

CIRCUIT CELLAR®

THE MAGAZINE FOR COMPUTER APPLICATIONS

CONSIDERING THE DETAILS

Bob Perrin

Mixed-Logic Notation

A Tool for Concise Expression

In these days of VHDL and Verilog, universities may be short-changing their students by not teaching them how to schematically communicate digital logic. Get out your notebooks as Bob explains how important mixed-logic notation is and how easy it is to learn and use.



Recently, I've come across a number of engineers who have trouble clearly expressing logic equations as part of a schematic. Although most of these folks have only been out of school for a few years, some have been engineers for many years but are primarily self-taught and don't have a degree in electrical engineering.

Upon investigating one case, I found that the textbook used at my colleague's university settled on a positive-logic system [1]. Although I'm sure the author felt this approach was less confusing, all he accomplished was dumbing down the material and short-changing the student. Homologic is the digital equivalent of training wheels—suitable for beginners but cumbersome in the long run.

Mixed-logic notation is a method for expressing logical equations in schematic form. It takes advantage of DeMorgan's theorems to enable engineers to express the Boolean equations that govern the system's operation clearly and concisely on the schematic.

Schematics are often thought of simply as graphical net lists used by our autorouters while they generate Gerber files for PCBs. Some "forward-looking" engineers consider schematics a temporary encumbrance, an anachronism to be tolerated until the world migrates entirely to VHDL.

Well, for now at least, schematics are the primary medium for communicating circuit topology in most areas outside of VLSI design. The schematic that is created today will be referred to countless times by other engineers, technicians, customers, and (God help us) managers.

With this in mind, I decided that a short explanation of the mixed-logic system might be useful to have around. This article is not a review of Boolean algebra; it simply explains how to create logic schematics that are effective communication tools.

Given the constraints of HTML, I can't use the overbar (indicating "complement of") in this article. Therefore, when you see a word or variable preceded by an asterisk in this article, you should interpret it the same as you would a bar completely across the word or variable.

PHYSICAL VS. LOGICAL TRUTH TABLES

In physical systems, voltage levels are where the rubber hits the road. Binary systems have two legitimate voltages—HIGH and LOW. In a 5-V system, HIGH means 5 V. In a 3-V system, HIGH means 3 V. For our purposes, LOW is taken to mean 0 V.

Physical devices are described by physical truth tables. A physical truth table tells the observer what voltage behavior to expect from a device. A physical truth table may have multiple logical interpretations, depending on how we decide to interpret the HIGH and LOW voltages in the circuit.

HIGH and LOW are meaningless in a Boolean sense—they are voltages, not mathematical entities. To say a HIGH has any implicit Boolean implication is incorrect, at least in the mixed-logic system.

Before HIGH and LOW in a physical system can be related to the Boolean equations governing the system operation, we must agree how to interpret HIGH and LOW in our system.

One homologic approach would be to decide a HIGH is always a 1, and a LOW is always a 0. This approach is referred to as positive logic. Another homologic approach would be to decide a HIGH is always a 0, and a LOW is always a 1, which is referred to as negative logic.

The mixed-logic approach says a HIGH can be a 0 or a 1, depending on the notation we use. A LOW is interpreted as the opposite of HIGH. We are free to mix and match interpretations as long as we are consistent with the notation.

So, let's agree that a signal name suffixed with "(H)" means a HIGH is interpreted as 1. We'll refer to such a signal as active high. Likewise, let's agree that a signal name suffixed with "(L)" means a HIGH is interpreted as 0, and we'll refer to it as active low.

Other texts use other syntax, such as the suffixes ".H" and ".L", for active high and active low. Another common

convention used for naming nets in CAD software is to suffix the net name with "_H" or "_L" to designate the activation level. These names are referred to as polarized mnemonics because the activation level of the signal is integrated with the name.

Another common practice is the use of a bar above a signal name to indicate active low. This is a bad way to designate active low because the bar above a signal means "perform the NOT operation on this signal." This is a Boolean operation, not a physical interpretation.

Boolean algebra is a mathematical abstraction, a way to model the physical world. To create concise documentation, Boolean abstractions must not be mixed with naming conventions. The abstraction is used to concisely express the logical operation of the system. Naming conventions only tell the reader how to interpret voltage levels.

As an example, consider *RESET. A common misuse of this symbol is to designate a RESET line as active low. As a Boolean variable, *RESET means "if you do not want the system initialized, assert this signal." This meaning is consistent if you are using a strictly positive-logic system.

In a positive-logic system, all signals are active high so if you assert *RESET, you are really forcing it to a HIGH voltage. If the physical system interprets the *RESET as active low, then by asserting

this line (pulling it HIGH) you are NOT resetting the system. This is clearly a convoluted method for naming signals.

