01001110010000110101001101010101

More on Decoders and Muxes

4e435355

1,313,035,093

ECGR2181

Lecture Notes 2A

1.525×2^{29}

Making Larger Components

- Consider destination then use component given to work toward solution.

- Think about the functionality of the destination component

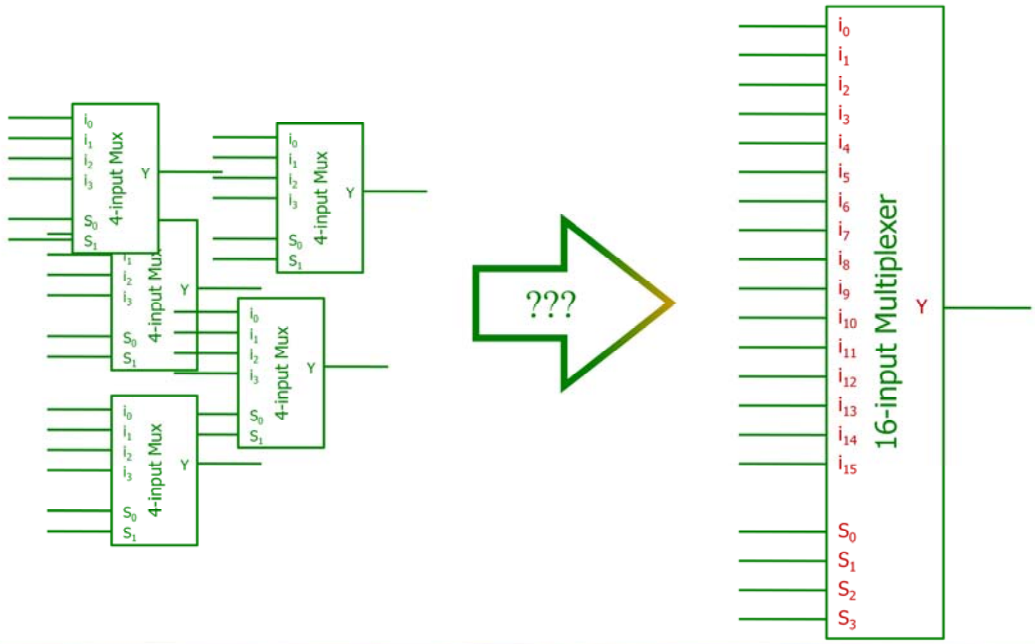
- Think about what the given parts can do

- Then, bring it all together

- Finally, once design is done make a couple of test cases to see if the design is valid. If not, repeat from start.

Making Large Muxes

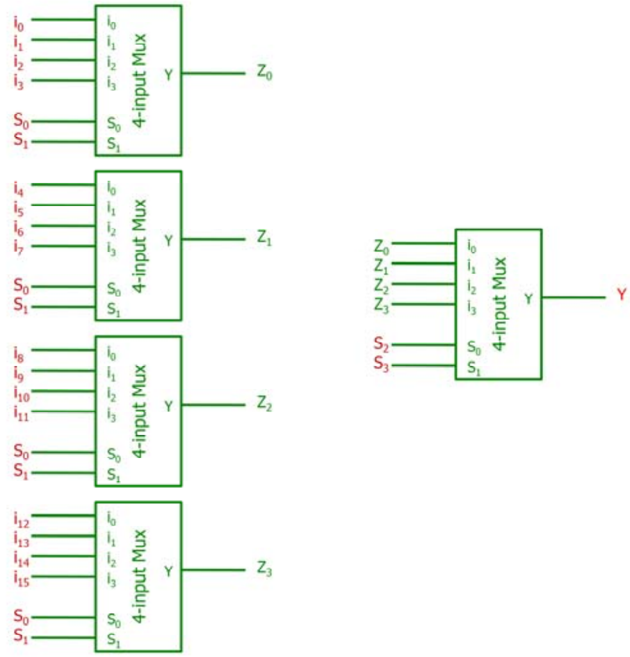
- i.e., construct a 16-input mux from any number of 4-input muxes



