CALIFORNIA STATE UNIVERSITY
LOS ANGELES
Department of Electrical and Computer Engineering

EE-2449 Digital Logic Lab

EXPERIMENT 4
LOGIC FUNCTIONS

Text: Mano and Ciletti, *Digital Design, 5th Edition,* Chapter 2
Required chips:

7404: hex inverters (NOT) **7486**: quad 2-input XOR
7408: quad 2-input AND **7432**: quad 2-input OR

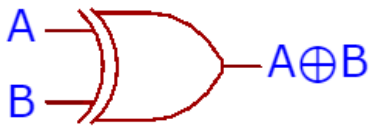
4.1* Combinational logic circuits implement logic functions using a combination of logic gates. Recall that all logic functions can be implemented with AND, OR, and NOT functions. Consider a 2-input XOR function. Recall that when inputs are different, the output is 1; when inputs are the same, the output is 0. Assume that the inputs variables are A and B. If A is true (1) and B is false (0) or if A is false (0) and B is true (1) the function will be true. Putting that into a logic equation:

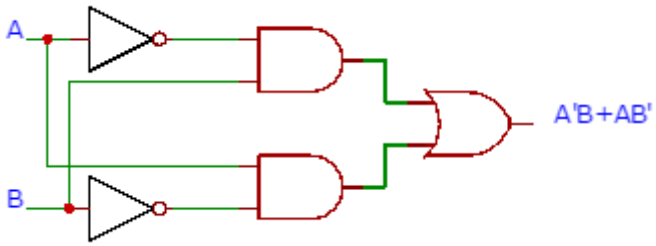
$$A \oplus B = AB' + A'B$$

One way to prove that two logic equations implement the same function is to show that they have the same truth table. The truth table below shows that $A \oplus B$ and $AB' + A'B$ have the same truth table, and, thus, they must implement the same function.

A	B	A ⊕ B	A'	B'	AB'	A'B	AB' + A'B
0	0	0	1	1	0	0	0
0	1	1	1	0	0	1	1
1	0	1	0	1	1	0	1
1	1	0	0	0	0	0	0

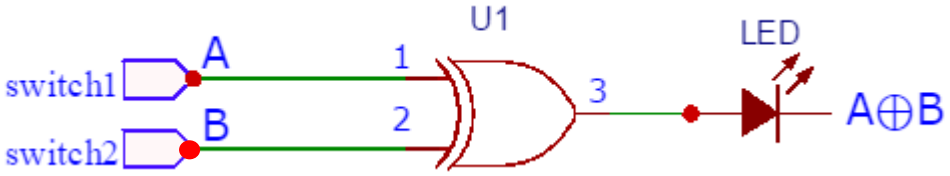
The logic diagrams for these two functions are:





If we implement both circuits they should have the same behavior. A logic diagram does not have enough information to implement a function. A wiring diagram is similar to a logic diagram but also has information about how to wire the chips such as chip type, unit numbers, and pin numbers. Below are the wiring diagrams for both logic functions.

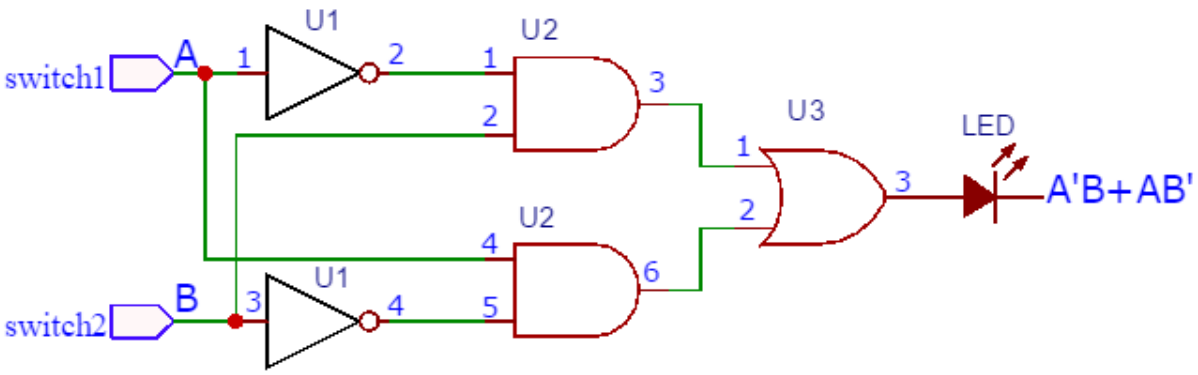
Wiring diagram for logic equation A ⊕ B:



Unit	Chip	VCC	GND
1	7486 (XOR)	14	7

In this diagram, there is one chip (Unit 1 or U1) which is a 7486. The power and ground table has the unit number, the chip type, the power (VCC) pin, and the ground (GND) pin information. Insert the 7486 into the breadboard and connect the power pin (pin 14) to VCC (+5V) and ground pin (pin 7) to GND (0V). Then connect a switch from the array of switches to pin 1 of U1, another switch to pin 2 of U1, and pin 3 of U1 to an LED from the LED bar graph. Build this circuit and verify that it produces the same results as given in the truth table for A ⊕ B.