A clearer way is to designate the system's active low reset signal as RESET (L). This says, "if you want the system initialized, assert this signal." And the (L) tells us clearly that the physical system interprets the signal as active low. This method provides an unambiguous, succinct way to communicate what the signal does.

The signals may only assume values of asserted (Boolean 1 or TRUE) or deasserted (Boolean 0 or FALSE). The polarized mnemonics simply tell what voltage levels correspond to the two states.

The polarized mnemonics are useful for helping readers of your schematic deduce the Boolean logic equations expressed graphically (through gates and wires) on your schematic. However, when writing the Boolean equations governing the operation of the system, it is customary and natural to drop the (L) and (H) from the signal names. The (L) and (H) have meaning on a schematic (a graphical communication tool that explains the operation of a physical circuit) but no real value in a Boolean logic equation.

LOGICAL INTERPRETATION OF PHYSICAL TRUTH TABLES

Now that we have a way to express activation level in a signal name, all that's left is to identify activation level as part of a schematic symbol. This is done universally with bubbles. A bubble on the input or output of a gate means the signal is active low.

As I mentioned, a physical device is described with a physical truth table. A physical device is simply a collection of transistors and has no implicit Boolean interpretation. Figure 1a shows a physical truth table for the physical device found in a 7404 (hex inverter).

Because there are two physical signals, there are four possible logical interpretations of the physical interpretations of the physical device. Bet you never knew the 7404 was so versatile. Figures 1b–e show the four possible logical inter-

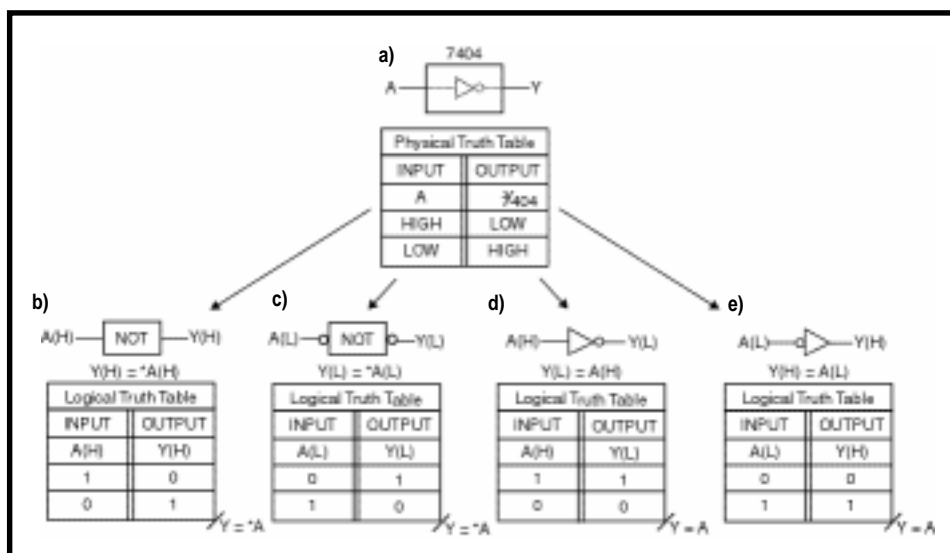


Figure 1a—A physical truth table describes the voltage behavior of a device. **b**—This logical truth table implies a logical NOT operation. **c**—This logical truth table implies a logical NOT operation. **d**—This interpretation of the physical truth table is an active high to active low converter. **e**—This interpretation of the physical truth table is an active low to active high converter.

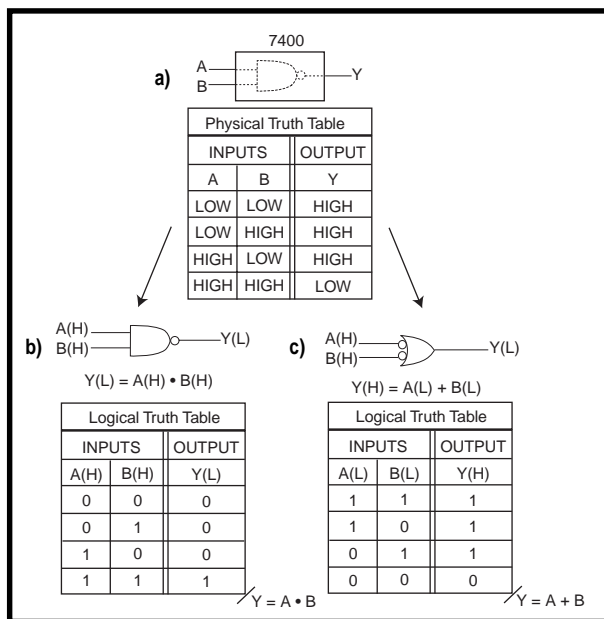


Figure 2a—A physical truth table describes the voltage behavior of a device. **b**—This logical truth table implies an AND operation. **c**—This logical truth table implies an OR operation.

pretations of Figure 1a.

