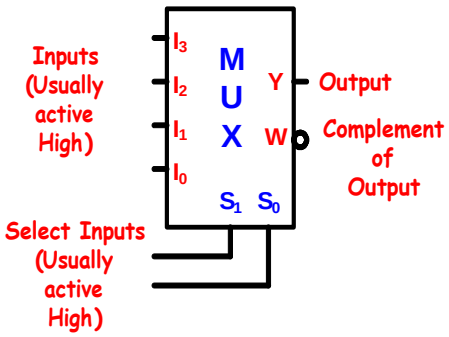
Multiplexers (Data Selectors) Introduction

A **multiplexer (MUX)** is a logic component that has several inputs but only a single output **(might have a 2nd output which is the complement of the 1st.)** With the addition of the MUX, the designer has the capability to direct one of the inputs to the output. For instance, take the figure in example 4-10. One might have several pulse trains on the 4 inputs, each with a different set of characteristics. Then the MUX can direct each signal one at a time onto a single transmission wire. At the other end, a de-MUX can sort the signals back out onto 4 lines. The only requirement is that each end must be **sync'ed together** so that the signals are sorted correctly.

The Generic MUX

In the digital MUX, any signal or logic level can be placed on any of the inputs. Binary numbers placed on the **Select (S)** lines **determine which input will be connected to the output.** The select lines are **selecting the subscript associated with the desired input.** It is a **ONE WAY DEVICE.**

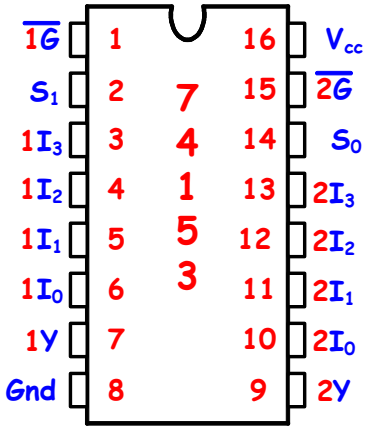


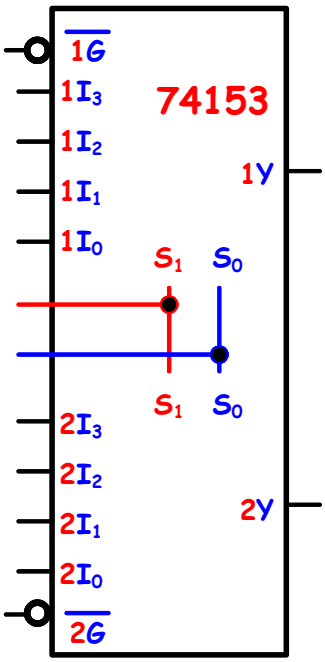
Generic MUX

One use for this would be to place several different frequencies and/or wave-shapes as inputs to the MUX. Then, a different input signal can be selected by just changing the "S" lines. **(Note: The S, or 'Select', lines are sometimes called the 'Control' lines).**

