Chapter 5 Logic Design

Once logic expressions have been derived for each output of a system, the next step is to design the circuitry that implements those expressions. To do this we need to understand the fundamental building blocks of digital circuitry, *logic devices*, and how to combine them to design *combinational logic* circuits. The material in this chapter is applicable to any digital design, no matter what technology is eventually used to implement it.

5.1 Signals vs. Logic Expressions

Before we can discuss logic devices, it is necessary to relate signals to Boolean variables. (Signals also relate to logic expressions too, as we'll soon see.) In Chapter 1, we learned about digital signals and how they take on one of two values: high or low. In Chapter 4, we were introduced to Boolean variables and learned that they take on one of two values: 0 or 1. It is important to understand that digital signals can be used to represent Boolean variables, *but the two are not the same*. This distinction is blurred by many digital texts and HDLs, but it is crucial to understand the difference. What is at issue is whether a high signal represents a 1 or a 0, and there are three ways to make that assignment. They are called *positive logic*, *negative logic* and *mixed logic*.

Positive Logic

When positive logic is used, each high signal represents a 1 and each low signal represents a 0. That is to say that when a signal is high, the corresponding Boolean variable is 1 and when the signal is low, the corresponding Boolean variable is 0. With positive logic, signals and Boolean variables usually share the same name (which further blurs the distinction).

Positive logic is used almost exclusively when digital signals are used for data (in, for example, computers, memory, A/D converters, and digital signal processors).

Negaitve Logic

When negative logic is used, each high signal represents a 0 and each low signal represents a 1. In other words, if a signal is low, the corresponding Boolean variable is 1 and vice-versa. With negative logic, signal names either match the variable name or its complement, which can be very confusing sometimes. Negative logic is often used for digital control signals.

Mixed Logic

With mixed logic, each signal defines its own *assertion level* (the voltage level that corresponds to a logic 1). A signal that has a low assertion level is said to be *active low*. A signal that has a high assertion level is said to be *active high*.

Usually, mixed logic signals are labeled both by the Boolean variable (or expression) they represent and an assertion level. This notation has not been standardized. For example, if a signal represents variable X and is active low, it might be labeled X(L), X_L or X.L. If instead it is active high, it might be labeled X(H), X_H or X.H.

This text will use positive logic for data signals and mixed logic with the “dot” notation for everything else. The reason for the exception is that for data signals, positive logic is ubiquitous, and the advantages of mixed logic do not justify the overhead, however slight, of the notation.

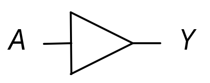
When using mixed logic, there are two signal identities that are important to know:

$$X.L = \bar{X}.H \tag{5.1}$$
$$X.H = \bar{X}.L \tag{5.2}$$

These identities allow a signal to be labeled two ways, one that is active high and the other that is active low.

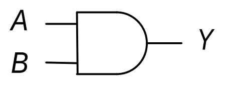
5.2 Logic Symbols

It is possible to design a digital circuit using individual transistors as shown in Figure 1-11, but such a design would be difficult and cumbersome both to create and to read. A better approach is to use *logic symbols*. Logic symbols are the schematic representations of *logic devices*. The inputs and outputs of logic devices are signals, not Boolean variables or logic expressions. Each device has a characteristic function that is defined by a function table. (A function table is similar to a truth table, except that the inputs and outputs are signals.) The symbols and function tables for the three basic logic devices is shown in Figure 5-1.



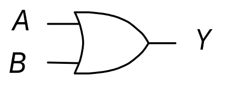
A	Y
L	L
H	H

(a) Buffer



A	B	Y
L	L	L
L	H	L
H	L	L
H	H	H

(b) And-Gate



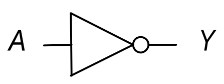
A	B	Y
L	L	L
L	H	H
H	L	H
H	H	H

(c) Or-Gate

Figure 5-1 Symbols and Function Tables for the Three Basic Logic Devices.

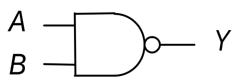
A *buffer* is a simple device whose output signal matches its input. We will learn more about buffers in Section TBD. The *and-gate* and the *or-gate* are a little more complicated. They may take any number of inputs. The output of the and-gate is high only if *every* input is high. The output of the or-gate is high if *any* input is high. Notice that the and-gate and the or-gate represent the Boolean operators **and** and **or** only if positive logic is used. If negative logic is used, the functions are reversed. (The student is encouraged to study the function tables and verify this is true.) This function reversal may seem confusing at first, but we will develop a technique that avoids this confusion by ensuring that the logic symbols always correspond to their Boolean operators.

