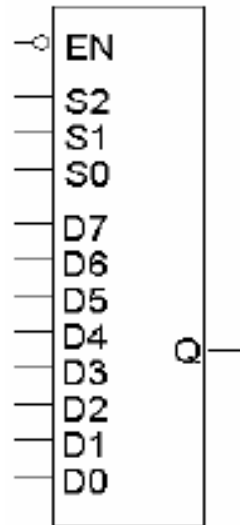
- LogicWorks multiplexers have **active-low** enable inputs, so the mux always outputs 1 when $EN' = 1$. This is denoted on the block symbol with a bubble in front of EN.



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

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.

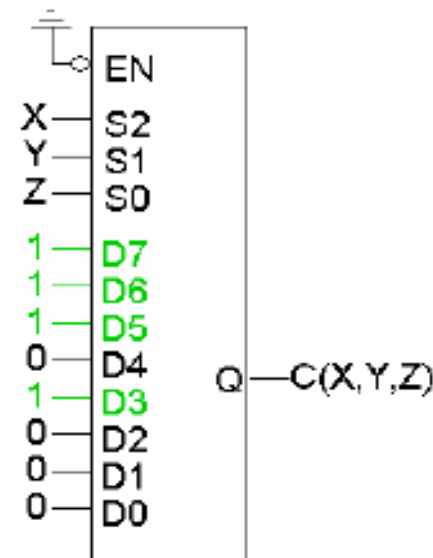
	X	Y	Z	C	S
	0	0	0	0	0
	0	0	1	0	1
	0	1	0	0	1
$0 + 1 + 1 = 10$ →	0	1	1	1	0
	1	0	0	0	1
	1	0	1	1	0
	1	1	0	1	0
$1 + 1 + 1 = 11$ →	1	1	1	1	1

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

Implementing functions with multiplexers

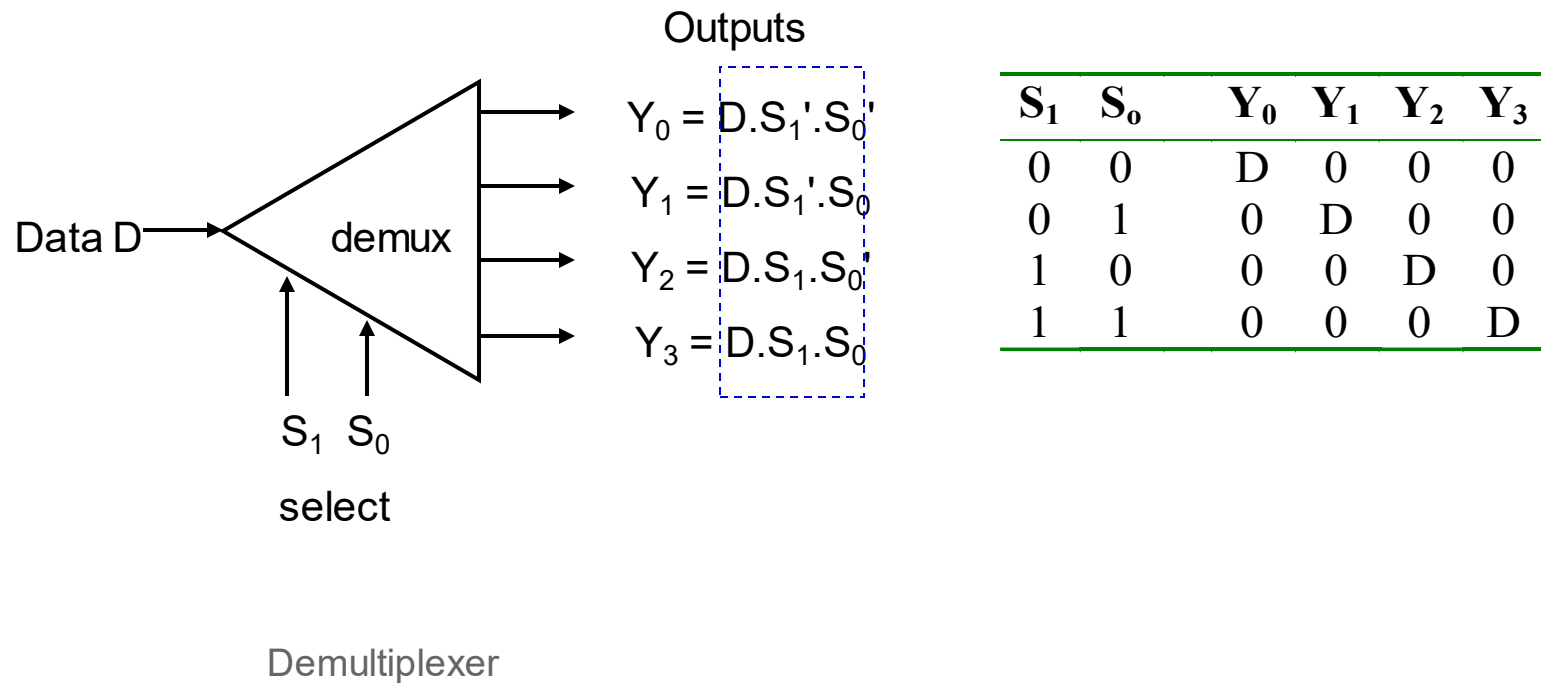
- We could implement a function of n variables with an n -to-1 multiplexer.
 - The mux select inputs correspond to the function's input variables, and are used to select one row of the truth table.
 - Each mux data input corresponds to one output from the truth table. We connect 1 to data input D_i for each function minterm m_i , and we connect 0 to the other data inputs.
- For example, here is the carry function, $C(X,Y,Z) = \Sigma m(3,5,6,7)$.