Example Continued

T.T. Goal:

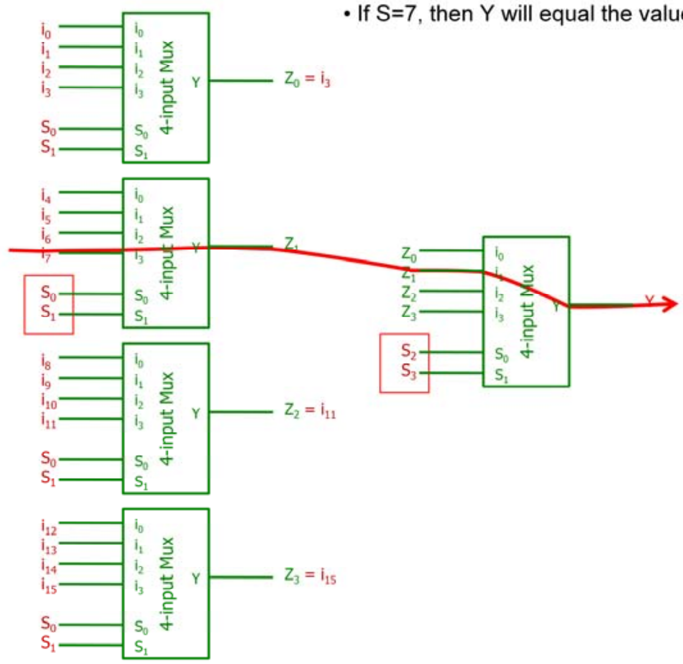
S_3	S_2	S_1	S_0	Y	$Z_?$
0	0	0	0	i_0	Z_0
0	0	0	1	i_1	Z_0
0	0	1	0	i_2	Z_0
0	0	1	1	i_3	Z_0
0	1	0	0	i_4	Z_1
0	1	0	1	i_5	Z_1
0	1	1	0	i_6	Z_1
0	1	1	1	i_7	Z_1
1	0	0	0	i_8	Z_2
1	0	0	1	i_9	Z_2
1	0	1	0	i_{10}	Z_2
1	0	1	1	i_{11}	Z_2
1	1	0	0	i_{12}	Z_3
1	1	0	1	i_{13}	Z_3
1	1	1	0	i_{14}	Z_3
1	1	1	1	i_{15}	Z_3



Signal Flow Illustrated

S_3	S_2	S_1	S_0	Y
0	0	0	0	i_0
0	0	0	1	i_1
0	0	1	0	i_2
0	0	1	1	i_3
0	1	0	0	i_4
0	1	0	1	i_5
0	1	1	0	i_6
0	1	1	1	i_7
1	0	0	0	i_8
1	0	0	1	i_9
1	0	1	0	i_{10}
1	0	1	1	i_{11}
1	1	0	0	i_{12}
1	1	0	1	i_{13}
1	1	1	0	i_{14}
1	1	1	1	i_{15}

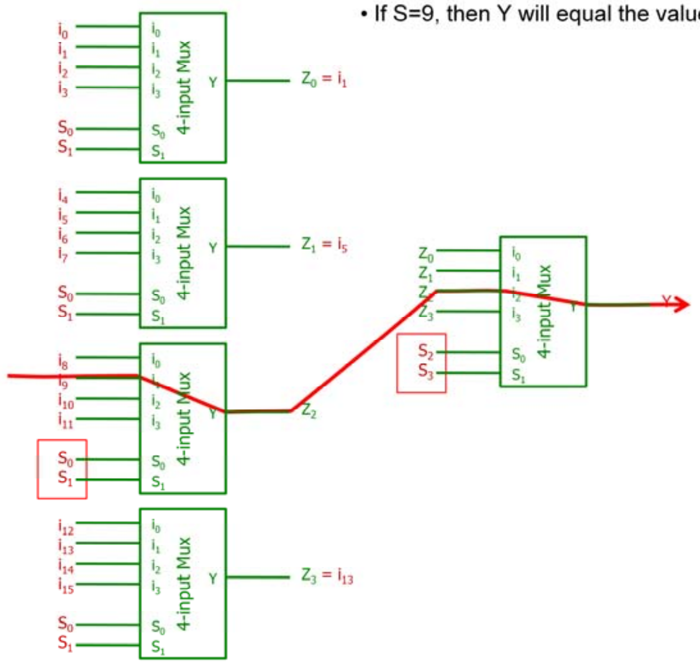
- If S=7, then Y will equal the value on i_7