There is no logic symbol for the **not** operator per se. Instead, a small circle or *bubble* is placed on the input or the output of a logic symbol to indicate an inversion. If we add a bubble to the output of each of the devices in Figure 5-1, we get a new set of devices shown in Figure 5-2.



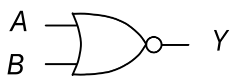
A	Y
L	H
H	L

(a) Inverter



A	B	Y
L	L	H
L	H	H
H	L	H
H	H	L

(b) Nand-Gate



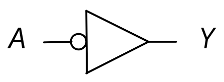
A	B	Y
L	L	H
L	H	L
H	L	L
H	H	L

(c) Nor-Gate

Figure 5-2 Symbols and Function Tables for Three Derived Logic Devices.

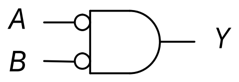
Notice that if the inputs use positive logic and the outputs use negative logic, these devices still perform the same Boolean operation they did in Figure 5-1.

Now suppose instead of adding the bubbles to the outputs we add them to the inputs. We get a new set of symbols, but an old set of function tables (Figure 5-3). We get old device names too, since a device (including its name) is defined by its function table.



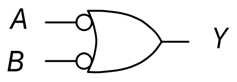
A	Y
L	H
H	L

(a) Inverter



A	B	Y
L	L	H
L	H	L
H	L	L
H	H	L

(b) Nor-Gate



A	B	Y
L	L	H
L	H	H
H	L	H
H	H	L

(c) Nand-Gate

Figure 5-3 Symbols and Function Tables for Three More Derived Logic Devices.

Again, if the outputs use positive logic and the inputs use negative logic, these devices still perform the same Boolean operation they did in Figure 5-1. The main principle illustrated here is this:

If active low signals connect to bubbles and active high signals do not, then the logic symbols for the and-gate and the or-gate correspond to the Boolean operators ***and*** *and* ***or****.*

We will hold to this rule whenever possible in this text. Usually, we can use Equations 5.1 and 5.2 to force a signal to have the right assertion level for each input or output.

5.3 Logic Circuit Design

Once all the inputs and outputs of a circuit are known, and a simplified logic expression that relates the inputs to the outputs has been obtained, we are in a position to design a digital circuit to implement the logic expression with logic devices.

There is a straightforward technique that we can use to implement any logic expression, as long as there are no complemented logic expressions (except, of course, individual variables). This simplification can always be accomplished with De Morgans's theorem. For example,

$$F = \overline{\overline{\bar{A}B} \cdot (\bar{A} + \bar{B} + C)} = \bar{A}B + \overline{(\bar{A} + \bar{B} + C)} = \bar{A}B + AB\bar{C}$$

The technique is outlined in the steps below.

1. Start with the output. If the output is active low, draw a bubble.
2. If the logic expression is the sum of *n* terms, draw an *n*-input *or-symbol* (the symbol used for an or-gate). If instead it is the product of *n* terms, draw and *n*-input *and-symbol*. You may choose whether or not to put bubbles on these *n* inputs.
3. Attach a wire (line) to each input and label it by its logic expression and assertion level.
4. Repeat steps 1-3 until all that remain are (possibly complemented) Boolean variables.
5. Re-label the inputs to remove any complemented inputs using Equations 5.1 and 5.2.
6. If the input signals that are needed have different assertion levels than those that are available, add inverters to produce the needed signals.

Consider a system with three input signals, *A.L*, *B.H* and *C.H*, and one output signal, *F.L*. Suppose the simplified logic expression relating the output to the inputs is the one we simplified earlier:

$$F = \bar{A}B + AB\bar{C}$$

The first step is to start with the output. Since it is active low, we draw a bubble, and since the expression is the sum of two terms, we draw an *or-symbol* with two inputs (Figure 5-4). Notice the *I/O Marker* surrounding *F.L*. Markers like this serve to indicate which signals in a circuit are inputs and outputs.

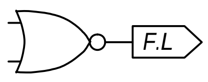


Figure 5-4 Output Gate for $F = \bar{A}B + AB\bar{C}$

Suppose we choose not to put bubbles on the inputs, which means that the inputs are active high. Next, we label each input with a term from the logic expression and its (high) assertion level. See Figure 5-5.