Here is the wiring diagram for the logic function AB' + A'B:



Unit	Chip	VCC	GND
1	7404 (Inverter)	14	7
2	7408 (AND)	14	7
3	7432 (OR)	14	7

This circuit has three chips, U1 is a 7404 (Inverter), U2 is a 7408 (AND), and U3 is a 7432 (OR). Line up these chips along the same section of the breadboard, all facing the same way. Leave some space between them in case one of them is faulty and has to be extracted.

For each chip connect power pin (pin 14) to VCC (+5V) and the ground pin (pin 7) to GND (OV). Next connect all of the wires as shown in the wiring diagram. For example, connect pin 2 of U1 to pin 1 of U2. To keep track of which connections have been made, after you connect two pins with a wire, place a small check mark on that wire in the wiring diagram. It is also a good idea to place a check mark next to pin 14 and pin 7 in the power and ground table after you connect each to VCC and GND, respectively. **Refer to the Implementing and Troubleshooting Designs (Implement) section of this manual for more hints on wiring circuits.**

Build this circuit and verify that it produces the same results as given in the truth table for AB' + A'B.

Note: there is a problem drawing wiring diagrams as discussed above. You must constantly refer to the section at the end of the manual called IC Specifications or else go on line to find each chip's pin numbers and then to place them in your drawing. Instead, you can use a circuit-drawing program called ExpressSCH. (SCH stands for "schematic", meaning a circuit diagram). It allows you to place and connect symbols for gates and chips in your diagram that contain all pin numbers plus Vcc and ground (so a Vcc-ground table is no longer needed).

Once you draw a circuit with this tool, it's all right there. (You may want to print it out so you can check off connecting wires as you install them.) The program is available on the lab PC's plus you can download it from the web for free. Refer to the section at the end of this experiment called **Drawing Circuits (ExpressSCH Tutorial)** for all instructions on how to download it to your computer and how to use it. It is of real benefit when wiring up your circuit to have your diagram on the PC screen, complete with pin and U (unit) numbers.

4.2 Rarely will a circuit work the first time it is built. Inevitably there will be mistakes, also known as "bugs", either due to an incorrect design or a mistake during wiring. In order to find the mistake you need to troubleshoot or "debug" your circuit. For a combinational logic circuit here are some steps to debugging a circuit:

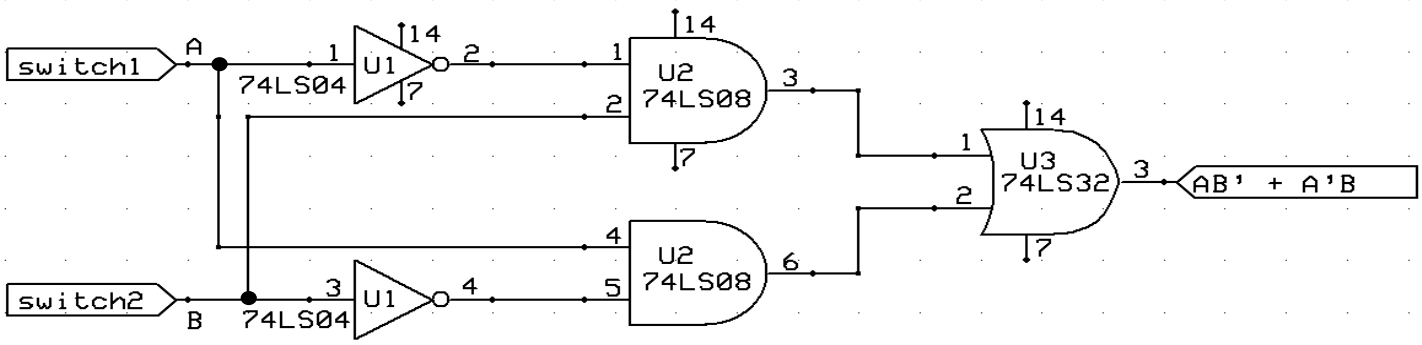
Steps for troubleshooting (debugging) a combinational logic circuit:

1. Determine whether or not the circuit generates the correct outputs for all possible input combinations. If so, the circuit works properly. If not, record the input combinations that generate incorrect outputs. (The number of inputs can't be too high, else too many combinations to test.)
2. For one of the input combinations that physically produced an incorrect output, show on the diagram what the correct input and output of each gate should be (internal gates as well as the final output).
3. Now, set the inputs to the circuit equal to that input combination. Note that while debugging a circuit it is best to manually control the inputs by connecting them to switches or by driving the 7493 counter, if used, by the pushbutton switch (pulser).
4. Using the logic probe, start from the output of the circuit and trace the values backwards through the various components until you identify an incorrect value (a value that differs from the one you determined in step 2).
5. Identify and fix the bug.
6. Repeat steps 1 through 5 until there are no bugs remaining in the circuit.

There are two keys to being successful in troubleshooting your design. First, you must understand how your circuit is supposed to behave. Also, you need to have a good understanding of the behavior of each logic gate, hence the importance of Experiment 3.