Signal Flow Illustrated, again

S_3	S_2	S_1	S_0	Y
0	0	0	0	i_0
0	0	0	1	i_1
0	0	1	0	i_2
0	0	1	1	i_3
0	1	0	0	i_4
0	1	0	1	i_5
0	1	1	0	i_6
0	1	1	1	i_7
1	0	0	0	i_8
1	0	0	1	i_9
1	0	1	0	i_{10}
1	0	1	1	i_{11}
1	1	0	0	i_{12}
1	1	0	1	i_{13}
1	1	1	0	i_{14}
1	1	1	1	i_{15}

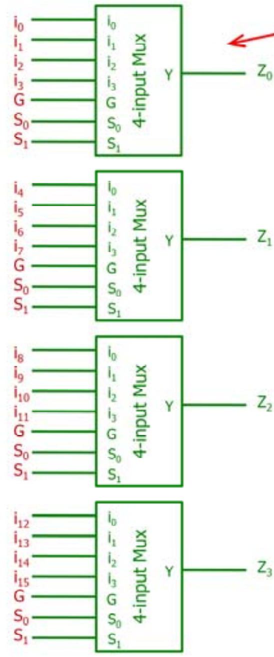
- If S=9, then Y will equal the value on i_9



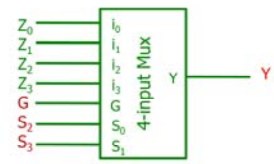
Enable Lines

T.T. Goal:

G	S_3	S_2	S_1	S_0	Y
0	x	x	x	x	0
1	0	0	0	0	i_0
1	0	0	0	1	i_1
1	0	0	1	0	i_2
1	0	0	1	1	i_3
1	0	1	0	0	i_4
1	0	1	0	1	i_5
1	0	1	1	0	i_6
1	0	1	1	1	i_7
1	1	0	0	0	i_8
1	1	0	0	1	i_9
1	1	0	1	0	i_{10}
1	1	0	1	1	i_{11}
1	1	1	0	0	i_{12}
1	1	1	0	1	i_{13}
1	1	1	1	0	i_{14}
1	1	1	1	1	i_{15}



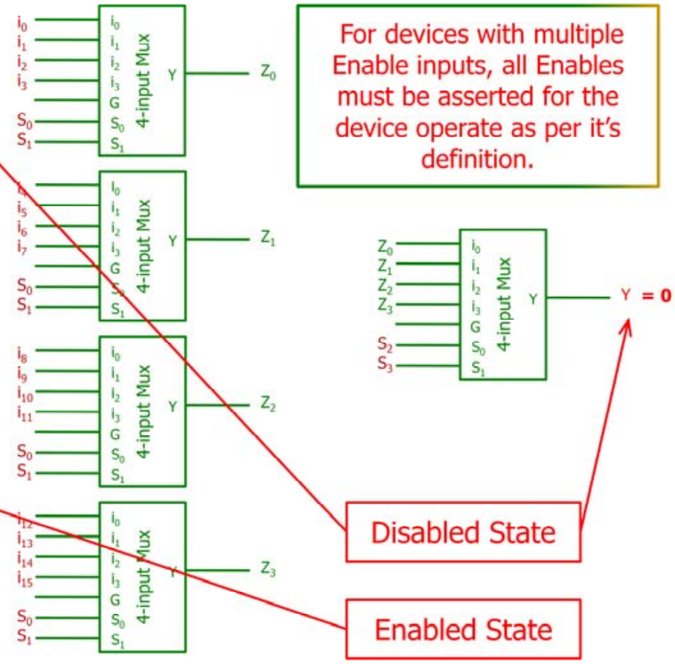
G	S_1	S_0	Y
0	x	x	0
1	0	0	i_0
1	0	1	i_1
1	1	0	i_2
1	1	1	i_3