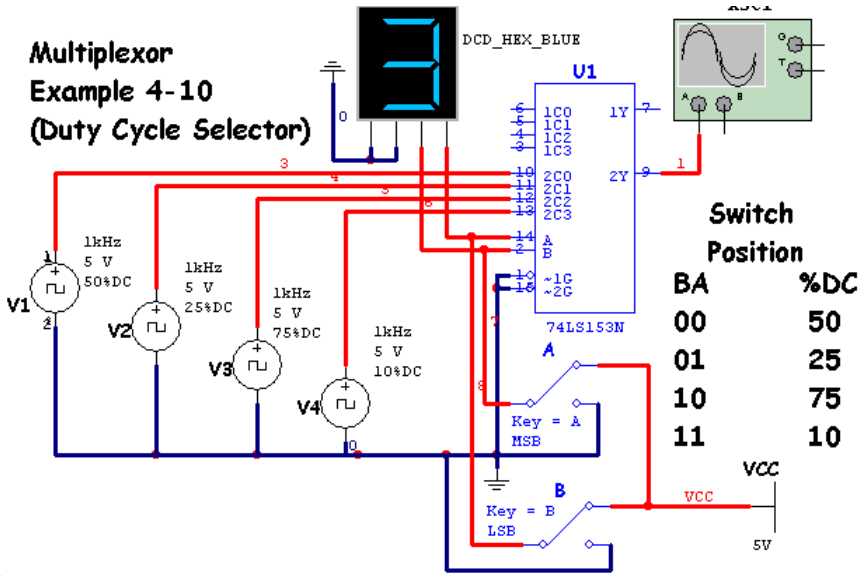
The 74153 dual 4/1 MUX

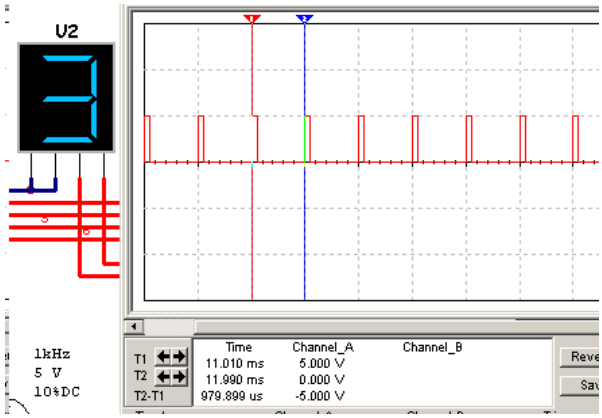
The first MUX we will discuss is the 74153 (dual 4/1 MUX). (As shown in the previous example.) While this is indeed a DUAL MUX, the two MUX's **share the select lines** so **you wouldn't be able to use the two for two different purposes**. You can either use a single MUX or you can put the two MUX's together to create a single 8 to 1 MUX. Note that this chip doesn't have a "**Complement of Y**" output.

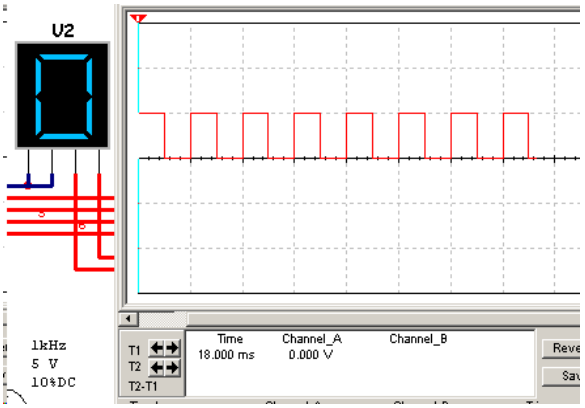




Example 4-10



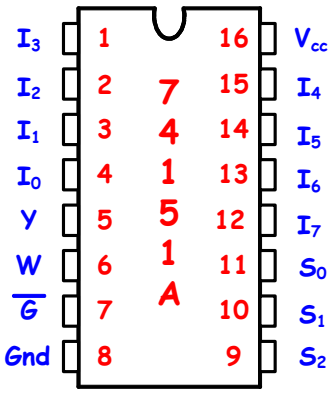


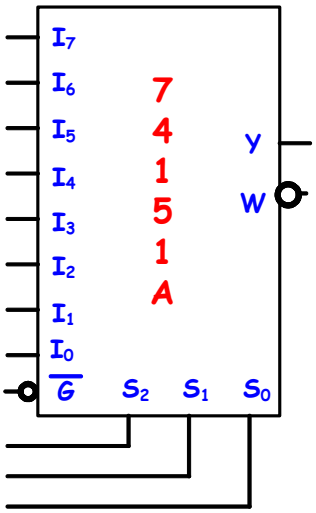


(The select lines controlled the duty cycle shown on the output of the Mux)

The 74151 8/1 MUX

This MUX is a **single 8/1 MUX** with an **active low Chip Enable input**. Since it has 8 input lines to select between, it has 3 select lines.





Other Multiplexer Applications:

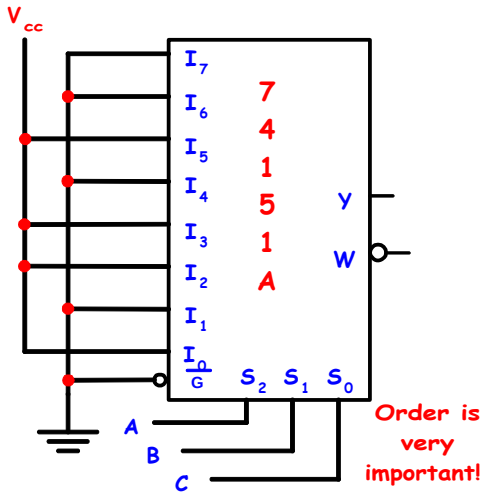
One very important use for the MUX is to be able to implement switching functions. Unlike the decoder which also was used for this purpose, **the MUX has the entire truth table effectively programmed into it.** The best way to understand this process is via examples.

Switching Function Implementation Example 1:

Implement a 3 variable switching function with an **8/1 MUX**

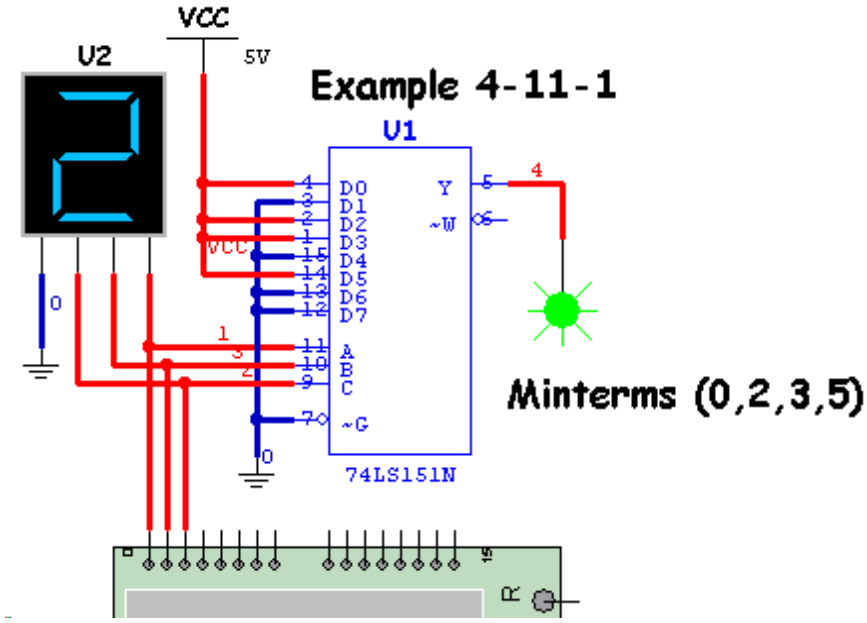
Implement $f(a,b,c) = \sum m(0,2,3,5)$ with an **8/1 MUX**.