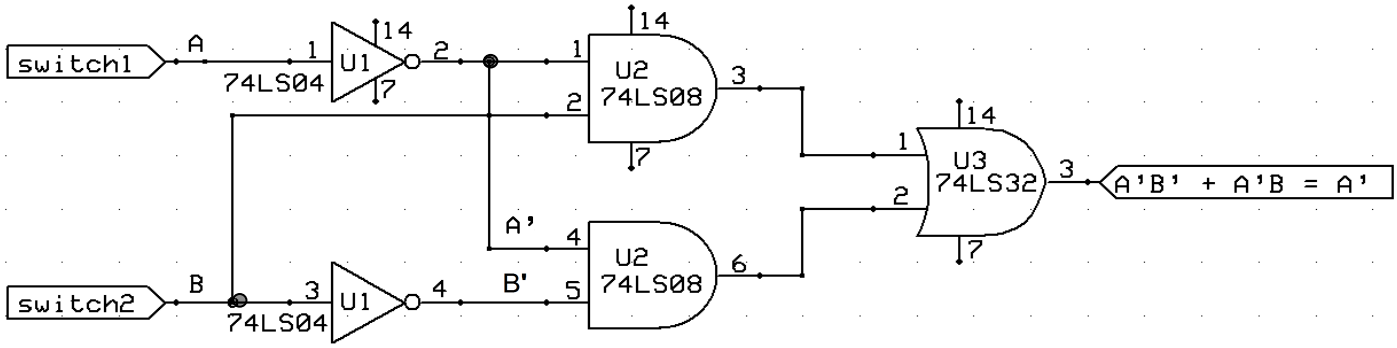
The circuits shown in the following debugging session have been drawn using ExpressSCH. Notice how gates from different chips show different U numbers and how pin numbers are included automatically. Once you are familiar with ExpressSCH, you'll see how much more efficient it is draw your circuits.

Let's experiment with debugging a circuit. First, this assumes that you have already built and tested the AB' + A'B circuit from section 4.1 (above) and have verified that it works properly. If it doesn't, you can use the following debugging steps to try to fix it.



CORRECT XOR CIRCUIT

Suppose the bug in your circuit is due to the fact that instead of connecting the wire from U2 pin4 to switch-A, you connected it to U1 pin2, as shown below. (A fairly easy mistake to make.)



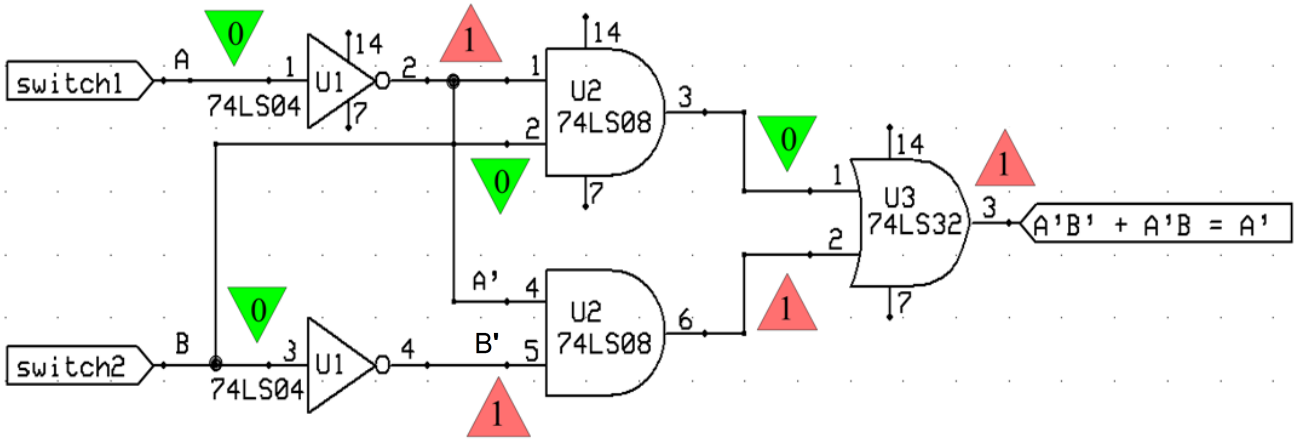
CIRCUIT WITH ERROR

Debugging steps.

Step 1: with the bug, the circuit will generate the output values shown in the following truth table under Actual Value, instead of the correct values shown under A ⊕ B. There are two input combinations that generate the incorrect results shown in bold. From the values in the Actual Value column, show that the expression for the output of the circuit is now just A'.

A	B	A ⊕ B	Actual Value
0	**0**	0	**1**
0	1	1	1
1	**0**	1	**0**
1	1	0	0

Step 2: for input combination, 00, the correct output value should be 0, but a check with your logic probe at the output of U3 (OR) will show it is 1. We start troubleshooting (debugging) by working backwards from the output gate using the logic probe. The triangular symbols (red=1, green=0) represent the LED indicators on the logic probe. As you proceed, record the 1 and 0 values on your diagram.