Enable Lines, Continued

16-input Mux, w/ 2 enable inputs {1 active-high (G_0) & 1 active-low (G_1)}

G_1	G_0	S_3	S_2	S_1	S_0	Y
x	0	x	x	x	x	0
1	x	x	x	x	x	0
0	1	0	0	0	0	i_0
0	1	0	0	0	1	i_1
0	1	0	0	1	0	i_2
0	1	0	0	1	1	i_3
0	1	0	1	0	0	i_4
0	1	0	1	0	1	i_5
0	1	0	1	1	0	i_6
0	1	0	1	1	1	i_7
0	1	1	0	0	0	i_8
0	1	1	0	0	1	i_9
0	1	1	0	1	0	i_{10}
0	1	1	0	1	1	i_{11}
0	1	1	1	0	0	i_{12}
0	1	1	1	0	1	i_{13}
0	1	1	1	1	0	i_{14}
0	1	1	1	1	1	i_{15}

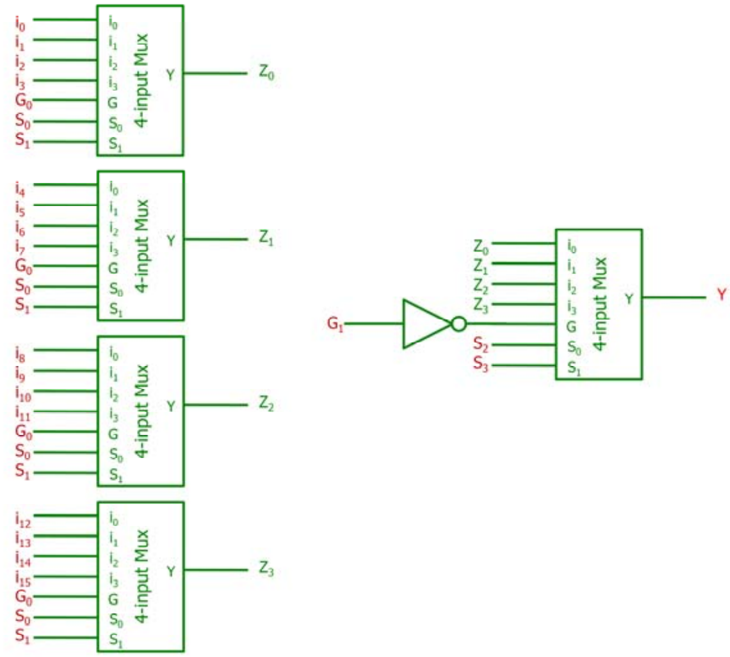


For devices with multiple Enable inputs, all Enables must be asserted for the device operate as per it's definition.

Enable Lines, Continued

16-input Mux, w/ 2 enable inputs {1 active-high (G_0) & 1 active-low (G_1)}

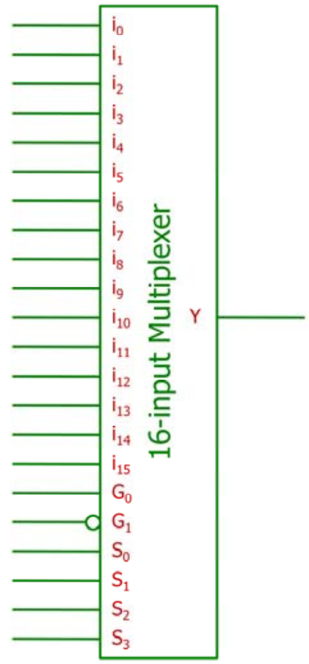
G_1	G_0	S_3	S_2	S_1	S_0	Y
x	0	x	x	x	x	0
1	x	x	x	x	x	0
0	1	0	0	0	0	i_0
0	1	0	0	0	1	i_1
0	1	0	0	1	0	i_2
0	1	0	0	1	1	i_3
0	1	0	1	0	0	i_4
0	1	0	1	0	1	i_5
0	1	0	1	1	0	i_6
0	1	0	1	1	1	i_7
0	1	1	0	0	0	i_8
0	1	1	0	0	1	i_9
0	1	1	0	1	0	i_{10}
0	1	1	0	1	1	i_{11}
0	1	1	1	0	0	i_{12}
0	1	1	1	0	1	i_{13}
0	1	1	1	1	0	i_{14}
0	1	1	1	1	1	i_{15}