Figure 1b shows the logical truth table if we interpret the input and output of the gate to be active high. Notice the familiar homologic $Y(H) = *A(H)$ truth table.

Figure 1c interprets all signals as active low. This is the equivalent of a negative-logic interpretation. As expected, this yields the familiar homologic $Y(L) = *A(L)$ truth table.

Figure 1d interprets the input A as active high, while interpreting output Y as active low. This interpretation yields the logic equation $Y(L) = A(H)$. Notice the schematic symbol representation is the familiar triangle with a bubble on the output.

Figure 1e interprets the input A as active low, whereas output Y is interpreted as active high. Notice the schematic symbol. Again, we have a logic-level converter.

The name “inverter” is a misnomer for the logical interpretations and schematic representations shown in Figures 1d and 1e. In Figures 1d and 1e, the Boolean value of the output follows that of the input $Y = A$. Clearly, no Boolean inversion occurs in this logical interpretation. However, “inverter” is a suitable name for the interpretations in Figure 1b and 1c.

Again, notice there is only one physical truth table, but we have four logical

interpretations (truth tables). This is paramount to understanding the mixed-logic system.

Physical devices are described by physical truth tables. Each physical truth table may be interpreted arbitrarily by the engineer and represented as multiple logical truth tables, and therefore the physical part will have multiple graphical representations.

Figure 2a shows a physical truth table for a gate from a 7400 (quad dual-input NAND gate). There are $2^3 = 8$ possible logical interpretations of this physical truth table.

However, only two are in common use (see Figures 2b and 2c).

The other six do not have a convenient representation in our graphical lexicon.

The logical truth table in Figure 2b interprets the inputs A and B as active high and the output Y as active low. Looking at the logical truth table, we deduce $Y(L) = A(H) \cdot B(H)$. The \cdot represents the Boolean AND function.

From the logic equation deduced from the logical truth table in Figure 2b, we draw an AND gate (flat bar with round nose), then we add bubbles to all active-low signals. The inputs are active high (no bubbles). The output is active low (add a bubble).

The logical truth table in Figure 2c interprets the inputs A and B as active low, and the output Y as active high. From the logical truth table we deduce $Y(H) = A(L) + B(L)$. The $+$ is used for the Boolean OR function.

From the logic equation we draw an OR gate and add bubbles to all the active low signals. The inputs are active low (add bubbles). The output is active high (no bubble).

Once again there is

only one physical truth table, but we use two logical of 8 possible interpretations. Notice that I’ve avoided the name “NAND” for either interpretation because it would be incorrect.

NAND is a homologic name for a positive-logic interpretation. You can clearly see from the truth table in Figure 2b that the logical interpretation for the symbol shown is an AND function.

In a purely homologic system, bubbles mean “take the complement of.” In a mixed-logic system, the bubbles simply mean active low.

The schematic symbol of Figure 2b is properly called a “two-input AND function with active-high inputs and an active-low output.” This is quite a mouthful indeed, but is logical and unambiguous. If you call Digi-Key and order a “two-input AND function with active-high inputs and an active-low output,” the clerk will just say, “Sorry, I don’t think we carry those—but can I interest you in some NAND gates?”

We are stuck for the foreseeable future with the legacy homologic names for physical devices (e.g., the physical chip 7400 is called a NAND gate). But when reading and drafting schematics, use and think the longer logical name.

Figure 2c is the best example thus far of what a poor name NAND is for the 7400 gate. The logical interpretation is that of a “two-input OR function with

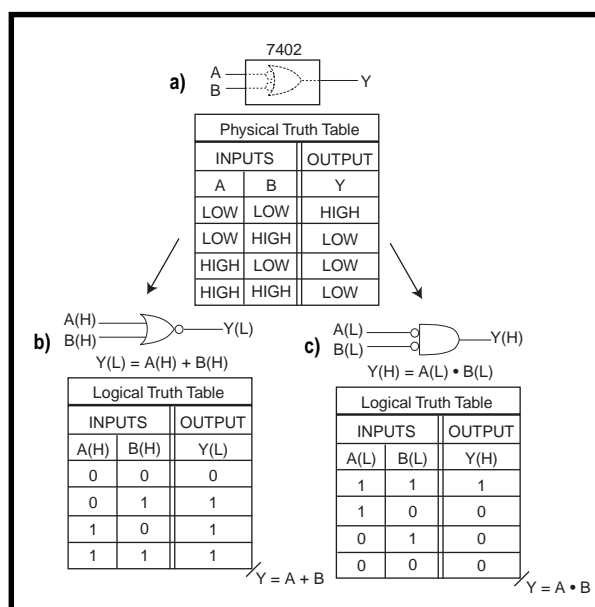


Figure 3a—A physical truth table describes the voltage behavior of a device. **b**—This logical truth table implies an OR operation. **c**—This logical truth table implies an AND operation.

active-low inputs and an active-high output.” Clearly there is no AND (or NAND) operation involved with this interpretation.