CIRCUIT WITH ERROR
Output becomes just A'

1) We first check the input pins of the OR. If the OR output is 0, then its inputs, pins 1 and 2, should both be 0's. The input at pin 1 turns out to be 0, as expected, so it's OK. But the input at pin 2 is a 1, which explains why the output of the OR is also 1 (i.e. 0 + 1 = 1). Since pin 1 is OK, we will ignore the upper half of the circuit and focus instead on the bottom half to see why the input to pin 2 is wrong.

2) Now, if pin 2 of the OR is properly connected to pin 6, as the diagram shows, then the AND output will also be a 1. But an AND output of 1 implies that both its inputs, pins 4 and 5, must be 1's (output = 1•1 = 1) and the logic probe should show that they are.

3) That pin 5 is a 1 is expected since B' is the complement of switch B (0). On the other hand, we find pin 4 to be 1 even though it is supposed to connect directly to switch A (0). If we follow the wire from pin 4, we find that it ends, not at switch A but at A', the output of the upper inverter. We have located the bug. The circuit output is, therefore, A'B' (from lower AND) + A'B (from upper AND) = A' (B'+B) = A', as the above truth table indicates.

Alternative debugging method:

Assuming there are no electrical (as opposed to logic) problems with your circuit, problems like a bad chip, or one with no connection to Vcc or ground, or a broken wire, the following approach may save time since it doesn't involve poking around inside the circuit with a logic probe as in the previous example. It just requires adding a column to the original truth table with output values found when testing the actual circuit. We'll use the table for the XOR circuit shown at the top of the previous page to illustrate this.

A	B	A ⊕ B	Actual Value
0	**0**	0	**1**
0	1	1	1
1	**0**	1	**0**
1	1	0	0

The 1's in the Actual Values column lie in the first two rows where A = 0. So we could say that the circuit output is simply A'. But this implies a circuit with no AND and OR gates--very different from the XOR circuit. In order to compare the two, we want the same structure for each (i.e. two inverters, two ANDs and an OR). So, instead, we write separate AND expressions (product terms) for the first and second rows in the Actual Values column and combine them as shown:

Actual output = A'B' + A'B
XOR output = AB' + A'B

We can see right away that the problem is the underlined A' which should be A. In other words, the wire into pin 4 of the lower AND should come directly from the A switch and not from the output A' of the upper Inverter. Problem solved.

(Refer to the Appendix at the very back of this experiment for more on this alternative debugging approach.)

4.3* A majority voter is a circuit with three inputs that we'll refer to as A, B, and C. It has one output which we'll refer to as V. The value of V will agree with the majority value on the inputs. Thus, if the majority (two or more) of the values are 0, V will be 0. If the majority of the values are 1, V will be 1. Given this definition of the majority voter, complete the truth table below for V.

A	B	C	V
0	0	0	
0	0	1	
0	1	0	
0	1	1	
1	0	0	
1	0	1	
1	1	0	
1	1	1	

Let's think about this circuit. If two or three inputs are 1, output V will be 1. Otherwise, it will be 0. Verify that your values for V in the truth table agree with this definition. If so, then we can determine a logical expression for V; i.e. V = AB + BC + AC. We can verify that this is correct by showing that the truth table for V and for AB + BC + AC are the same. Complete the truth table below to prove that these two functions are the same.

A	B	C	V	AB	BC	AC	(AB+BC)+AC
0	0	0					
0	0	1					
0	1	0					
0	1	1					
1	0	0					
1	0	1					
1	1	0					
1	1	1					

Draw the logic diagram for V = AB + BC + AC. Also draw the wiring diagram. Implement the circuit and verify that it behaves correctly. If it does not, use the debugging steps in section 4.2 to troubleshoot your design.

4.4* Assuming that you were lucky and wired your circuit properly, you haven't had a chance to practice your debugging skills. In this section, you will each take turns inserting a bug for your partner after which your partner will troubleshoot the circuit. Follow the steps shown in section 4.2 to debug your circuit. Document your process as shown in section 4.2 in your lab journal. Demonstrate your debugging technique to your instructor.

Laboratory Report Guidelines

Technical writing is a very important skill to acquire. A key component of the engineering design process is to effectively communicate your findings to others. Laboratory reports are different than project reports in that you are generally following a prescribed procedure. Regardless, laboratory reports should follow the same basic guidelines:

1. Use your own words. Do not copy from the laboratory manual or any other reference.
2. Write using an objective stance, that is, do not use I/we.
3. Reports should be concise, clearly organized, and well written.
    a. Provide section headings, left justified and in bold.
    b. Clearly delimit paragraphs with a space between each paragraph. Make sure your paragraphs are well written with a topic sentence and supporting sentences that have a logical flow.
    c. Proof read your report. There should not be any typos.
    d. Include figures and tables to effectively describe your design and results. Provide figure and table captions. Figure captions should be centered below the figure and have the format: "Figure 1. Four-bit ripple counter (7493)." Table captions should be centered above the table and have a similar format (use "Table" instead of "Figure"). In the text of your write-up you should refer to the figures and tables by number (e.g., "Figure 1 shows …" rather than saying "The figure below shows …").
    e. Briefly describe your design and results in the text of your report in addition to showing them in figure, graph, or table form.

Your laboratory report should include the following:

- **Title page:** Include the title of the experiment, the laboratory name (EE 2449 – Digital Logic Laboratory), your name, and the date.