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:



Demultiplexer ...

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

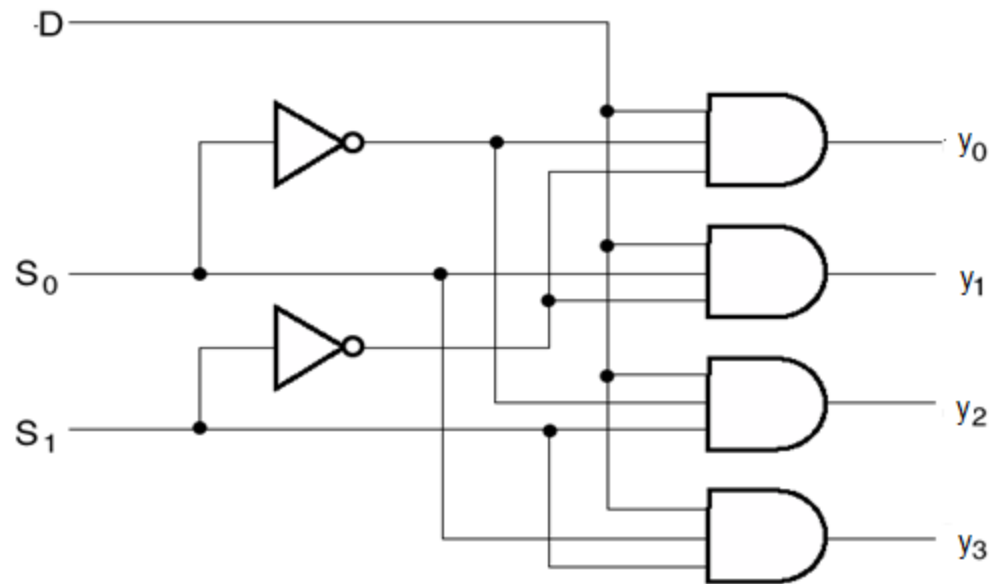
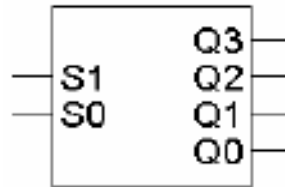


Fig. 3-24 1-to-4-Line Demultiplexer

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 Q_i 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:

$$Q0 = S1'S0'$$

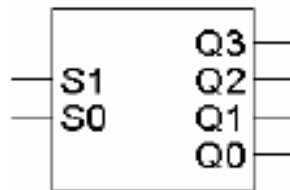
$$Q1 = S1'S0$$

$$Q2 = S1 S0'$$

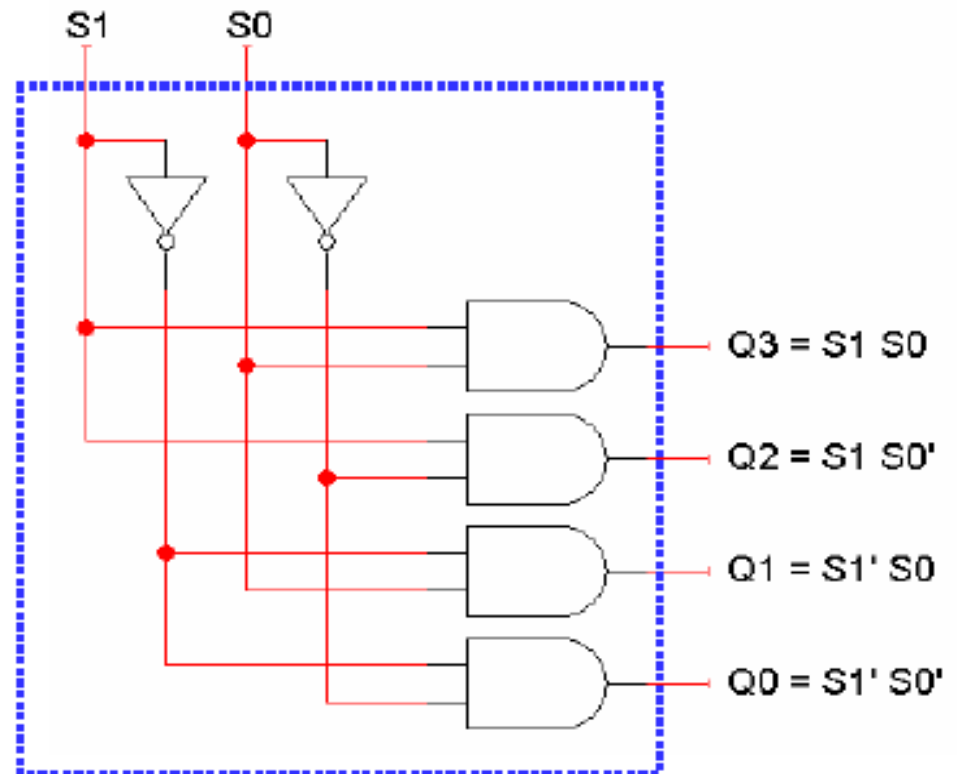
$$Q3 = S1 S0$$

Decoder circuit diagram

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

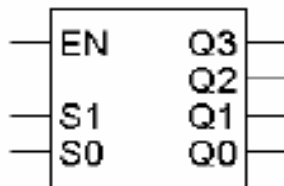


S1	S0	Q0	Q1	Q2	Q3
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1



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.



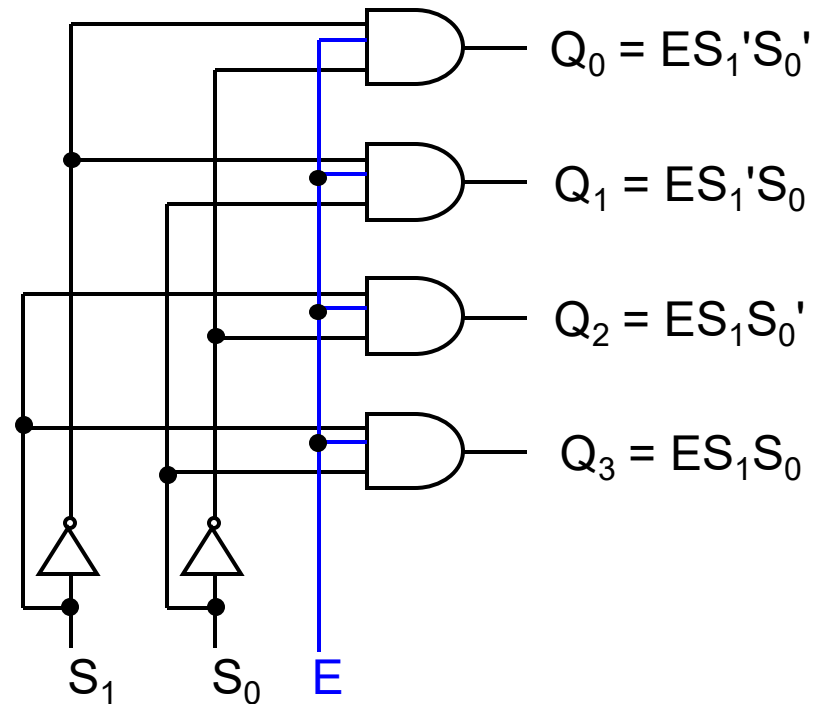
EN	S1	S0	Q0	Q1	Q2	Q3
0	x	x	0	0	0	0
1	0	0	1	0	0	0
1	0	1	0	1	0	0
1	1	0	0	0	1	0
1	1	1	0	0	0	1

Decoders with Enable ...

