UM EECS 270 F22
Introduction to Logic Design

8. Combinational Building Blocks

Combinational Blocks

- Goal is to create a “toolbox” of devices that are frequently required in logic design
  - Originally, each device would be on its own IC
  - Today, circuit can be made so small that many circuits can fit on a single chip
  - VLSI can *instantiate* adders, comparators, etc., as many times as needed within a larger circuit
  - We would like to identify other frequently used combinational circuits and find small circuits for them in advance

Equality Comparator

- Want a signal, EQ, that is 1 iff two 4-bit num… A and B, are equal. Under what logic condi… A and B equal?
  - Equal if $a_3 = b_3$ AND $a_2 = b_2$ AND a_1 … $_0 = b_0$
- Which gate performs an equality c… n?
  - XNOR!

A	B	A⊙B
0	0	1
0	1	0
1	0	0
1	1	1

$$EQ = (a_3 \odot b_3) \cdot (a \quad \odot b_1) \cdot (a_0 \odot b_0)$$

Already Seen This

How many gates are required to implement an *n*-bit "parallel" comparator?

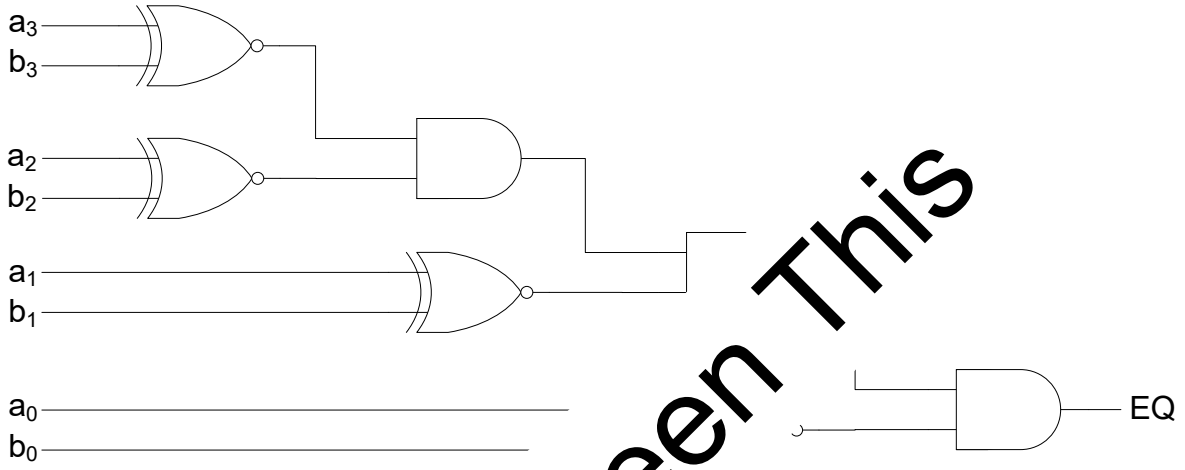
Assume $n = 2^k$

#gates = $2^k + 2^{k-1} + 2^{k-1} + \ldots 2^0$

$= 2^{k+1} - 1$

$= 2n - 1$

A "Series" Comparator Implementation



How many gat ired to implement an *n-* comparator?

- Comparing first two

- Comparing eac bit: 2 gates

- After comparin 2 bits, there are n-2 bits left to compare for a total of 2*(n-2) gat

-Total number of gates required: 2(n-2) + 3 = 2n – 1

- "Series" and "parallel" implementations require the same number of gates! Is there a reason to prefer one implementation over the other?

Magnitude Comparator

- Want to design a signal, AgrB, that is [illegible] A > B (A and B are 4-bit numbers). Under what lo[illegible]itions is A > B?
  - A > B if $a_3 = 1$ and $b_3 = 0$
  - A > B if $a_2 = 1$ and $b_2 = 0$ *ar*[illegible]
  - A > B if $a_1 = 1$ and $b_1 = 0$ [illegible] b_2 and $a_3 = b_3$
  - A > B if $a_0 = 1$ and b_0 [illegible] $a_1 = b_1$ and $a_2 = b_2$ and $a_3 = b_3$