The New Mux

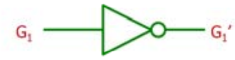
Here is a the TT & symbol for our shinny new 16-input Mux, with 2 enable lines

G_1	G_0	S_3	S_2	S_1	S_0	Y
x	0	x	x	x	x	0
1	x	x	x	x	x	0
0	1	0	0	0	0	i_0
0	1	0	0	0	1	i_1
0	1	0	0	1	0	i_2
0	1	0	0	1	1	i_3
0	1	0	1	0	0	i_4
0	1	0	1	0	1	i_5
0	1	0	1	1	0	i_6
0	1	0	1	1	1	i_7
0	1	1	0	0	0	i_8
0	1	1	0	0	1	i_9
0	1	1	0	1	0	i_{10}
0	1	1	0	1	1	i_{11}
0	1	1	1	0	0	i_{12}
0	1	1	1	0	1	i_{13}
0	1	1	1	1	0	i_{14}
0	1	1	1	1	1	i_{15}

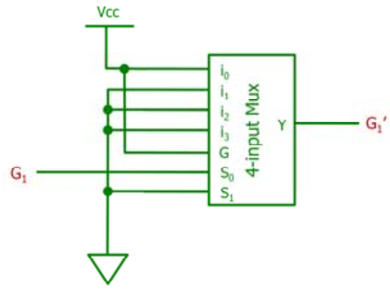


Mux Implementation of Combinational Logic

- Make an inverter out of a 4-input Mux



G	S_1	S_0	Y
0	x	x	0
1	0	0	i_0
1	0	1	i_1
1	1	0	i_2
1	1	1	i_3



- Start with the Mux, keep an eye on the TT.
- Configure the inputs such that …
  - when G1 = '0' the output will equal '1'
  - and, when G1 = '1' the output will equal '0'
- Enable the Mux always.
- Never leave unused inputs unconnected.

Mux Implementation of Boolean Expression

- More complex design … F(a,b,c)
  - For a 8-input mux the output can be described as

$$Y = i_0(S_2'S_1'S_0') + i_1(S_2'S_1'S_0) + i_2(S_2'S_1S_0') + i_3(S_2'S_1S_0) + \ldots$$

$$\ldots + i_4(S_2S_1'S_0') + i_5(S_2S_1'S_0) + i_6(S_2S_1S_0') + i_7(S_2S_1S_0)$$

1. Connect the inputs of the function (a,b,c) to the select lines of the mux … with a connected to S_3

$$Y = i_0(a'b'c') + i_1(a'b'c) + i_2(a'bc') + i_3(a'bc) + i_4(ab'c') + i_5(ab'c) + i_6(abc') + i_7(abc)$$

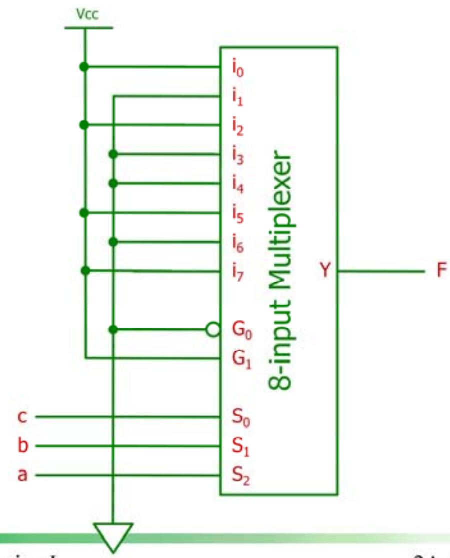
Continued →

Mux Implementation of Boolean Expression (cont.)