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

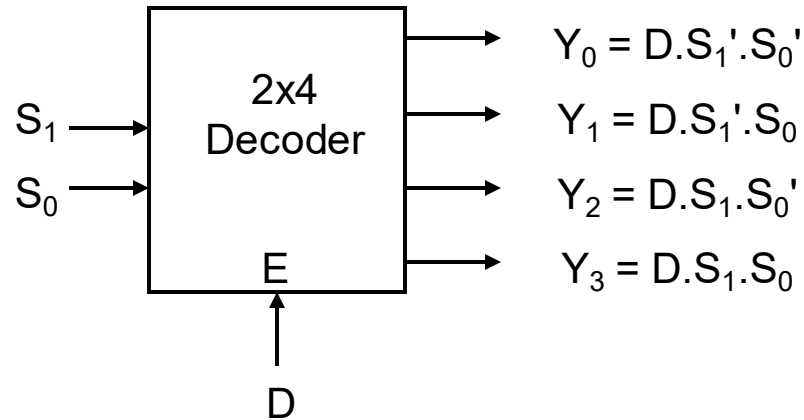
E	S ₁	S ₀	Q ₀	Q ₁	Q ₂	Q ₃
1	0	0	1	0	0	0
1	0	1	0	1	0	0
1	1	0	0	0	1	0
1	1	1	0	0	0	1
0	X	X	0	0	0	0

- Circuit:



Demultiplexer Vs Decoder ...

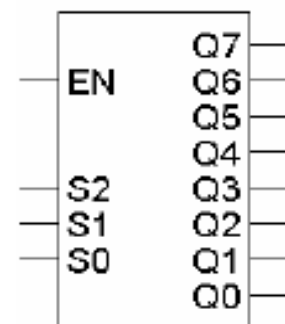
- The demultiplexer is actually identical to a decoder with enable, as illustrated below:



Exercise: Provide the truth table for above demultiplexer.

A 3-to-8 decoder

- Larger decoders are similar. Here is a 3-to-8 decoder.
 - There are three selection inputs $S_2S_1S_0$, which activate one of eight outputs, Q_0 - Q_7 .
 - 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

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

$$Q_1 = S_2' S_1' S_0$$

$$Q_2 = S_2' S_1 S_0'$$

$$Q_3 = S_2' S_1 S_0$$

$$Q_4 = S_2 S_1' S_0'$$

$$Q_5 = S_2 S_1' S_0$$

$$Q_6 = S_2 S_1 S_0'$$

$$Q_7 = S_2 S_1 S_0$$

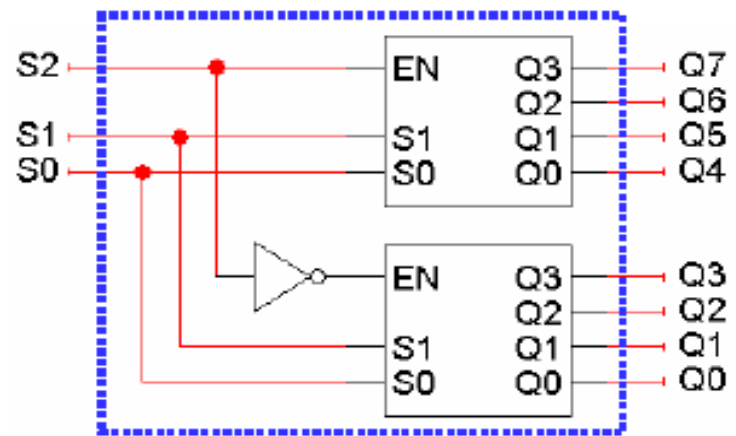
Building a 3-to-8 decoder

- 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 $S_2 = 0$, 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.

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 $S_2=0$, the bottom 2-to-4 decoder is enabled and generates a 1 for one of outputs Q_0 , Q_1 , Q_2 or Q_3 .
- When $S_2=1$, the top 2-to-4 decoder is enabled instead, and a 1 will be output for either Q_4 , Q_5 , Q_6 or Q_7 .