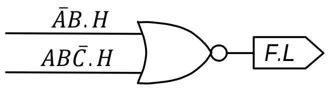


Figure 5-5 Output Gate with Labeled Inputs for $F = \bar{A}B + AB\bar{C}$

Now, we repeat the process for each of the inputs. They are both active high, so there are no bubbles, and both are products, so we will use and-symbols. The and-symbol for $\bar{A}B$ will need two inputs; the and-symbol for $AB\bar{C}$ will need three. This time, we choose to put bubbles on the inputs (Figure 5-6) which means that the inputs are active low.

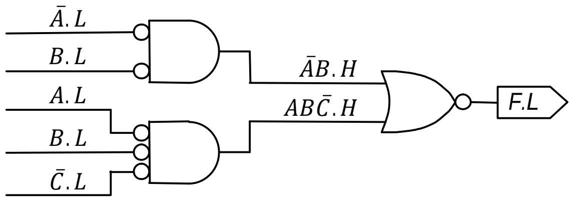


Figure 5-6 All Logic Devices to for $F = \bar{A}B + AB\bar{C}$

Notice that all the gates in this design so far are nor-gates. Experienced engineers favor nand- and nor-gates to and- and or-gates because they are usually smaller and faster.

The next step is to re-label the inputs using Equations 5.1 and 5.2, if necessary, to remove the complemented inputs. In this case, only $\bar{A}.L$ and $\bar{C}.L$ need to be re-labeled. Finally, for those signals that are not available with the correct assertion level (*A.H* and *B.L*), inverters must be provided. (SeeFigure 5-7.)

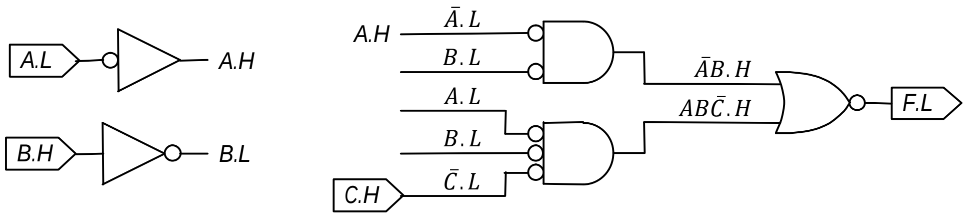


Figure 5-7 Logic Design for $F = \bar{A}B + AB\bar{C}$

Note that there are no explicit connections from the bank of inverters (or *rail*) on the left to the gates on the right. This is common practice because the connections, when they are shown, are often messy and confusing. It is often better to make these connections by giving all the wires that should be connected the same name.

Wires that connect together in a schematic are called a *net* or a *node*. Some computer *schematic capture* tools do not allow a net to be assigned two different names, and most have a difficulty with a net being labeled by an expression. To work around this problem, the labels may be omitted from the intermediate nets and the complemented input names may be discarded (Figure 5-8).

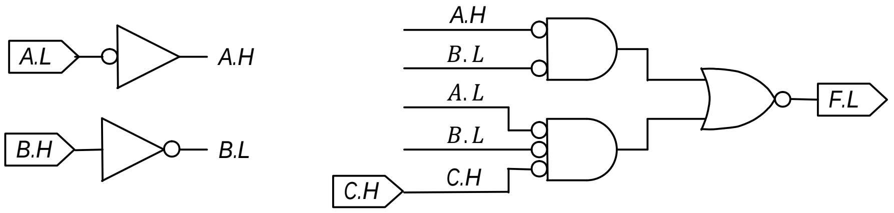


Figure 5-8 Implementation of $F = \bar{A}B + AB\bar{C}$ Suitable for Schematic Capture.

It is instructive to verify that this circuit generates the correct output. One verification technique is to make a combination truth and function table. First, list the Boolean input variables, then every signal in the circuit starting with the inputs and ending with the output(s). Use the output signal(s) to derive the output Boolean variable(s) and verify that the original logic expression has been implemented. For the circuit in Figure 5-7, the first half of the combination truth and function table is shown below. The second half is left as an exercise.

Table 5-1 Combination Truth and Function Table for Figure 5-7.