2. Connect the inputs to the mux ($i_7 - i_0$) to one or zero depending upon the value in the truth table

- For $F(a,b,c) = \Sigma m(0,2,5,7)$ ... $i_0=1$; $i_1=0$; $i_2=1$; $i_3=0$; $i_4=0$; $i_5=1$; $i_6=0$; $i_7=1$
- F(a,b,c) = 1(a'b'c') + 0(a'b'c) + 1(a'bc') + 0(a'bc) + 0(ab'c') + 1(ab'c) + 0(abc') + 1(abc)
- F(a,b,c) = a'b'c' + a'bc' + ab'c + abc
- $F(a,b,c) = m_0 + m_2 + m_5 + m_7$

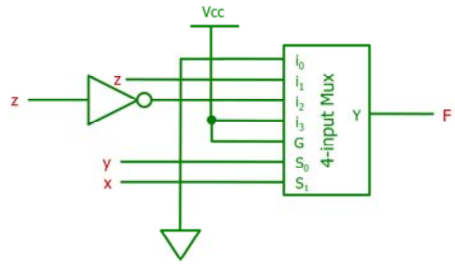
- Begin by connecting the system inputs (a,b,c) to the select lines as described above
- Connect the minterms where the function equals '1' to Vcc
- Connect the remaining mux inputs to gnd
- Connect the enable lines to the appropriate Vcc or gnd to always enable the mux
- Connect the output of the mux to the output of the system (F)



More Mux Implementation of Comb. Logic

- Implement $F(x,y,z) = \Sigma m(3,4,6,7)$ using a 4-input mux

x	y	z	F
0	0	0 →	0
0	0	1 →	0
0	1	0 →	0
0	1	1 →	1
1	0	0 →	1
1	0	1 →	0
1	1	0 →	1
1	1	1 →	1



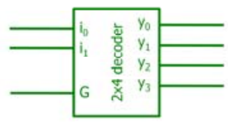
- Connect the most significant two inputs to the select lines
  - Pairs of minterms (where x & y remain constant) are then considered
- For xy = 00, the output F is independent of z … i_0 should be connected to gnd.
- For xy = 01, the output F is dependent on z … when z = 0, F = 0 and when z = 1 F = 1; thus $i_1 = z$
- For xy = 10, the output F is dependent on z … when z = 0, F = 1 and when z = 1 F = 0; thus $i_2 = z'$
- For xy = 11, the output F is independent of z … F = 1 for both cases of z … i_3 should be connected to Vcc
- Enable mux and connect the output of the mux to F

More on Decoders

- Recap:
  - When enabled: the input combination value (**i**) is the subscript of the output that is asserted. Otherwise, the output is zero. Where **i** is i_1 & i_0 concatenated.
  - When not enabled all outputs are zero.

- Making larger decoders:
  - A decoder is used to select the appropriate output decoder

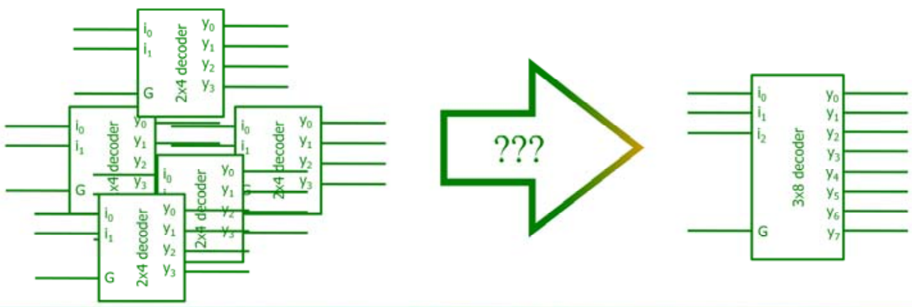
G	i_1	i_0	y_3	y_2	y_1	y_0
0	x	x	0	0	0	0
1	0	0	0	0	0	**1**
1	0	1	0	0	**1**	0
1	1	0	0	**1**	0	0
1	1	1	**1**	0	0	0