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

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

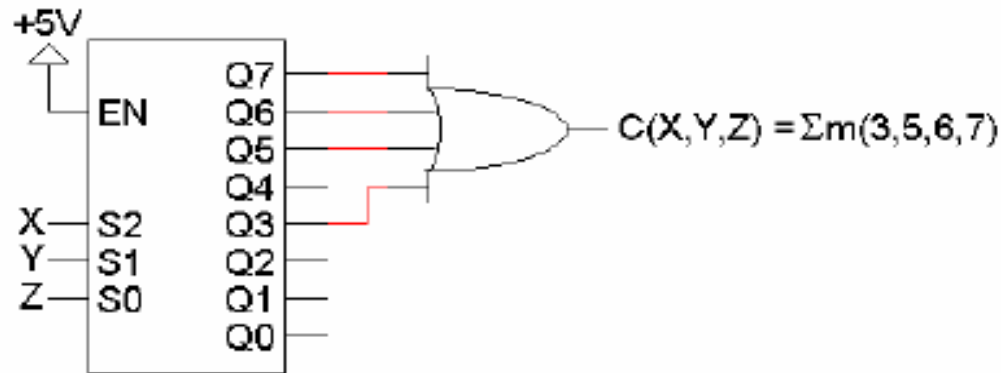
$$C(X,Y,Z) = \sum m(3,5,6,7)$$

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

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

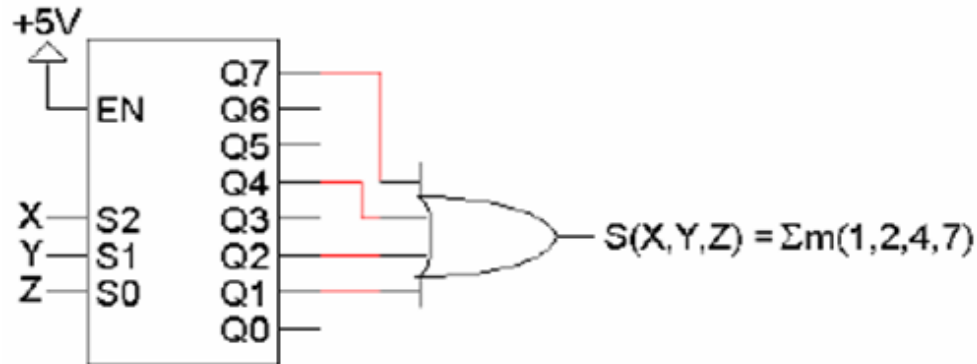
Implementing functions with decoders

- 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.