A	B	C	Y
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	0



Example 4.11.1

The order of the variables on the select lines is VERY important!



Type number

The type number of a MUX circuit will be a **function of the # of variables in the switching function and the # of inputs on the MUX**. For example, a **three variable function** coupled with an **8:1 MUX** will create a **Type 0** implementation. A **four variable function** coupled with a **4:1 MUX** will result in a **Type 2** implementation. The equation for this process is:

$$2^{(\#\text{var}-\textit{type}\#)} = \textit{size of Mux}$$

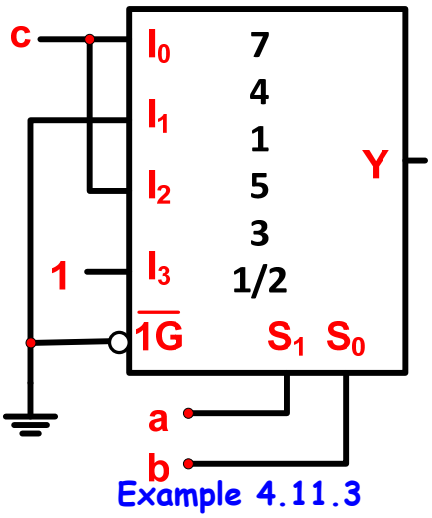
The circuit in the last example is a **Type "0" Implementation of the switching function**. The **Type # indicates the # of variables which will go into the Input lines (I_x)**. Since all three variables go into the '**Select** lines' and '0'(None) inputs go into the input lines here, this circuit is a **Type "0"** implementation.

MUX Example 2

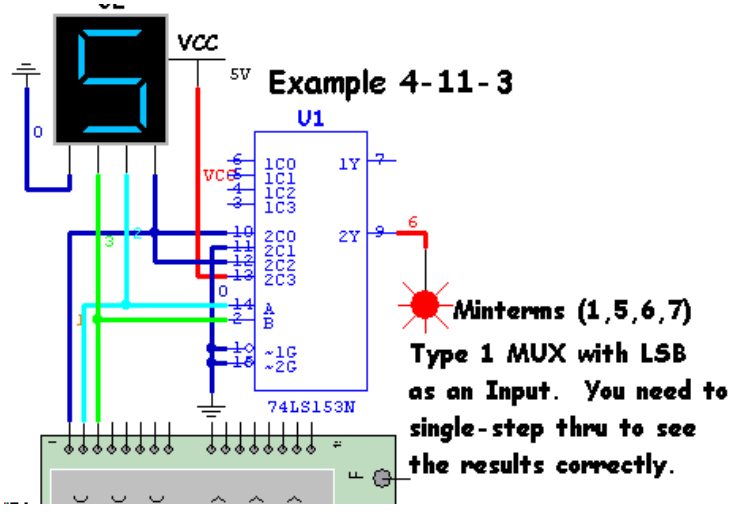
Implement the following switching expression using a 4:1 MUX. Work the problem two different ways. First give a **Type "1" implementation using "c" as an input and then using 'a' as an input.**

$$f(a,b,c) = ab + \bar{b}c = ab(c + \bar{c}) + (a + \bar{a})\bar{b}c$$
$$= \underbrace{abc}_{7} + \underbrace{ab\bar{c}}_{6} + \underbrace{a\bar{b}c}_{5} + \underbrace{\bar{a}\bar{b}c}_{1} = \sum m(1,5,6,7)$$

Row	Select lines a	b	c	Y	
0	0	0	0	0	
1	0	0	1	1	$I_0 = C$
2	0	1	0	0	
3	0	1	1	0	$I_1 = 0$
4	1	0	0	0	
5	1	0	1	1	$I_2 = C$
6	1	1	0	1	
7	1	1	1	1	$I_3 = 1$



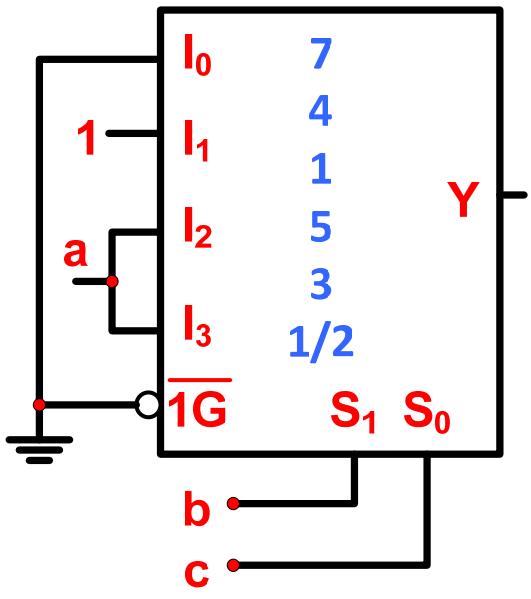
Example 4.11.3



MUX Example 2 a different way

Next, rework the problem but this time use (variable 'a')(the MSB) as the input instead of (variable 'c') (the LSB).