- **Introduction:** Include the purpose of the experiment and summarize relevant background theory.

- **Body:** Include procedure/method/design, results, and discussion of findings. Organize the body so that you describe each circuit you design, build, and test separately.

    - Describe the purpose of this step and your procedure.
    - If appropriate, show truth tables, K-maps, state diagrams, state tables, and other methods used to design the circuit.
    - Show the wiring diagram of the circuit (include the power and ground table).
    - In addition, describe any challenges you had in completing the experiment including a brief description of steps taken to trouble shoot (debug) your circuit
    - If appropriate, include waveforms captured from the oscilloscope. Discuss how the results compare to the expected results. If you make any other observations, include them in your report.
    - Include the answer to any questions posed in the laboratory manual.

- **Conclusion:** Briefly summarize the purpose of the laboratory and your findings, observations, and recommendations.

- **References:** Include references at the end of your report. At a minimum you should reference the laboratory manual. In addition, reference any other sources you use such as the textbook, websites, etc.

  Example (IEEE Reference Style):
  [1] J. Levine, N. Warter-Perez *EE-2449 Digital Logic Laboratory*. Los Angeles, CA: California State University, Los Angeles, School of Engineering, 2016.

- **Appendix:** Normally your pre-labs and lab notes (with the instructor sign-off) would be included in the appendix. Since we are using laboratory notebooks that will be submitted at the end of the quarter, you do not need to include these in the appendix. You may include an appendix if there is any additional information you would like to include.

** **Note:** Lab reports should be self-contained (i.e., include all of the necessary information to understand the experiment) but also be concise. DO NOT simply restate the experimental procedure given; instead summarize the procedure in your own words.

Drawing Circuits (ExpressSCH Tutorial)

In EE 2449, we are currently using the program **ExpressSCH to** draw circuit diagrams. ExpressSCH is part of a two-program package called ExpressPCB. ExpressSCH is a **sch**ematic-capture program used to draw circuit wiring diagrams; the other program, ExpressPCB, is a **p**rinted-**c**ircuit-**b**oard program used to lay out circuit components on a printed circuit board. Since the latter function is not part of what you do in EE 2449, this tutorial focuses only on ExpressSCH.

You can download ExpressPCB from the net for free (go to *expresspcb.com*). Installation should put two icons on the screen. The one for ExpressSCH is shown at right; ignore or delete the one for ExpressPCB.

The program provides a listing of files containing symbols for commercially available components (gates and chips) which you can access when you open ExpressSCH. However, the listing contains far more components than you will need, and it also lacks some that had to be custom made.

Instead, you can obtain the files (including custom files) for only those components used in the lab. For your desktop or laptop, do the following:

Go to the website, **http://www.calstatela.edu/ecst/ece/ee-2449-digital-logic-lab**, click on 2449 under lab manuals and then click on 2449_parts.zip. Once it downloads (check bottom of screen at left), click to open. A folder will appear called "parts2". It contains the files representing all components (gates, counter, decoder, etc) that you will use during the lab. Copy them and go to “My Documents” in your PC. A folder, *ExpressPCB*, was created there when you downloaded and installed the program ExpressPCB. Store the files in a subfolder called **SchComponents_Custom**..

Using ExpressSCH

When you open the program, you will see menu items at the top and below them a row of buttons including zoom in and out. The drawing area shows grid markings. When you place objects on the screen, they will snap to the grid. (This feature and grid spacing can be changed, but there will probably not be a need for that.)

To find the components you'll want to place in the drawing area, right click anywhere in the area and select *Component Manager* (actually "Component and symbol Manager").

- On your desktop or laptop, click on *Custom Components*; you will find all the files you stored there (i.e. all the components used in the lab).
- On a lab PC, click on *Library Components*. The files you want are at the top of a long list. (They are followed by a listing of Connectors and after that a listing of files whose names all have an "IC-" prefix. *Ignore all of these*.)

Note that all of the components you'll be using are in the 74LSxx series (e.g. 74LS00, 74LS86, etc.) the

letters between 74 and the number at the end represent the electrical, not the logical, properties of the device. Thus, designations like 7400, 74LS00, 74HC00 all describe chips with four 2-input NAND gates. Their logic is the same even though they are somewhat different electrically.)

Now, if you left-click on, say, 74LS00, you will see all four gates displayed on the right. Although the gates all belong to one chip, they are displayed separately (not inside a chip rectangle). The same is true for flip-flops and, in general, for any chip containing multiple identical elements. Notice that pin numbers are different for each gate, as they should be. Also, the top gate shows the power and ground pins (14 and 7) for this chip. They serve to remind you not to leave them unconnected.