Already Seen This

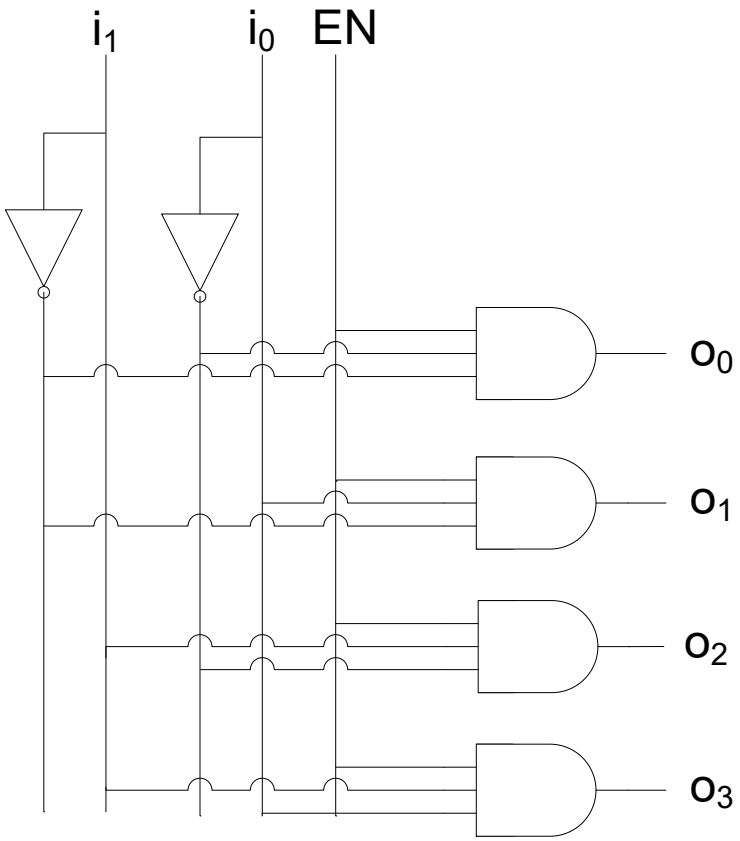
$$AgrF = [illegible]$$

$$[illegible]_3 \odot b_3) \cdot a_2 \cdot \overline{b_2} +$$

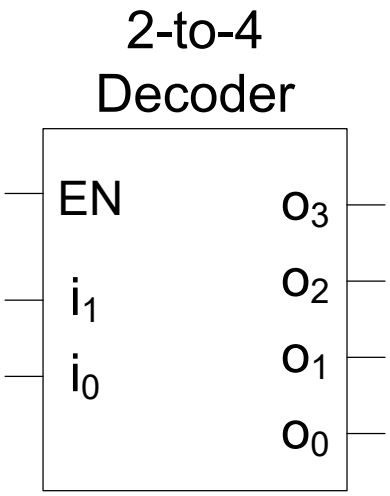
$$(a_3 \odot b_3) \cdot (a_2 \odot b_2) \cdot a_1 \cdot \overline{b_1} +$$

$$(a_3 \odot b_3) \cdot (a_2 \odot b_2) \cdot (a_1 \odot b_1) \cdot a_0 \cdot \overline{b_0}$$

Decoder Circuit

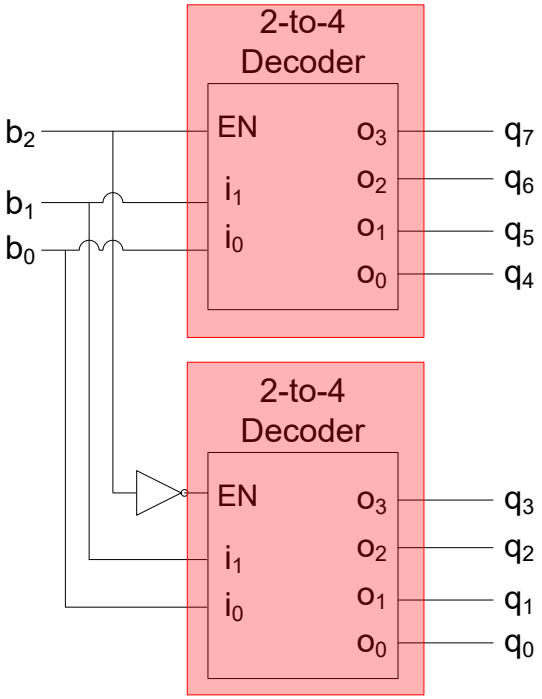


Symbol:



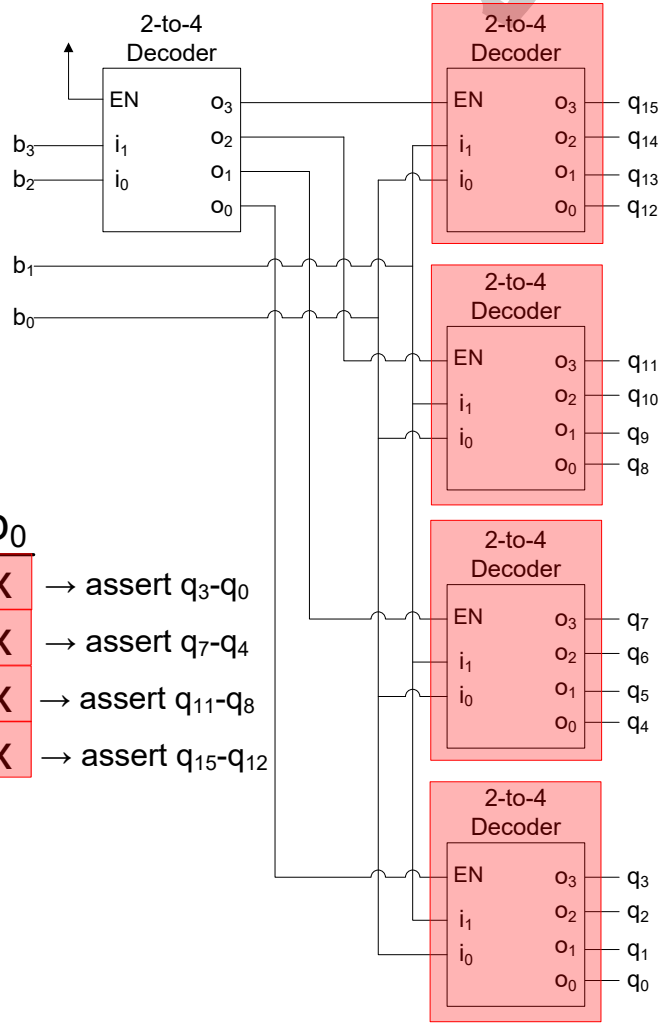
Cascading Decoders

Creating a 3-to-8 decoder from two 2-to-4 decoders:



b_2	b_1	b_0	
0	x	x	→ assert q_3-q_0
1	x	x	→ assert q_7-q_4

Creating a 4-to-16 decoder from five 2-to-4 decoders:

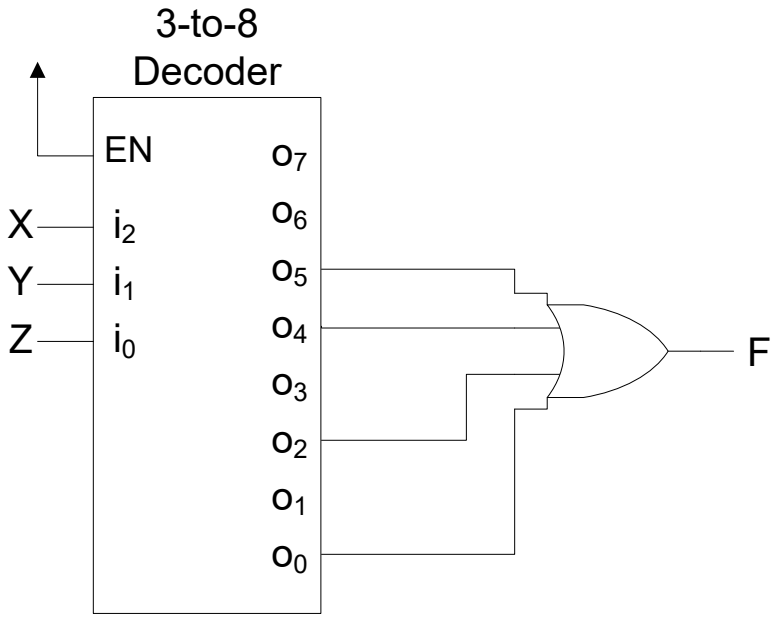


b_3	b_2	b_1	b_0	
0	0	x	x	→ assert q_3-q_0
0	1	x	x	→ assert q_7-q_4
1	0	x	x	→ assert q_{11}-q_8
1	1	x	x	→ assert q_{15}-q_{12}

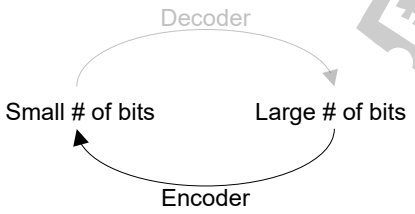
Implementing Functions with Decoders

- Any n-variable function can be implemented with a n-to-2^n decoder!
  - Recall that each decoder output is implemented as a minterm of the input variables
  - Recall that any function can be implemented as a sum of minterms
  - Therefore, any function can be implemented as a sum of decoder outputs!

$$F = \sum_{X,Y,Z}(0,2,4,5)$$



Encoders



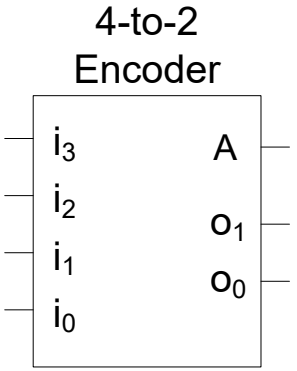
- Opposite of decoders: an encoder takes as input a code with a large number of bits and outputs a corresponding code with a small number of bits
- Most common encoder maps a 2^n one-hot code to an n-bit binary number, where the binary number represents the input number that is asserted
- What if all input signals are deasserted? Requires additional output, usually called A (active) that is asserted if *at least* one input is asserted

i_3	i_2	i_1	i_0	o_1	o_0	A
0	0	0	0	0	0	0
0	0	0	1	0	0	1
0	0	1	0	0	1	1
0	1	0	0	1	0	1
1	0	0	0	1	1	1
all others				d	d	d

$$o_1 = i_3 + i_2$$

$$o_0 = i_3 + i_1$$

$$A = i_3 + i_2 + i_1 + i_0$$

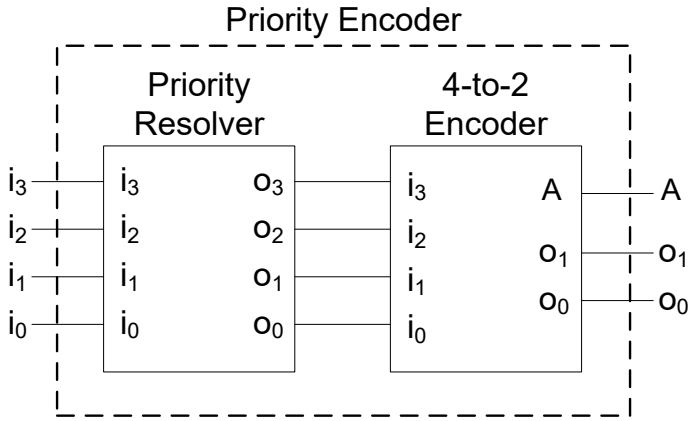


- What if we want to allow *multiple* inputs to be asserted simultaneously and output the asserted input number with the highest priority?
  - This task is performed by a **priority encoder**