	MSB	Select lines		Out	
Rows	a	b	c	Y	
0	0	0	0	0	I_o = '0'
1	0	0	1	1	I_1 = '1'
2	0	1	0	0	I_2 = 'a'
3	0	1	1	0	I_3 = 'a'
4	1	0	0	0	I_0 = '0'
5	1	0	1	1	I_1 = '1'
6	1	1	0	1	I_2 = 'a'
7	1	1	1	1	I_3 = 'a'



Example 4.11.3.1

This circuit will work just as well as the previous one. The difference is that the designer had to work harder to find the answer since the equivalent rows were no longer adjacent. **The harder the job, the easier it is to make a mistake!**

Question: **Then why do it at all? Why not always choose the lowest significance bits to go into the input structure of the MUX (not the Select lines)?**

Answer: **Because the job of the designer isn't to make life easier on himself. His job is to design the simplest, most reliable, and least expensive systems he can. If using a different set of columns will result in a better design, then he better do it. This example doesn't turn out any better but it is a good practice problem.**

So, if this new method makes it easier to make mistakes, it is also the job of the designer to apply **the METHODICAL engineering thought process** to make it simpler and ensure that there won't be any errors. The thing to do is to reorder the table based on the values in the columns going into the select lines. Let's look at the example again.

MUX Example 2 still again!

You should note that this process of reordering the rows results in the same circuit inputs to each I_x line but it is clearer what each input should be! Note that we moved the MSB line over to the right to be next to the output line so we could perform an easier comparison. But the 'a' column is **STILL the MSB.**

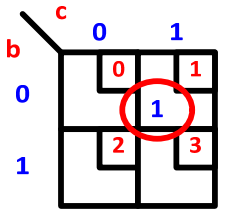
	Select Lines		msb		
Row	b	c	a	Y	
0	0	0	0	0	$I_0 = 0$
4	0	0	1	0	
1	0	1	0	1	$I_1 = 1$
5	0	1	1	1	
2	1	0	0	0	$I_2 = a$
6	1	0	1	1	
3	1	1	0	0	$I_3 = a$
7	1	1	1	1	

MUX Example 2 one last time

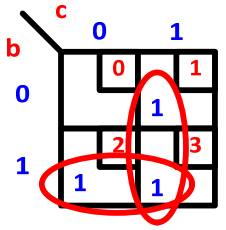
Now, let's rework the problem but this time we will design a type 2 system. Since there are 3 variables, the MUX size would be:

$$MUX\ size = 2^{(\#\ of\ variables\ -\ type\ \#)}$$
$$MUX\ size = 2^{(3\ -\ 2)}$$
$$MUX\ size = 2^1 = 2$$
$$MUX\ size = 2\text{-}to\text{-}1\ MUX$$

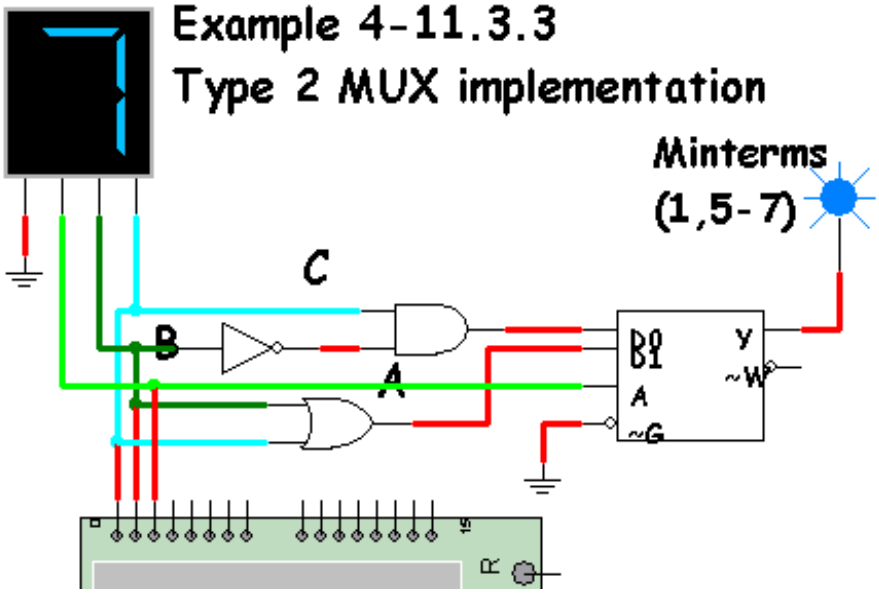
Rows	Select line a	b	c	Out y	
0	0	0	0	0	
1	0	0	1	1	$I_0 = \bar{b}c$
2	0	1	0	0	
3	0	1	1	0	
4	1	0	0	0	
5	1	0	1	1	$I_1 = b + c$
6	1	1	0	1	
7	1	1	1	1	