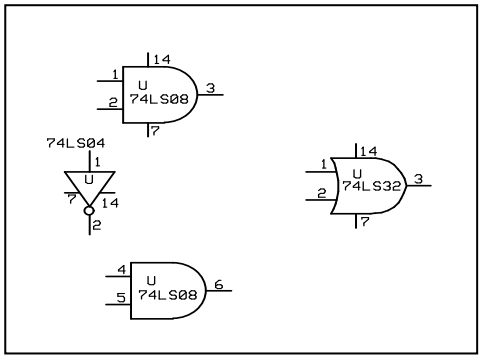
Now double click on 74LS00. You will see all four gates (in blue) in the drawing area. If you don't need all four in your circuit, delete the extra ones from the bottom up so the one at the top with power and ground pins will remain. With the mouse, simply draw a box around the extras with the mouse. Hit the Delete key and they're gone. The remaining ones turn black. You can enlarge the symbols with the mouse's wheel or by clicking on the zoom button ⊕ at the top of the screen.

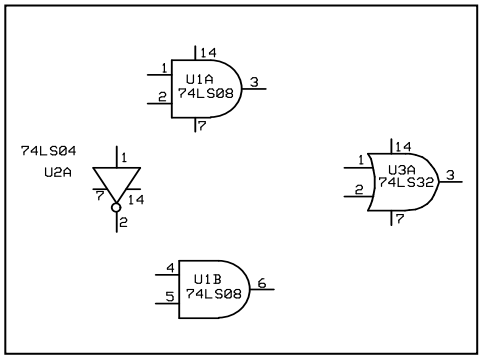
A note about blue color: to activate a symbol, click anywhere on its *outline*--not inside the symbol--to turn the outline blue. You can now drag it to another place. Or, if you double-click on any part of the outline, you open a window in which you can enter a label for the gate (a part ID).

For example: gates of the first chip you select should be labeled U1 and each gate should also receive a letter to distinguish it from the others. So the top NANDs ID should be U1A, the next down should be U1B, etc. The next chip you select should be labeled U2. If it also contains multiple identical elements, their IDs should be U2A, U2B, etc.

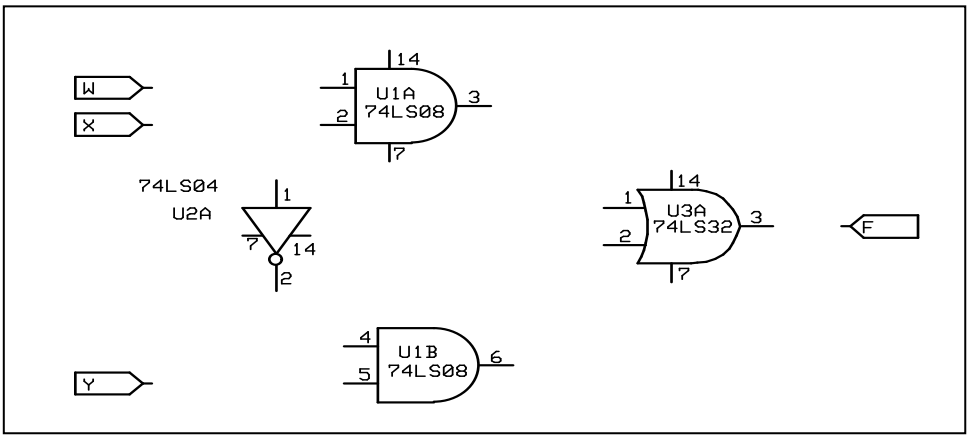
Suppose you want to build a circuit for the function **F = WX + X'Y**. You will need 2 AND gates (74LS08) one inverter (74LS04) and an OR gate (74LS32). Start by placing all 4 AND gates, 6 inverters, and 4 OR gates in the drawing area. Delete the last 2 ANDs, the last 3 ORs, and the last 5 inverters. The screen should look as shown below at the left.

The inverter has been rotated as shown (click the Rotate 90º button at the top until you get the orientation you want). Also, the gates have been moved into desired positions. Next, the ANDs will be labeled U1A and U1B, the inverter U2A, and the OR will be U3A. Just click twice on the outline of each gate to open the ID assignment window. The result is at the right. (The inverter's text has been dragged a bit to the left so as not to interfere with the wires, which will be added later.)





Next, circuit inputs and output (W, X, Y, and F) have to be placed on the screen. Right-click on the screen and open *Component manager*, but this time click on *Library Symbols.* For circuit inputs, select "Port-Right pointing-4 letters wide" and double-click. Move the port symbol to the left of AND pin 1. Double-click on the outline and enter symbol name W. Then right-click on the outline and select Copy. Paste the symbol to the left of pins 2 and 5 and name them X and Y. Return to the Library Symbols list, select "Port-Left pointing--4 letters wide", and place it to the right of OR gate pin 3. Enter F as its name. The result should appear as shown below:

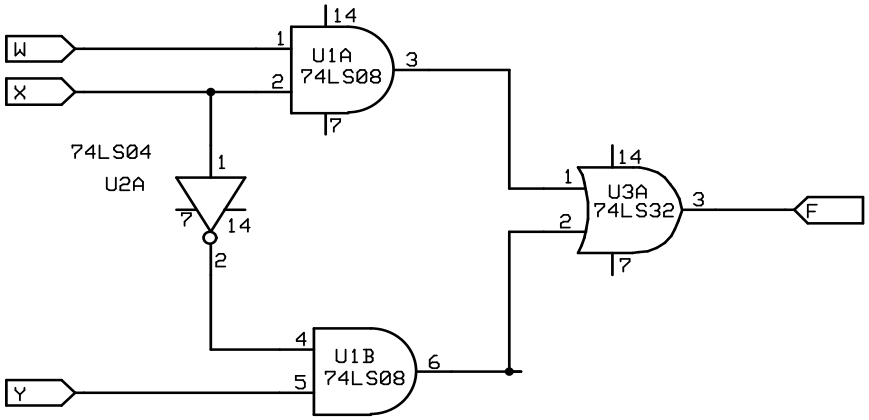


Lastly, you need to connect all these symbols with wires.