Expanding Decoders

3x8 Decoder using only 2x4 decoders

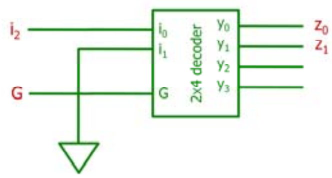
G	i_2	i_1	i_0	y_7	y_6	y_5	y_4	y_3	y_2	y_1	y_0
0	x	x	x	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	**1**
1	0	0	1	0	0	0	0	0	0	**1**	0
1	0	1	0	0	0	0	0	0	**1**	0	0
1	0	1	1	0	0	0	0	**1**	0	0	0
1	1	0	0	0	0	0	**1**	0	0	0	0
1	1	0	1	0	0	**1**	0	0	0	0	0
1	1	1	0	0	**1**	0	0	0	0	0	0
1	1	1	1	**1**	0	0	0	0	0	0	0

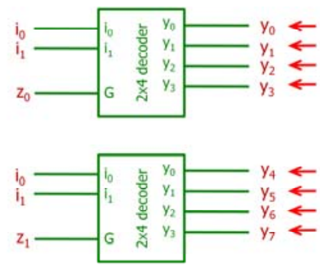


3x8 Decoder

G	i_2	i_1	i_0	y_7	y_6	y_5	y_4	y_3	y_2	y_1	y_0
0	x	x	x	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	**1**
1	0	0	1	0	0	0	0	0	0	**1**	0
1	0	1	0	0	0	0	0	0	**1**	0	0
1	0	1	1	0	0	0	0	**1**	0	0	0
1	1	0	0	0	0	0	**1**	0	0	0	0
1	1	0	1	0	0	**1**	0	0	0	0	0
1	1	1	0	0	**1**	0	0	0	0	0	0
1	1	1	1	**1**	0	0	0	0	0	0	0

- Connect the outputs as shown
- Connect the least significant inputs to the inputs of the output decoders
- Each combination of i_1 & i_0 will produce two asserted outputs; thus, we need to enable the decoder with the desired output and disable the others …
- Connect the first level decoder shown based on the remaining inputs. In this case, i_2 of the new decoder. When $i_2 = 0$, the top decoder is enabled. When i2 = 1 the bottom decoder is enabled.
- Connect the enable for the new decoder to the enable of the first level mux.





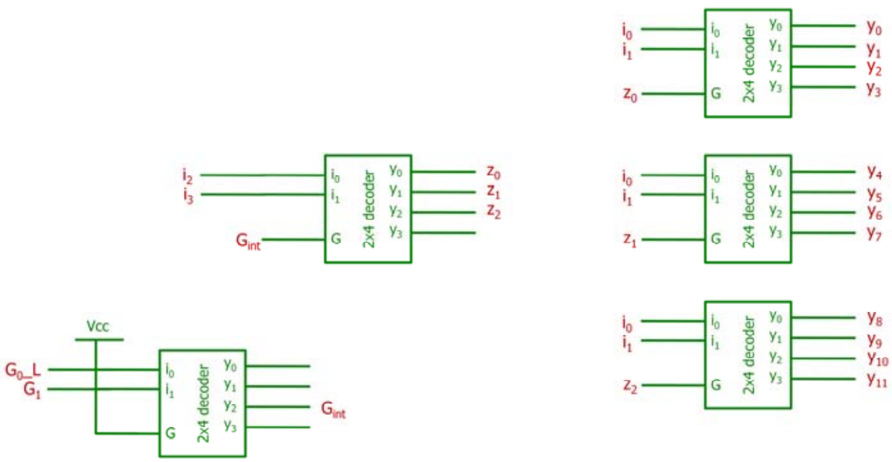
NOTE: Unused inputs must connect to something … unused outputs are left hanging.

4x12 Decoder (non-std.)