$$I_0 = \bar{b}c$$



$$I_1 = b + c$$



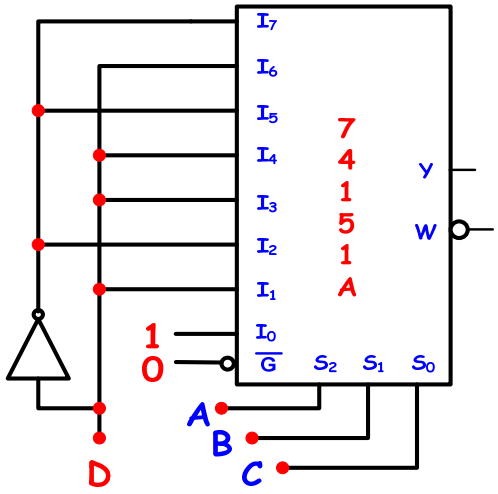
MUX Example 3

Implement the following switching expression using a Type 1 implementation.

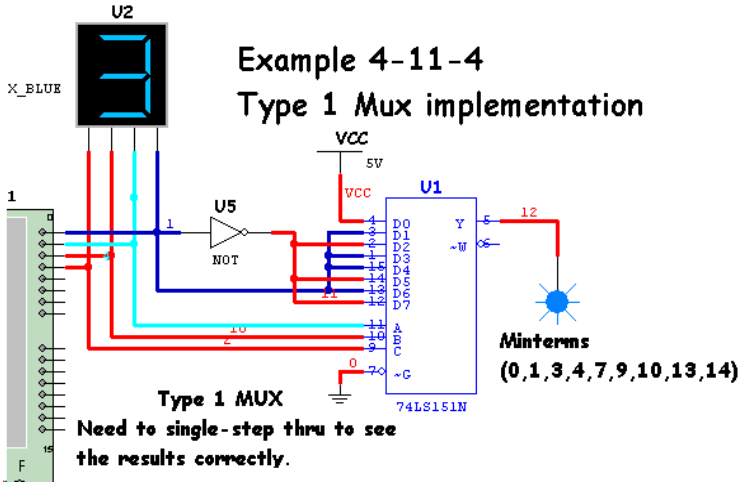
$$f(A,B,C,D) = \sum m(0,1,3,4,7,9,10,13,14)$$

$$\text{Size of Mux} = 2^{(\#\text{var-type}\#)} = 2^{(4-1)} = 2^3 = 8$$

S_2 A	S_1 B	S_0 C	D	y	
0	0	0	0	1	
0	0	0	1	1	$I_0 = 1$
0	0	1	0	0	
0	0	1	1	1	$I_1 = D$
0	1	0	0	1	
0	1	0	1	0	$I_2 = D'$
0	1	1	0	0	
0	1	1	1	1	$I_3 = D$
1	0	0	0	0	
1	0	0	1	1	$I_4 = D$
1	0	1	0	1	
1	0	1	1	0	$I_5 = D'$
1	1	0	0	0	
1	1	0	1	1	$I_6 = D$
1	1	1	0	1	
1	1	1	1	0	$I_7 = D'$



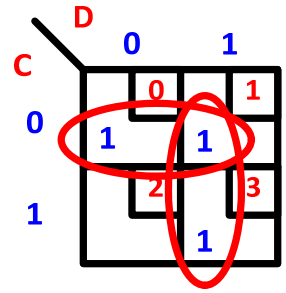
Example 4.11.4



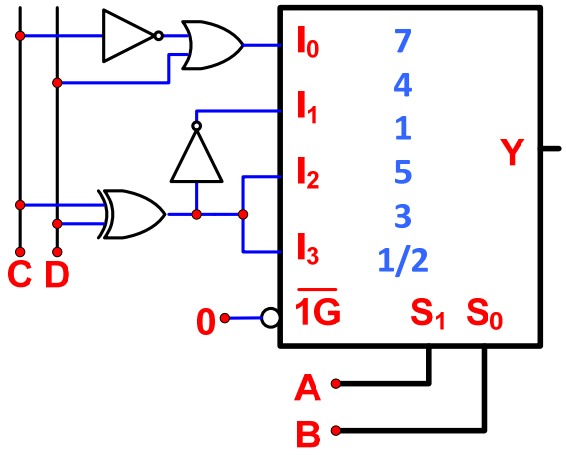
Always take as many of the most significant variables as you have select switches.

MUX Example 4

Implement the same function as before except use a 4:1 MUX.