Notice that the arrow button at the top of the list of icons at the left is pressed. This is like the Esc key on a keyboard. To add wires, you have to press the wire button (5th one down), but when you're through, you need to press the arrow again to "escape" from the add-wire mode.

First, run a wire from W's port across to pin 1 (just click and drag across. At pin 1, left-click, and then right-click to release the mouse). Repeat from X's port to pin 2 and Y's port to pin 5.

Next, run a wire from pin 1 of the inverter up to the wire from X. Then add a wire from pin 2 of the inverter straight down, click to make a corner, and run it across to pin 4 of the lower AND. Finish up with wires from the two ANDs to the OR. Each wire requires two corners. Finally connect the OR's pin3 to the output port, F. The result is shown below. F = WX + X'Y



The equation at the top is optional. It is added by clicking the text button A--last one down on the left. A window opens at the top of the screen where the text can be entered. Text size can be changed from default value (not done in this case). Then just click on the screen to place the text.

Now that the diagram is complete, you will first want to save it as a **.sch** document so you will have it if you want to make changes later. Then paste it as a picture into a Word document for your report. Go to the Edit menu and select "Copy schematic image to clipboard". Then open a Word doc and paste. What you will see is the entire drawing sheet complete with the block at the lower right with Company Name, etc... The circuit itself will be too small, so you will need to edit the picture in Word.

Click on the picture and with the Picture Toolbar, select the Text Wrapping tool and choose Square. That will allow you to move the picture about the page. Use the Crop tool to eliminate unwanted sections of the picture; i.e. *everything* except the circuit itself. Then, stretch the picture across the page so that the circuit symbols are large and clear. Since you will print this out for inclusion in your report, you may want to change the color to Grayscale so as not to use your printer's color cartridge.

DeMorgan (DM) equivalent NAND and NOR symbols

A number of custom files that you will need in future experiments include those that contain alternative (DM) gate symbols.

Example: IC-74LS02-NOR gate-DM.s (compare with IC-74LS02-NOR gate-NORMAL.s).

In addition, there are others which were not supplied by ExpressPCB (7493, 7476) or which were somewhat altered to make them more useful in this lab (74155).

These are all ***.s*** files, an extension reserved for files with custom symbols. When you want to insert one of these symbols while drawing a circuit, open *Component Manager*. Then, click on *Custom Components* (or *Library Components* if in lab) and choose the gates you want. The DM symbols represent the complement of the usual algebraic definition of NAND and NOR.

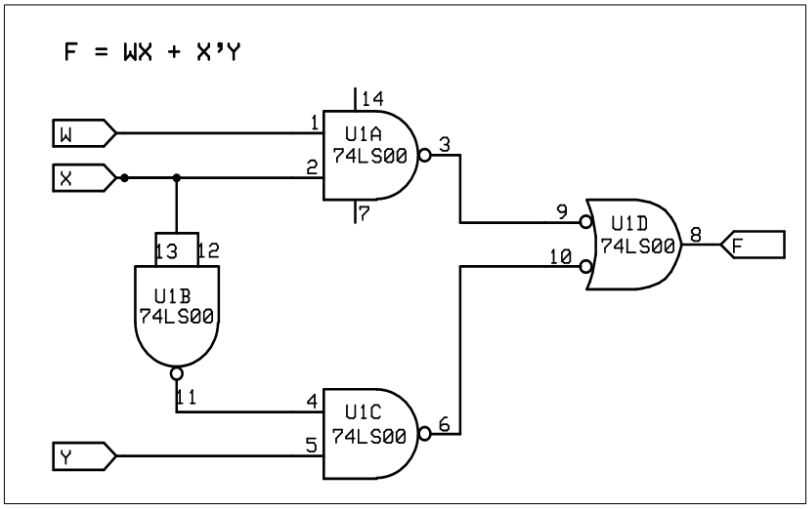
NAND: X = (YZ)' = Y'+Z' (an OR with inverted inputs replaces an AND with inverted output).
NOR: X = (Y+Z)' = Y'Z' (an AND with inverted inputs replaces an OR with inverted output).

The previous circuit for F = WX + X'Y was built with three chips (AND, NOT, and OR). Using NANDs and/or NORs instead can sometimes reduce the number of chips and, therefore, the physical size and cost of the circuit. In the following circuit, only one chip is required, a quad NAND 74LS00.

The other advantage of using DeMorgan symbols is that they help preserve the appearance of the original AND/OR circuit and thus makes it easy to follow the logic.

In this circuit, the eye sees inverter bubbles at each end of the wires into the final gate and can ignore them since they cancel each other out.