Highest priority

i_3	i_2	i_1	i_0	o_1	o_0	A
0	0	0	0	0	0	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

- An easy way to implement: use our existing "toolbox" – specifically, make use of the encoder that we've already designed
- Need to add a circuit that will resolve priority: assert output signal corresponding to asserted input with highest priority



Priority Resolver

i_3	i_2	i_1	i_0	o_3	o_2	o_1	o_0
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	x	0	0	1	0
0	1	x	x	0	1	0	0
1	x	x	x	1	0	0	0

$$o_3 = i_3$$

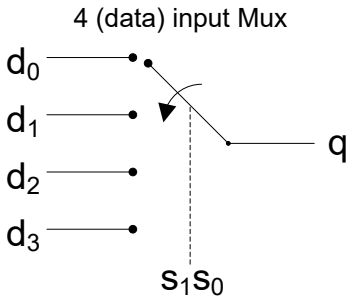
$$o_2 = i_2 \cdot \overline{i_3}$$

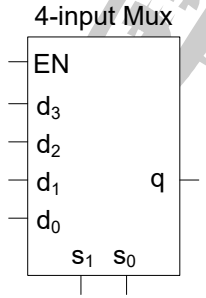
$$o_1 = i_1 \cdot \overline{i_2} \cdot \overline{i_3}$$

$$o_0 = i_0 \cdot \overline{i_1} \cdot \overline{i_2} \cdot \overline{i_3}$$

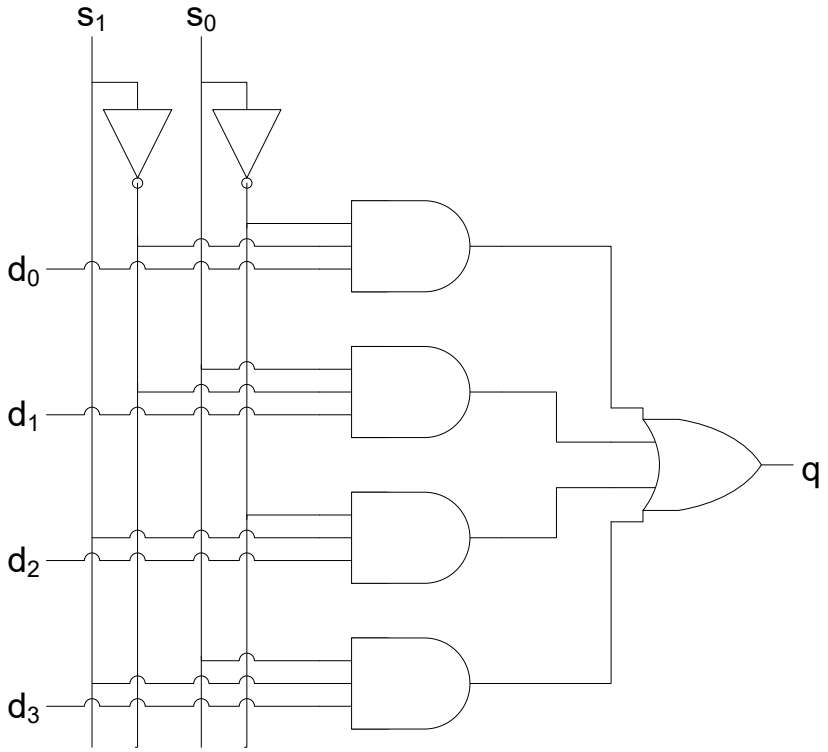
Multiplexer (Mux)

- A **mux** is a digital switch
- The output copies one of *n* data inputs, depending on the value of the *select inputs*
- Implementation:





$$q = s_1' s_0' d_0 + s' s_0 d_1 + s_1 s_0' d_2 + s_1 s_0 d_3$$

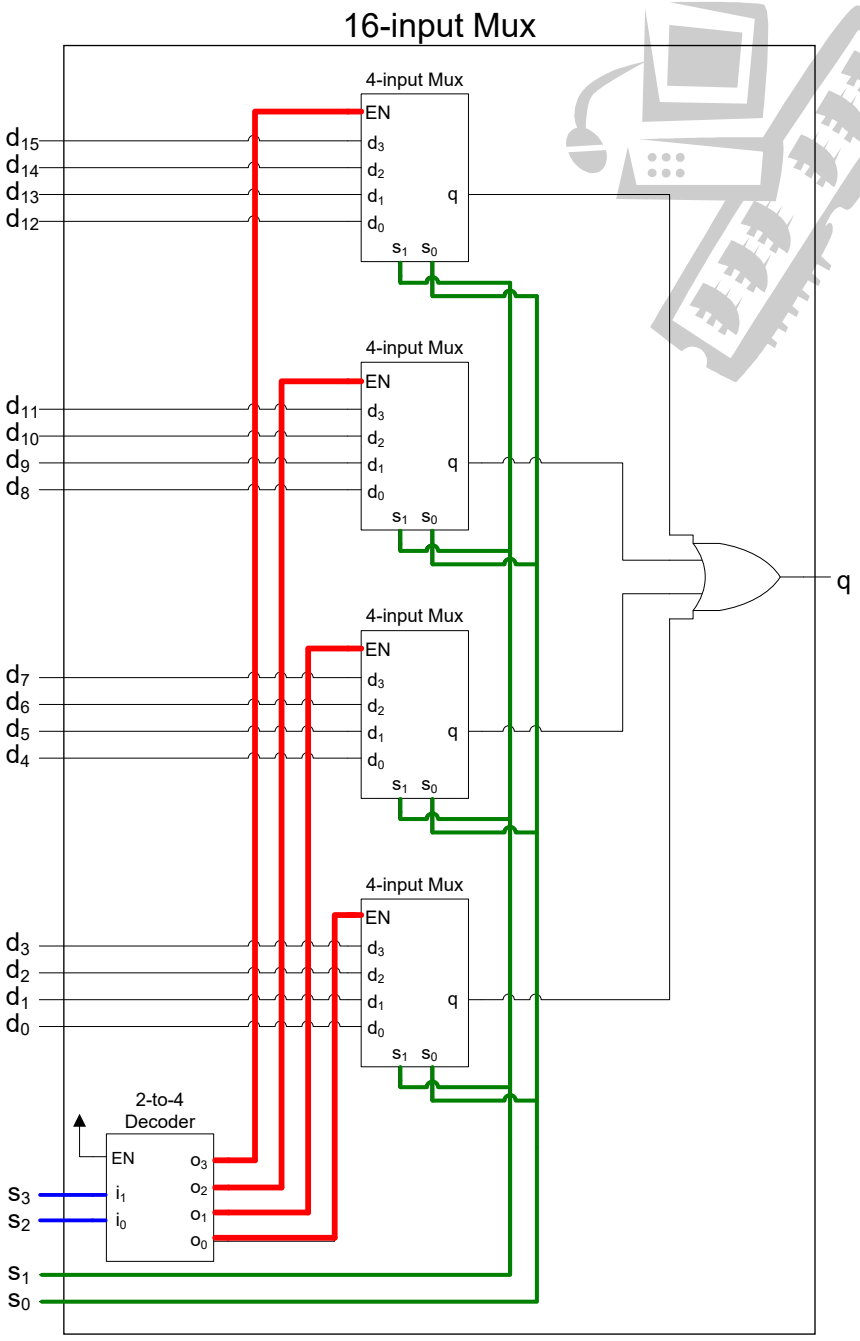


Product terms are mutually exclusive!

Building large multiplexers from smaller multiplexers (and other blocks)

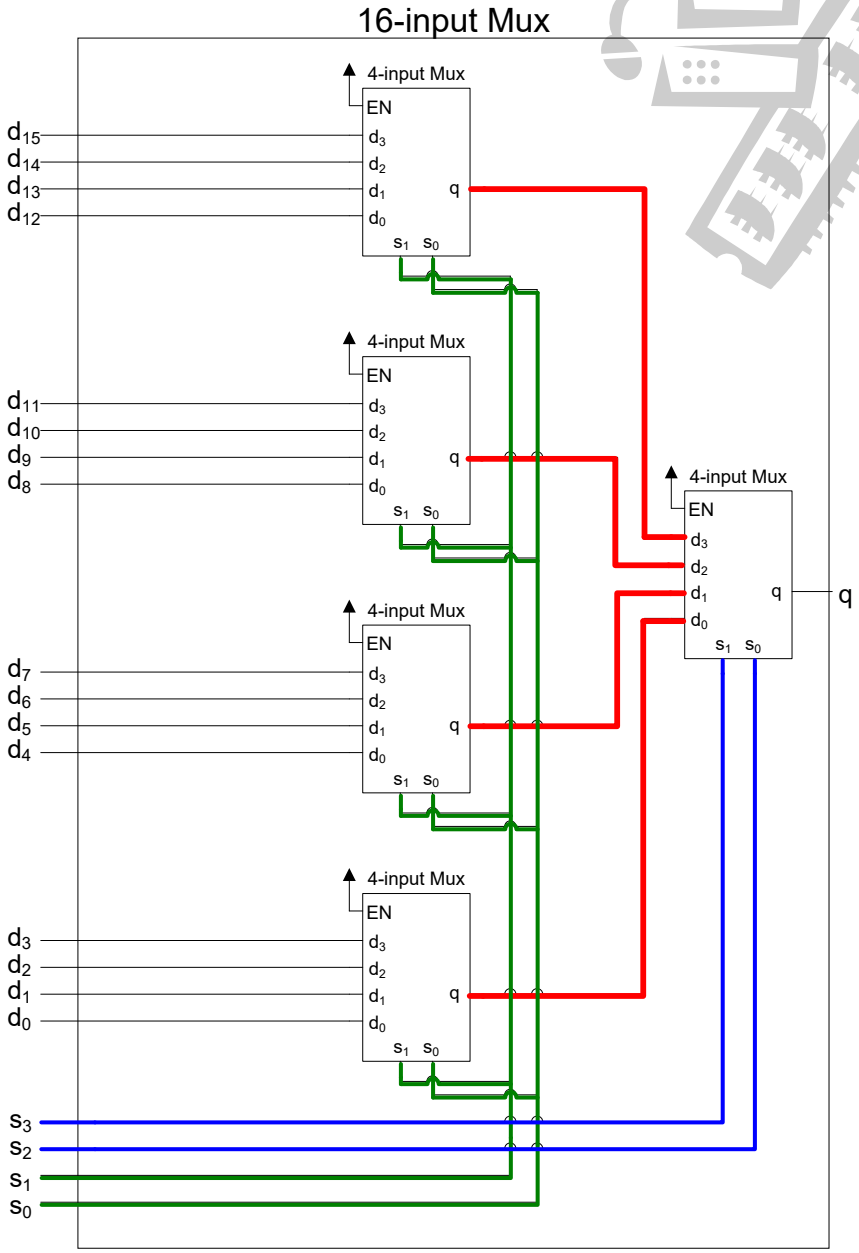
…using four 4-input muxes and 1 2-to-4 decoder