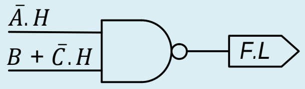
A	*B*	*C*	*A.L*	*B.H*	*C.H*	*A.H*	*B.L*	$\bar{A}B.H$	$AB\bar{C}.H$	*F.L*	*F*
0	0	0	H	L	L	L	H	L	L	H	0
0	0	1	H	L	H	L	H	L	L	H	0
0	1	0	H	H	L	L	L	H	L	L	1
0	1	1	H	H	H	L	L	H	L	L	1

Example 5.1 Design a logic circuit for the equation $F = \overline{A + \bar{B}C}$. Assume *A* and *C* are asserted *high* while *B* and *F* are asserted low. Use only nand and nor gates.

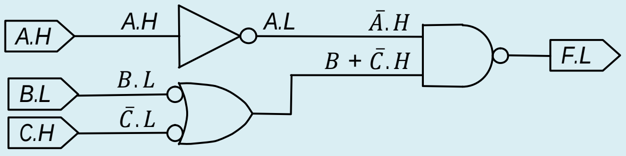
Solution: Before we can begin, we must simplify the equation so that only the variables are complemented:

$$F = \overline{A + \bar{B}C} = \bar{A} \cdot \overline{\bar{B}C} = \bar{A}(B + \bar{C})$$

The output is active low, so we draw a bubble. The output is the product of two factors, $\bar{A}$ and $(B + \bar{C})$, so we draw the symbol for a 2-input and-gate. We are constrained to use only nand- or nor-gates, so bubbles must not be added to the inputs:



We repeat the process for $(B + \bar{C})$ and re-label complemented inputs, $\bar{A}.H = A.L$ and $\bar{C}.L = C.H$. The signal A.L is not available, so we add an inverter to produce it.



If we wish, we can verify the circuit in Example 5-1 implements the logic equation $F = \bar{A}(B + \bar{C})$ with Table 5-2, below.

Table 5-2 Combination Truth and Function Table for Example 5-1.

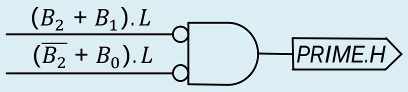
A	*B*	*C*	*A.H*	*B.L*	*C.H*	*A.L*	$B+\bar{C}.H$	*F.L*	*F*	$\bar{A}(B+\bar{C})$
0	0	0	L	H	L	H	H	L	1	1
0	0	1	L	H	H	H	L	H	0	0
0	1	0	L	L	L	H	H	L	1	1
0	1	1	L	L	H	H	H	L	1	1
1	0	0	H	H	L	L	H	H	0	0
1	0	1	H	H	H	L	L	H	0	0
1	1	0	H	L	L	L	H	H	0	0
1	1	1	H	L	H	L	H	H	0	0

Example 5.2: Design a logic circuit for the logic equation derived in Example 4.5 using only nand- or nor-gates. Assume all inputs and outputs are asserted high.

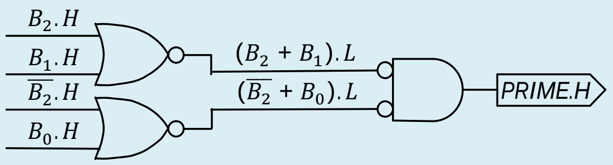
Solution: From Example 4.5, *PRIME* = $(B_2 + B_1)(\overline{B_2} + B_0)$

Based on the given assertion levels, the input signals are: *B₀.H, B₁.H, B₂.H,* and the output signal is *PRIME.H*

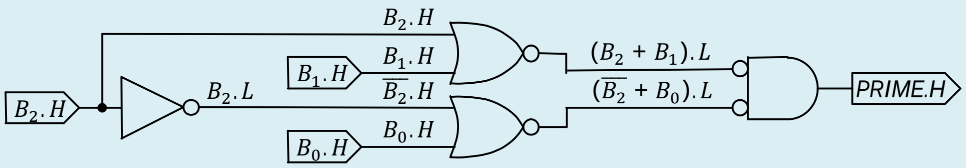
The output is active high, so we do not draw a bubble. The output is the product of two factors, $(B_2 + B_1)$ and $(\overline{B_2} + B_0)$, so we draw the symbol for a 2-input and-gate. We are constrained to use only nand- or nor-gates, so bubbles must be added to the inputs:



Repeating the process for $(B_2 + B_1)$ and $(\overline{B_2} + B_0)$, we have:



Only the $\overline{B_2}.H$ signal needs to be re-labeled, and it is also the only one that needs an inverter. Rather than connect the inverter to the gate by naming the nets, this circuit is simple enough to make explicit connections:



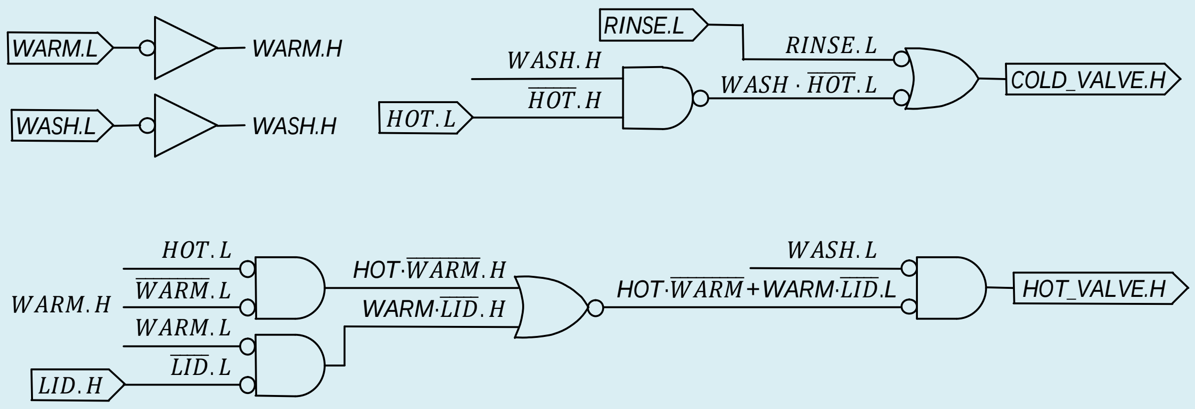
Example 5.3. Design a logic circuit for the water valve controller in Section 4.1. Assume that the HOT, WARM, WASH and RINSE signals are connected as shown in Figure 3-2 and are therefore active low. The lid switch is also connected in the same way, but the switch is normally closed, so LID is active high. The outputs HOT_VALVE and COLD_VALVE both drive solid state relays as shown in Figure 3-11b, so they are also active high.

Solution. In this case, there are two outputs, two equations and two logic circuits. (They may share the same input rail, however.) From Section 4.1, Equation 4.3 and Example 4-6 tell us:

COLD_VALVE = *RINSE* + *WASH* · $\overline{HOT}$, and
HOT_VALVE = *WASH* · (*HOT*·$\overline{WARM}$ + *WARM*·$\overline{LID}$)

Based on the given assertion levels, the input signals are *HOT.L, WARM.L, WASH.L, RINSE.L* and *LID.H*. Likewise, the output signals are *COLD_VALVE.H* and *HOT_VALVE.H.*

Using the design technique presented in this chapter, the logic circuit is shown below. Note that the inverter rail needs to generate *WASH.H* because that signal is needed to generate *COLD_VALVE.H,* and it needs to generate *WARM.H* because that signal is needed for *HOT_VALVE.H*.



Exercises

1. Write (a) a combination truth and function table for the devices in Figure 5-1 assuming input *A* is active low. (b) Write a similar table for the devices in Figure 5-3 assuming input *A* is active high. (c) What conclusion can you draw from this exercise?
2. Suppose the gates in Figure 5-1 have bubbles on both their inputs and outputs. (a) Write the functions tables for all three devices. (b) For each of these three devices tell whether the function table defines a new device or one that has already been presented in this chapter. If it has already been presented, give its s name.
3. Complete Table 5-1.
4. Design a logic circuit for $F = AC + \bar{A}\bar{B}$ assuming *A*, *B*, *C* and *F* are asserted high.
5. Repeat problem 4 assuming *A*, *B* and *C* are asserted high and *F* is asserted low.

6. Design a logic circuit for $F = (\overline{AD} + B)(\bar{B} + C)\left(\overline{AC\bar{D}}\right)$ assuming *A*, *B*, *C* and *F* are asserted high and D is asserted low. Remember to simplify first.
7. Repeat problem 6 assuming all inputs and outputs are asserted low.
8. Design a logic circuit for the logic equation in Equation 4.25 using only nand- or nor-gates. Assume all inputs are asserted high and the output, PRIME, is asserted low.
9. The circuit in Example 5-3 does not make use of the fact that WARM and HOT can never be true at the same time. Can we use this fact to get a simpler circuit? If so, what has to change and how many logic devices can we save?