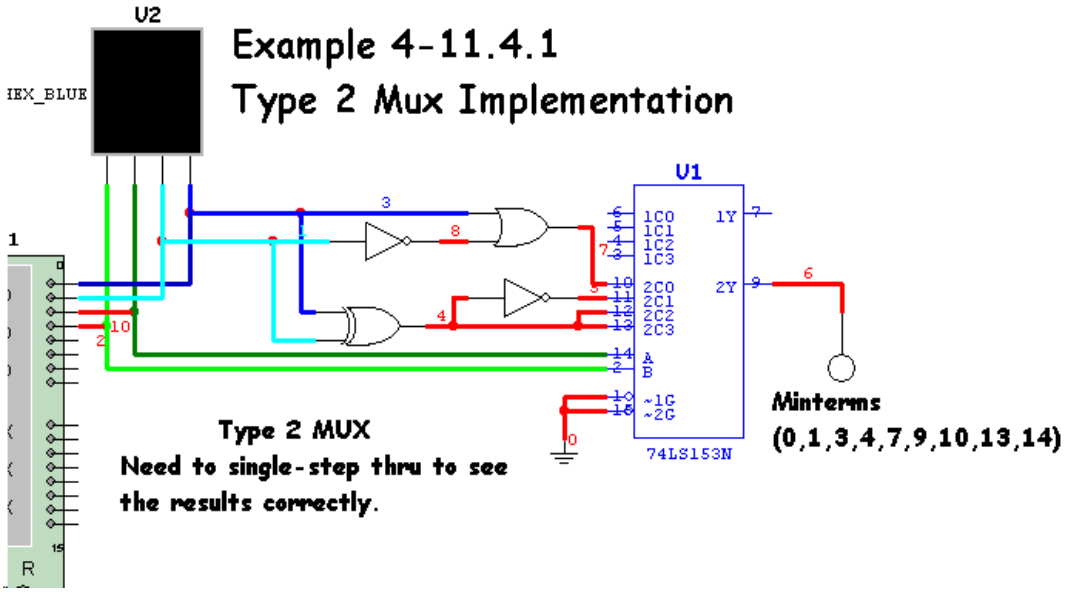
$$I_0 = \overline{C} + D$$



Example 4.11.4.1

S_1	S_0				
A	B	C	D	Y	
0	0	0	0	1	$I_0 = \overline{C} + D$
0	0	0	1	1	
0	0	1	0	0	
0	0	1	1	1	
0	1	0	0	1	$I_1 = \overline{C \oplus D}$
0	1	0	1	0	
0	1	1	0	0	
0	1	1	1	1	
1	0	0	0	0	$I_2 = C \oplus D$
1	0	0	1	1	
1	0	1	0	1	
1	0	1	1	0	
1	1	0	0	0	$I_3 = C \oplus D$
1	1	0	1	1	
1	1	1	0	1	
1	1	1	1	0	

I_1 and I_2 were found by "Observation".



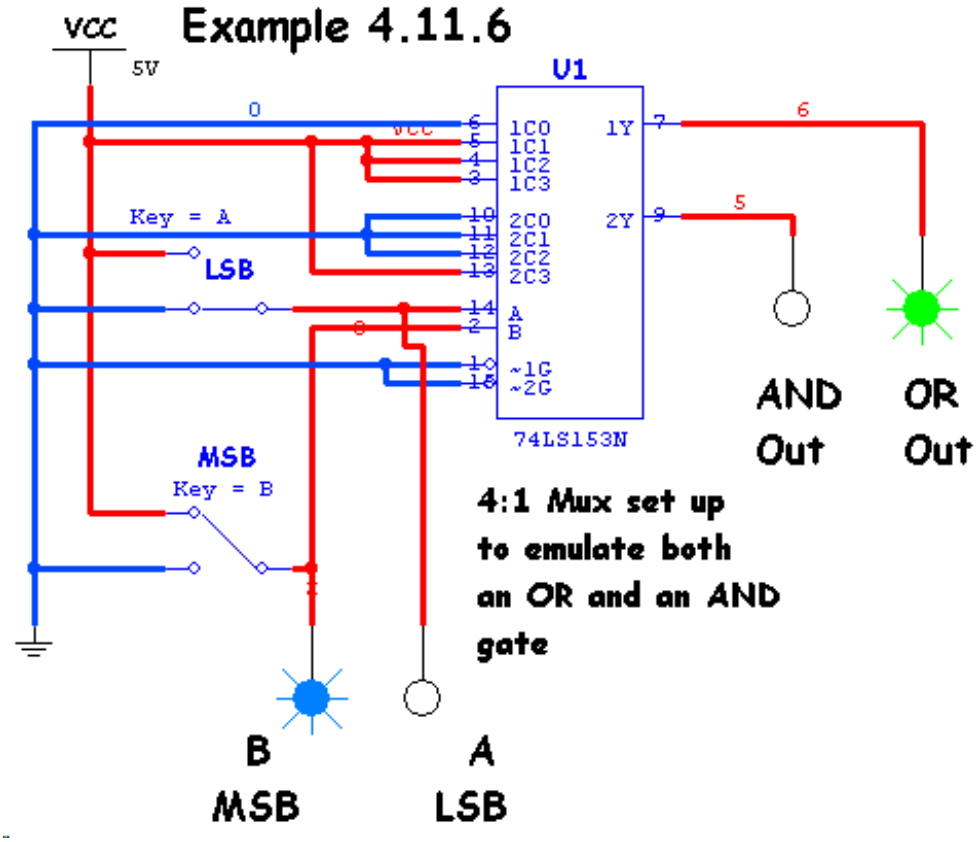
Type numbers greater than 2

Type 3 and above implementations are normally too complicated. Normally it is better to design the logic circuit straight out. **(Law of Diminishing Return) (at least, they are too hard to teach in class)**