Suppose, instead, we used the normal NAND symbol for the output gate, U1D. The gate inputs are (WX)' and (X'Y)', so that F is [(WX)'•(X'Y)']'. This is the same as F = (WX) + (X'Y) but the logic is *much* harder to follow.

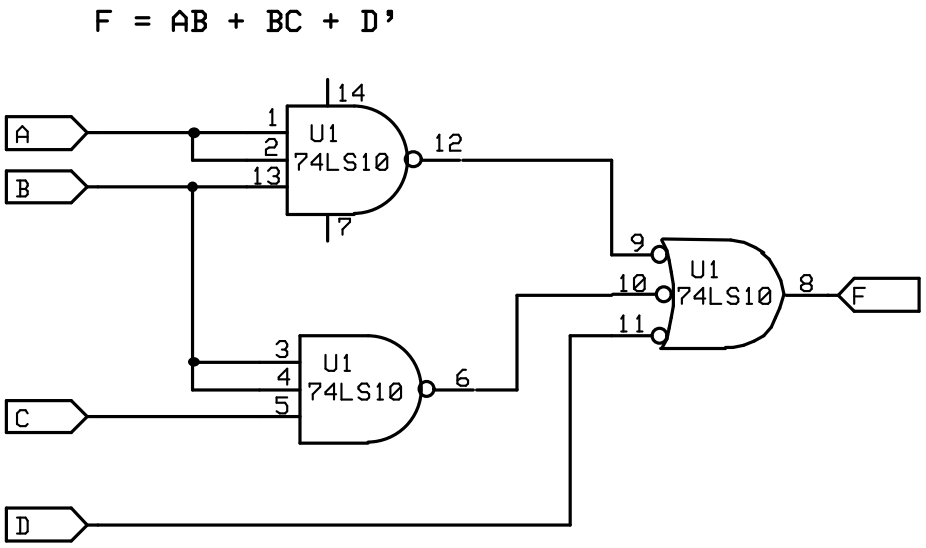


Finally, consider the function F = AB + BC + D'.

For an AND/OR circuit, you would need two 2-input ANDs, an inverter, and a 3-input OR. Since there are no 3-input OR chips, you would have to use two 2-input ORs, for a total of 5 gates from 3 chips.

Now, replace these gates with three 3-input NANDs (74LS10) from only one chip. One of the NANDs has inputs AAB (same as AB), another has inputs BBC (same as BC), and the last, substituting for the two ORs, has inputs from the other NANDs plus D.

Again, this is more compact than an AND/OR circuit, but by using the DM equivalent symbol for the last gate, it preserves the AND/OR appearance, which makes the logic so much easier to follow.

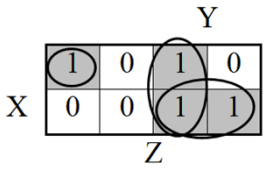


APPENDIX: TROUBLE-SHOOTING A CIRCUIT USING MAPS

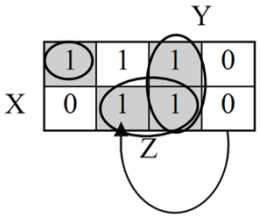
This appendix is designed to be used in situations where counter outputs are used to test a circuit by applying all possible input combinations in binary sequence. Assume the counter is triggered with a pulser output, where the circuit output is monitored with an LED.

Suppose you build a circuit for the function **F = X'Y'Z' + YZ + XY**.
Since a 3-input AND is not available, two 2-input ANDs are required for the first term; i.e. **(X'•Y') • Z'**. The last two terms require two more ANDs, so together with the two OR gates (+, +), a total of 6 gates are needed.

The map for F, derived from the equation, is at the right.



Now suppose that you build the circuit from the equation for F and drive it with outputs from the counter. You then record values for F as the counter goes through all 8 combinations in binary order. You insert F values into a map and get the result shown at right. Since the two maps don't match, there's an error in the circuit.

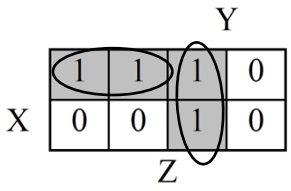


Naturally, you could check connections among all the gates to see what's wrong. But it may also be possible to find the error instead from the differences between the two maps.

Note that the 1 in square 6 of the original map has moved to square 5, so that F = X'Y'Z' + YZ + X**Z**.
Conclusion: XY in the original expression for F is replaced by XZ, implying that Z was mistakenly connected to the last AND gate (the last term in F) instead of Y.

(This method is fairly quick if only one incorrect connection is made to an AND gate, as in this case and the next.)

As another example, suppose your measured values for F lead to this map (the 1 in square 6 appears instead in square 2). By grouping the 1's as shown, this could be simplified as F = X'Y' + YZ. But that would imply a design of the form F = xx + xx (2 ANDs and 1 OR) which is not like the circuit you built where F = xxx + xx + xx.



So rather than over-simplify, try instead to maintain the *form* of the original equation if possible. Grouping the 1's as in the lower map, we get F = X'Y'Z' + YZ + **X'**Z.
Here the mistake was connecting X' to the last gate instead of X.