S_3	S_2	S_1	S_0	
0	0	x	x	→ select d_3-d_0
0	1	x	x	→ select d_7-d_4
1	0	x	x	→ select d_{11}-d_8
1	1	x	x	→ select d_{15}-d_{12}



…using five 4-input muxes

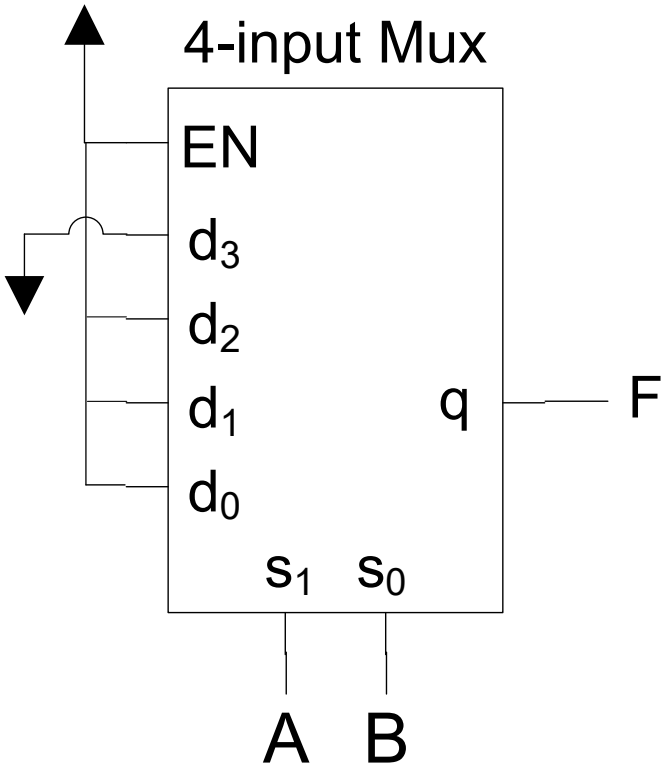
S_3	S_2	S_1	S_0	
0	0	x	x	→ select d_3-d_0
0	1	x	x	→ select d_7-d_4
1	0	x	x	→ select d_{11}-d_8
1	1	x	x	→ select d_{15}-d_{12}



Implementing Functions With a Multiplexer

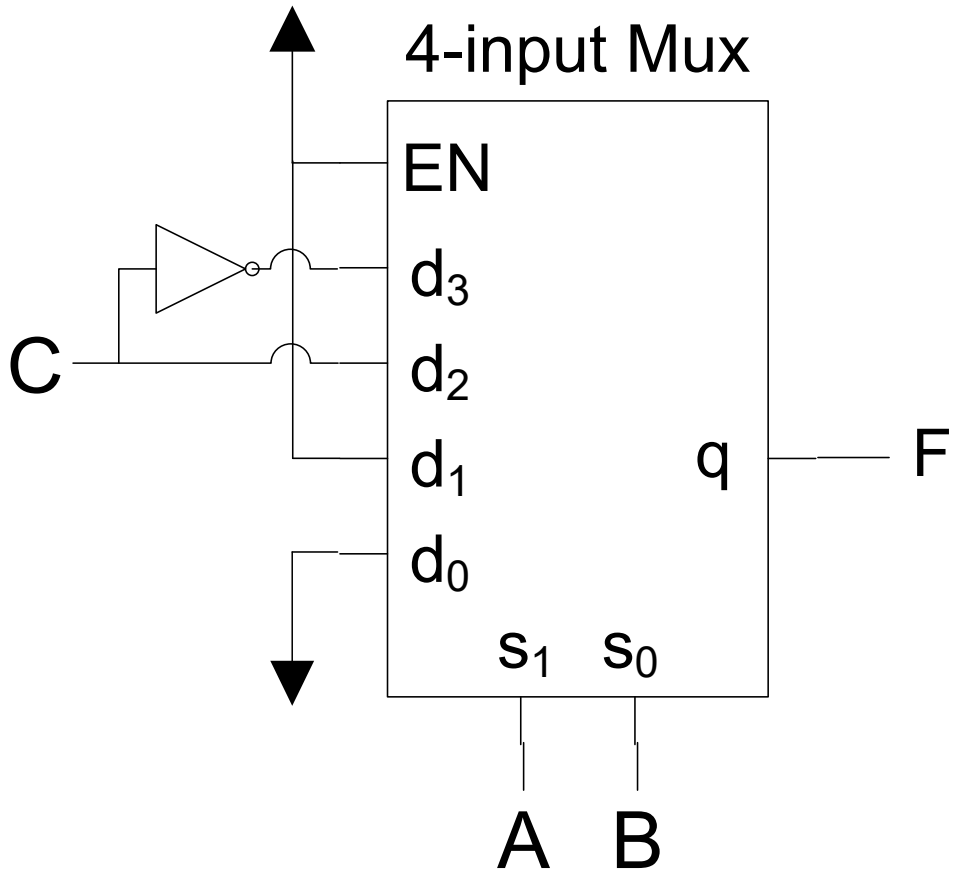
- We can implement an *n*-variable function using an *n*-select (2^n-input) mux

A	B	F
0	0	1
0	1	1
1	0	1
1	1	0



- We can do better: we can implement an *n*-input function with a 2^{n-1}-input mux!