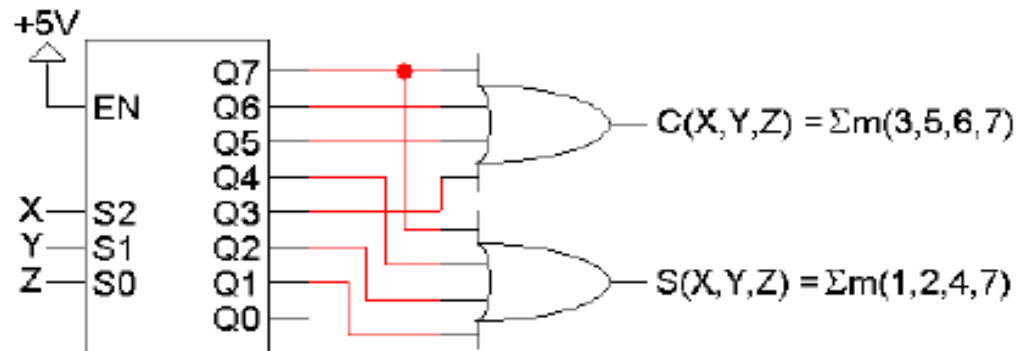
Decoder-based sum



- 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

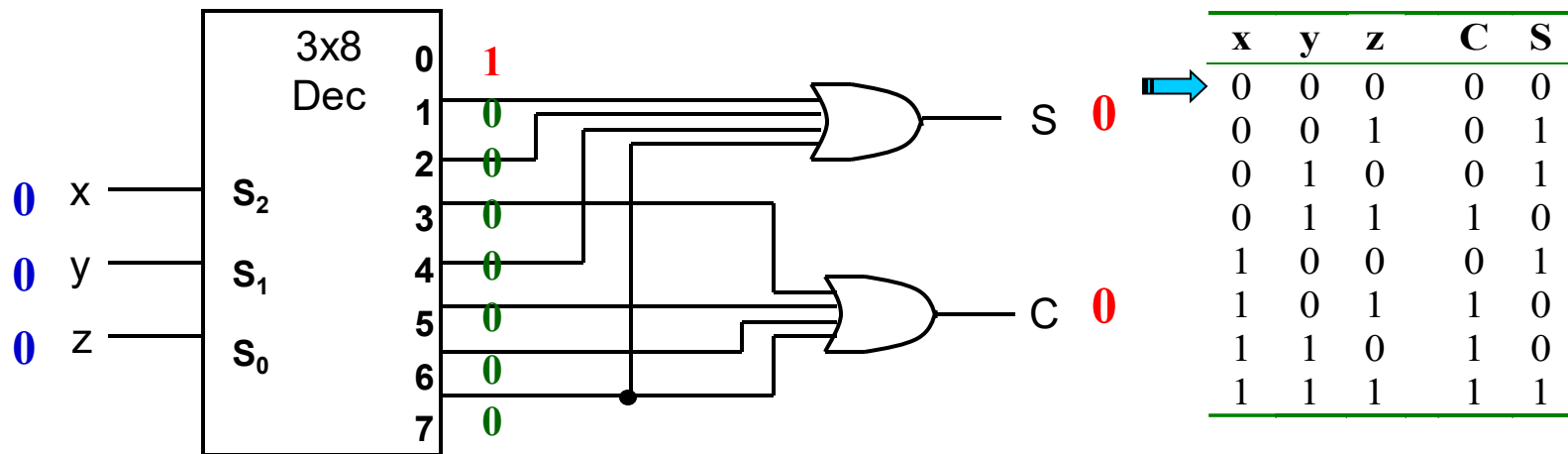
- Since the two functions C and S both have the same inputs, we could use just one decoder instead of two.



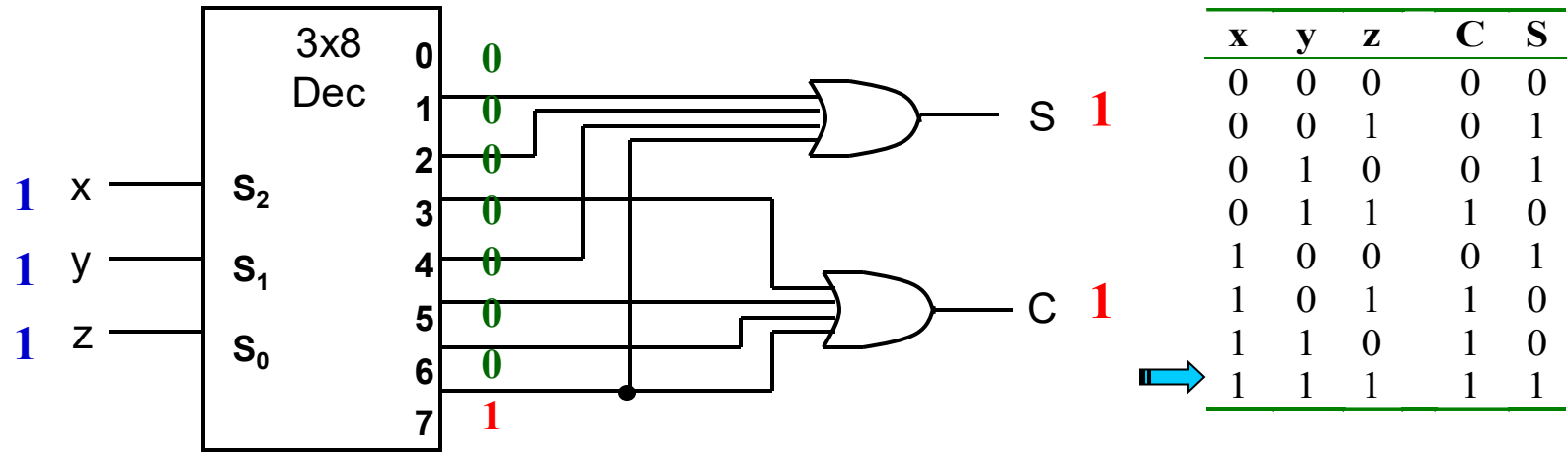
- Decoder output Q_0 is unused, while Q_7 is used multiple times. In general, you can always use circuit outputs as many or as few times as you need.

Decoders: Implementing Functions

Example: Full adder



Decoders: Implementing Functions ...



Encoder

- Encoder is the opposite of decoder
- **2^n inputs** or less and **n outputs**
- Example: **Decimal to BCD**
 - 10 inputs : $I_0, I_1, I_2, I_3, \dots, I_9$
 - 4 output lines

Encoder ...

Example: Designing an **Octal to binary** Encoder

Inputs: 8 ($D_7 \dots D_0$)

Outputs: 3 ($A_2 \dots A_0$)

Truth Table:

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

Encoder ...

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