Decode binary word to 1-of-12 code (12 outputs)

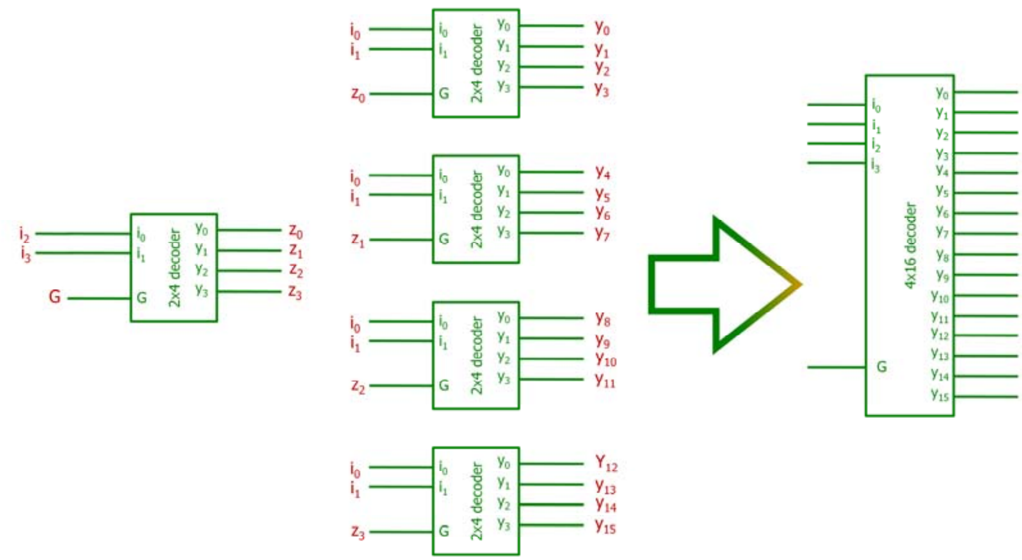
G	G_0_L	i_3	i_2	i_1	i_0	y_{11}	y_{10}	y_9	y_8	y_7	y_6	Y_5	y_4	y_3	y_2	y_1	y_0
x	1	X	x	x	x	0	0	0	0	0	0	0	0	0	0	0	0
0	x	X	x	x	x	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**1**
1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	**1**	0
1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	**1**	0	0
1	0	0	0	1	1	0	0	0	0	0	0	0	0	**1**	0	0	0
1	0	0	1	0	0	0	0	0	0	0	0	0	**1**	0	0	0	0
1	0	0	1	0	1	0	0	0	0	0	0	**1**	0	0	0	0	0
1	0	0	1	1	0	0	0	0	0	0	**1**	0	0	0	0	0	0
1	0	0	1	1	1	0	0	0	0	**1**	0	0	0	0	0	0	0
1	0	1	0	0	0	0	0	0	**1**	0	0	0	0	0	0	0	0
1	0	1	0	0	1	0	0	**1**	0	0	0	0	0	0	0	0	0
1	0	1	0	1	0	0	**1**	0	0	0	0	0	0	0	0	0	0
1	0	1	0	1	1	**1**	0	0	0	0	0	0	0	0	0	0	0

4-to-12 Decoder (non-std.)



4-to-16 Decoder

- 4x16 Decoder with an active-high enable



Demultiplexer (demux)

- A demux performs the reverse operation from the mux
- The demux routes one input to 1-of-n outputs based on the select-line input combination
- System description for 4 output demux (2 select-lines)
  - $y_0 = i \cdot s_1' \cdot s_0'$
  - $y_1 = i \cdot s_1' \cdot s_0$
  - $y_2 = i \cdot s_1 \cdot s_0'$
  - $y_3 = i \cdot s_1 \cdot s_0$
- Recall from the decoder a description could be given as follows:
  - $y_0 = G \cdot s_1' \cdot s_0'$
  - $y_1 = G \cdot s_1' \cdot s_0$
  - $y_2 = G \cdot s_1 \cdot s_0'$
  - $y_3 = G \cdot s_1 \cdot s_0$

Demux (concluded)

- You can see, from the previous expressions, that the demux could be implemented with a decoder that has an enable

- Just connect I to G of the decoder and you have a demux.

•Warning: Most CAD tools rely on the user knowing this fact …
… the libraries do not contain any demuxes, just decoders.