A	B	C	F	
0	0	0	0	d_0
0	0	1	0	
0	1	0	1	d_1
0	1	1	1	
1	0	0	0	d_2
1	0	1	1	
1	1	0	1	d_3
1	1	1	0	

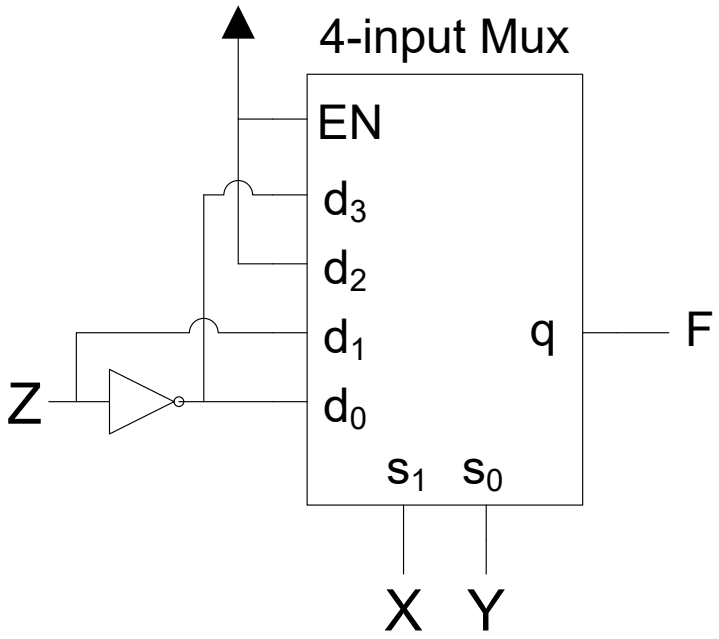


In Class Exercise

Implement the following 3-input function using a single 4-input mux:

$$F = \sum_{X,Y,Z}(0,3,4,5,6)$$

X	Y	Z	F
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0



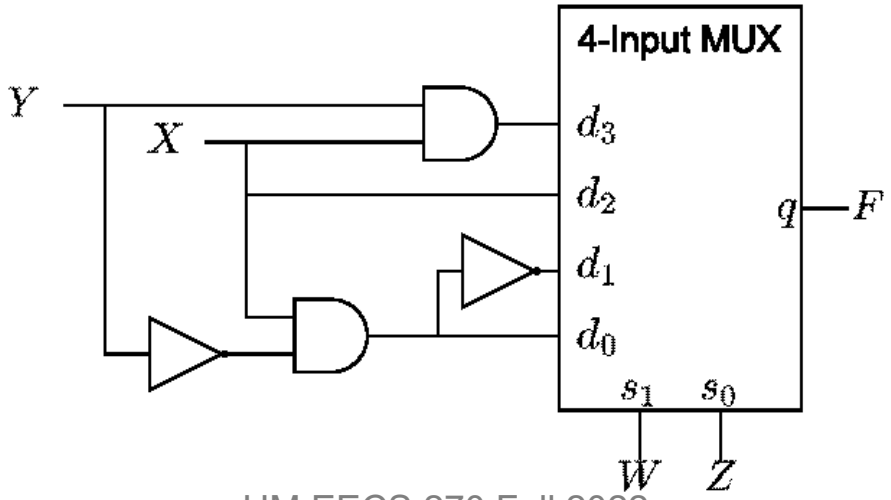
Function Implementation with MUX *is Application of* Boole/Shannon Expansion Theorem

$$F(W,X,Y,Z) = W'X'Z + XY'Z' + XYZ + WXY$$

$$= W'Z'(?) + W'Z(?) + WZ'(?) + WZ(?)$$

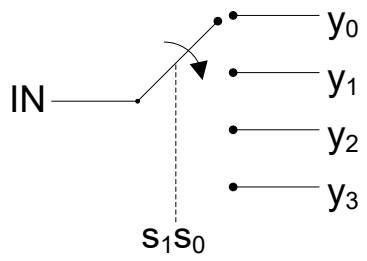
$$= W'Z'(XY') + W'Z(X' + XY) + WZ'(XY' + XY) + WZ(XY)$$

$$= W'Z'(XY') + W'Z(X' + Y) + WZ'(X) + WZ(XY)$$



Demultiplexer (demux)

- A **demux** connects an input signal to one of several output signals, depending on the value of the select signals