MUX's used as Logic Elements

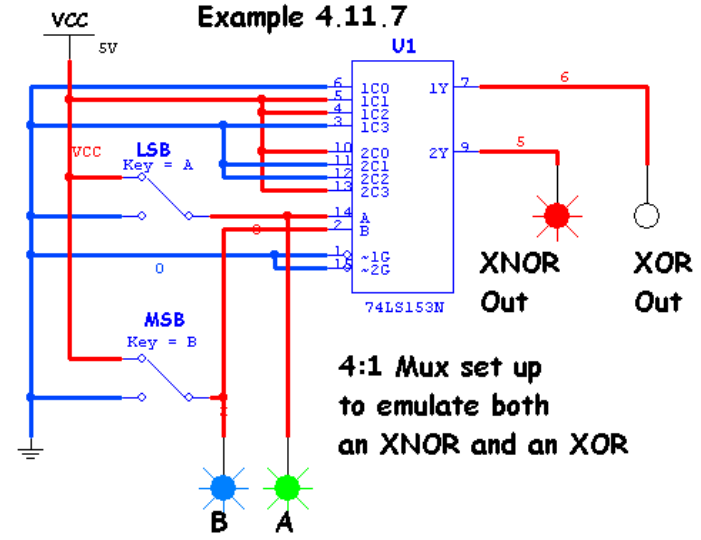
Implement a 74153 as both an OR gate and an AND gate:

Note that the example shows A = 0 and B = 1 with the corresponding correct OR and AND outputs.

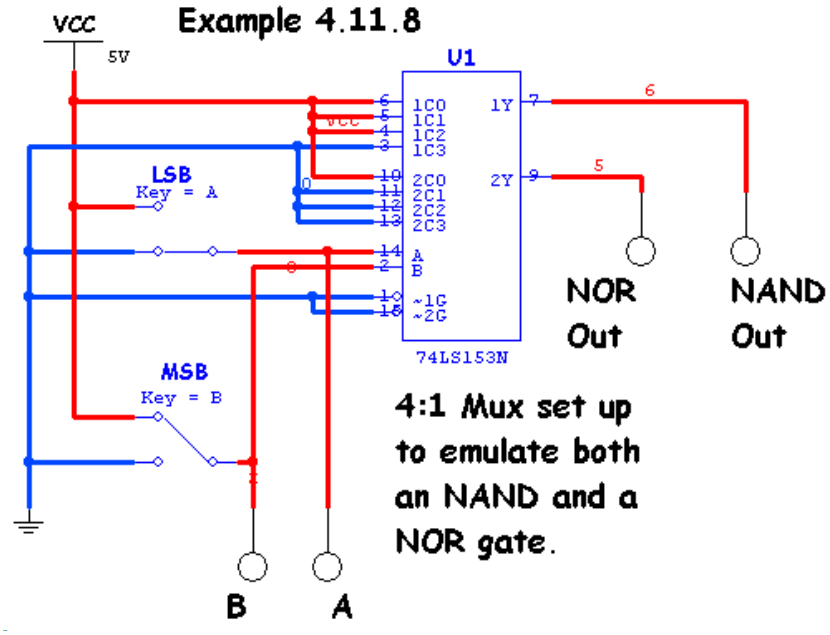


Multisim Example for XOR and XNOR 74153 implementation

Implement a 74153 as both an XOR gate and an XNOR gate:

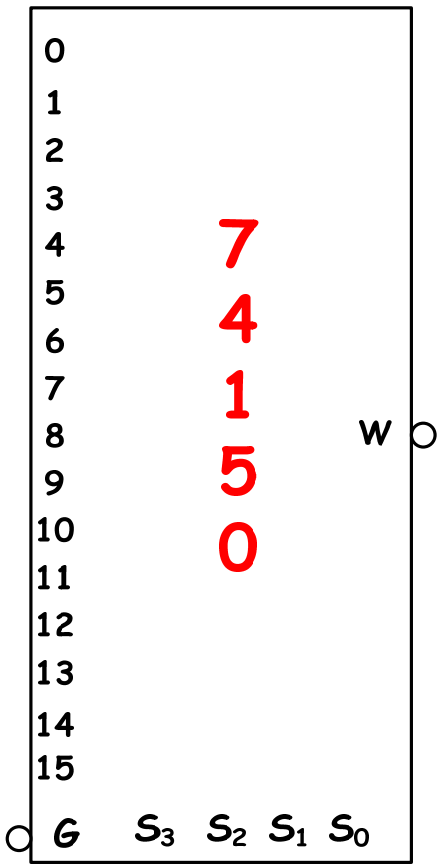


Multisim 74153 Implement of a NAND and NOR gate



The 74150 16/1 MUX

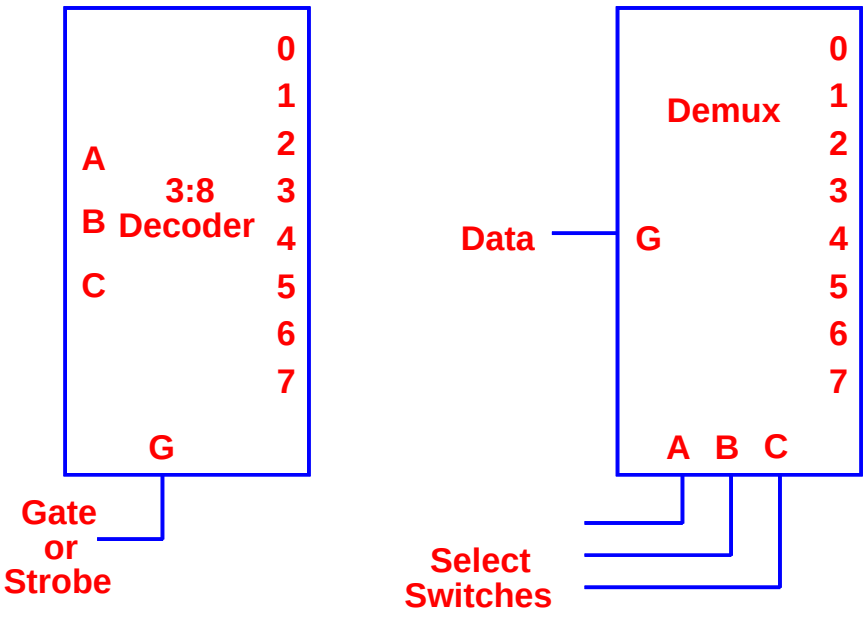
One last MUX which needs to be mentioned is the 74150, 16 to 1 MUX as shown below.



Unlike the other MUX's, instead of having an ACTIVE HIGH Y output, it has ONLY an ACTIVE LOW 'W' output. This device could have been used to create a TYPE '0' implementation of the previous 4 bit example.

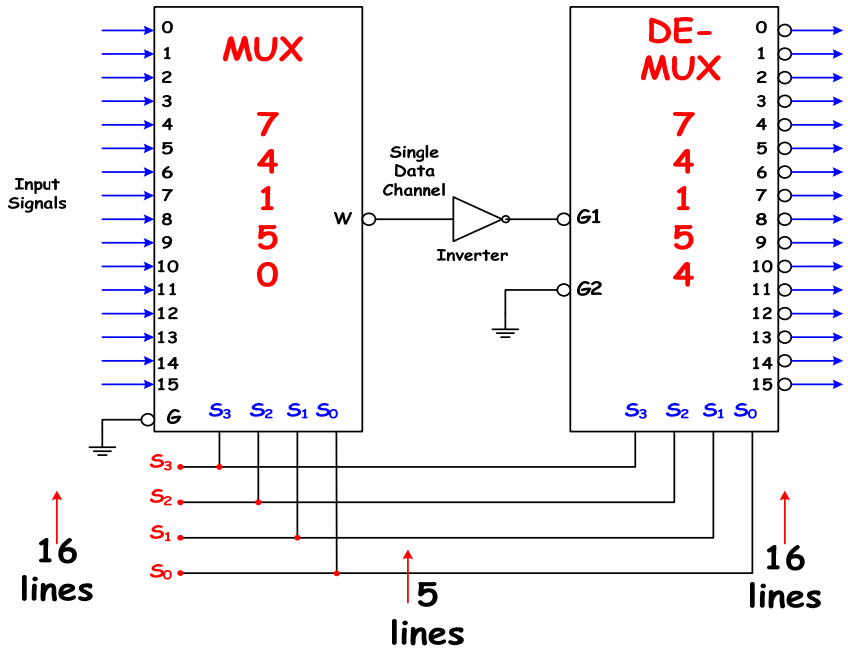
Demultiplexer (Data Distributer)

Now that we have discussed the Multiplexer end of the process, remember that at the other end of the trail was a Demultiplexer. A **Demultiplexer** is essentially **a Decoder which is drawn differently.**



The **Demultiplexer will connect a single data line with one of (n) output lines.** The specific output line is determined by the select switches.

Example:



The purpose of this circuit would be to share a single channel to transmit 16 separate signal channels to another location and then to split them back out to individual channels at the other end. **The inverter is used to match the bubbles.** **When this occurs, the output lines will effectively be active HIGH not active LOW!** An additional caution must be taken to take into account the **delay time** between the two sides when syncing them together. And finally it should be noted that there is a cost to performing this MUX - DEMUX operation. While we have saved on wire, system utility has been reduced. Each signal is now only available one at a time instead of all at the same time.