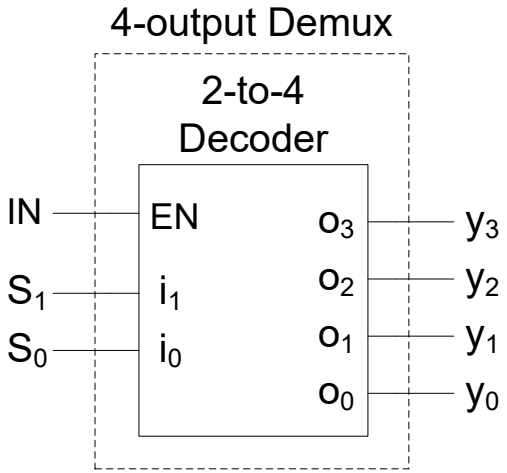


s_1	s_0	y_0	y_1	y_2	y_3
0	0	IN	0	0	0
0	1	0	IN	0	0
1	0	0	0	IN	0
1	1	0	0	0	IN

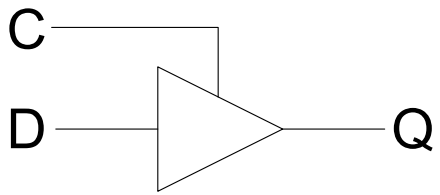
2-to-4 Decoder

EN	i_1	i_0	o_0	o_1	o_2	o_3
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

- How to implement?
  - TT should look very familiar…
  - Use a decoder!

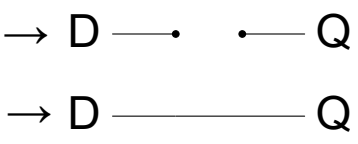


Tri-State Gates

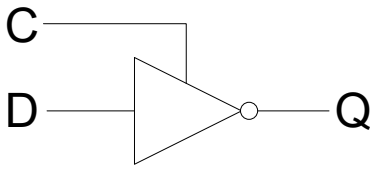


Tri-state Buffer

C	Q
0	High Impedance
1	D

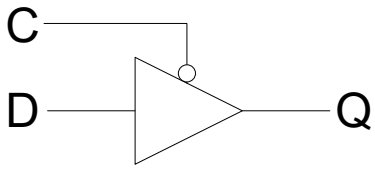


Other tri-state devices:



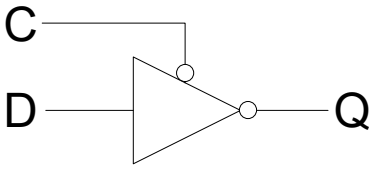
Tri-state Inverter

C	Q
0	High Impedance
1	$\overline{D}$



Active low
Tri-state Buffer

C	Q
0	D
1	High Impedance

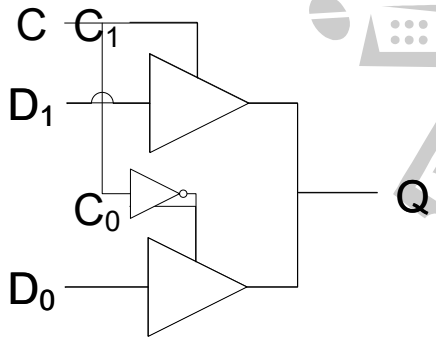


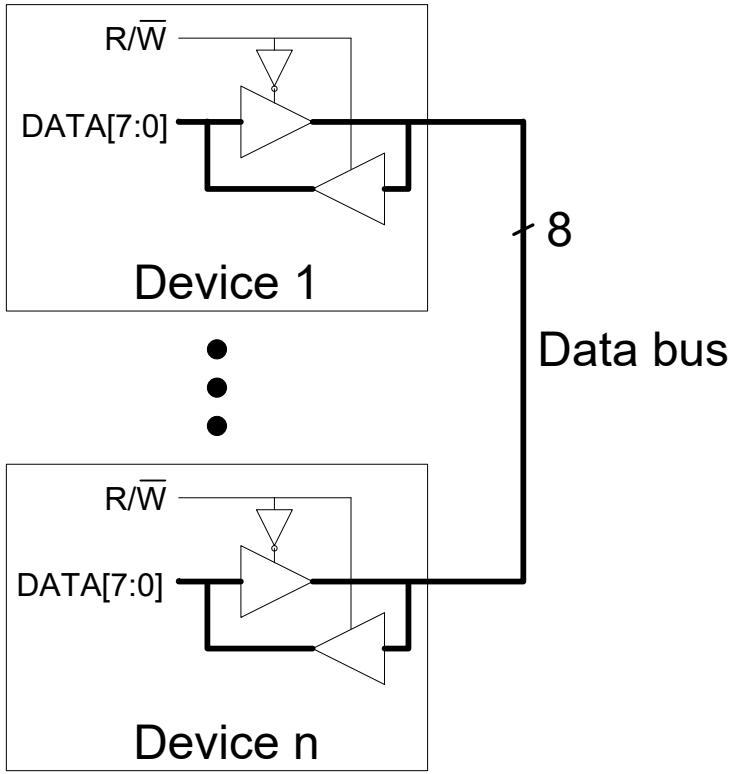
Active low
Tri-state Inverter

C	Q
0	$\overline{D}$
1	High Impedance

- Is it ok to connect the outputs of tri-state devices?
  - $C_1 = C_0 = 1$ and $D_1 \neq D_0 \rightarrow$ bad!
  - $C_1 = C_0 = 0 \rightarrow Q = ?$
  - $D_1 = D_0 \rightarrow$ OK!
- Typically, tri-state control signals are **one-hot**
  - Results in exactly one device connected to the output at a time
- Multiple devices can communicate over a single shared wire
  - Only one device is allowed to drive the wire at a time
  - A functional group of such wires is called a **bus**.

This looks familiar…





Tying it all together…