The physical device on a 7400 can be drawn equally correctly as is shown in Figure 2b and 2c. How you put it on your schematic depends on the Boolean function you intend (AND or OR) and the signal assertion levels (i.e., active high or active low).

Figure 3a shows the physical truth table the device found in a 7402. Again, we have $2^3 = 8$ possible logical interpretations. And again, there are only two in use (see Figures 3b and 3c).

The logical truth table in Figure 3b is one common interpretation of the physical truth table in Figure 3a. The inputs, A and B, are assumed active high and the output Y is assumed active low. The logical truth table indicates an OR function is performed on A and B to get Y. The schematic symbol for an OR function is drawn, and because Y was assumed low, a bubble is added.

The logical truth table in Figure 3c is the other common interpretation of the physical behavior described in Figure 3a. Inputs A and B are assumed active low and output Y is assumed active high.

The logical truth table indicates that an AND function is performed on A and B to get Y. The schematic symbol for an AND function is drawn. The inputs were assumed active low, so bubbles are added to the symbol. The output was assumed active high, so no bubble is added.

The physical device in a 7402 can be drawn equally well as an AND or an OR function, assuming we place the bubbles in the right spots. Which symbol you use depends only on the Boolean function you wish to convey.

Figure 4a shows the physical truth table for the gate found in a 7408. Like the 7400 and 7402, there are $2^3 = 8$ possible interpretations of the data found in Figure 4a. Again, only two are used (see Figures 4b and 4c).

The logical truth table in Figure 4b is one common interpretation of the physical truth table in Figure 4a. All the inputs and the output are assumed active high. The logical truth table clearly shows that $Y(H) = A(H) \cdot B(H)$. To

most engineers, this certainly should be a familiar result.

Deriving the symbol in Figure 4b is trivial. The logical truth table tells us the Boolean function is AND. And we know from the assumptions that were used to create the truth table (namely, that all signals are active high) that we will not have to add any bubbles to the schematic symbol.

The logical truth table in Figure 4c is the second common interpretation of the physical behavior described in the physical truth table of Figure 4a. All the inputs and the output are assumed active low. The truth table reveals that the Boolean operation associated with this interpretation is OR. The schematic symbol is derived by combining the symbol for an OR operation with bubbles to designate the active low inputs and output.

The last physical part we need to round out this discussion of logic symbols is the physical device found in the 7432. Figure 5a has a physical truth table that describes the behavior of this device. As before, we have eight possible interpretations, with only two making much sense to use.

The logical truth table in Figure 5b makes the assumption that all the signals on the gate are active high, which leads us to the conclusion that $Y(H) = A(H) + B(H)$. The schematic symbol is simply an OR gate schematic with two inputs and one output. No bubbles are required because none of the signals are taken to be active low.

The logical truth table in Figure 5c interprets the information from the physical truth table in Figure 5a as uniformly active low. The logical truth table shows us that

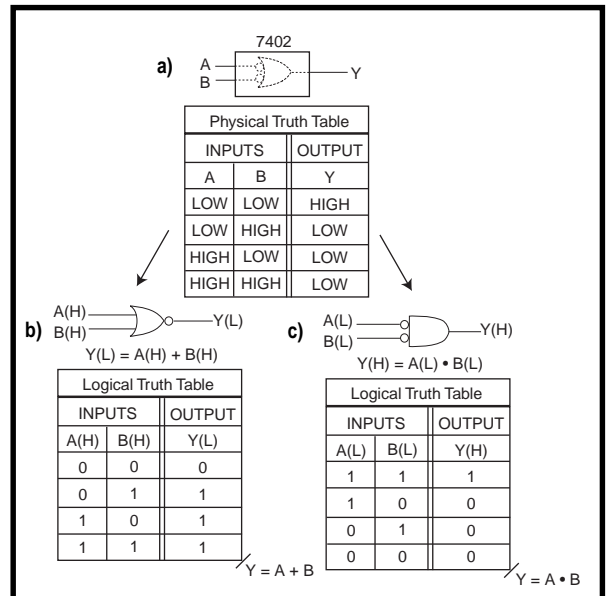


Figure 4a—A physical truth table describes the voltage behavior of a device. b—This logical truth table implies an AND operation. c—This logical truth table implies an OR operation.

an AND operation is being performed on the two active-low inputs yielding the active-low output. The schematic is therefore an AND symbol with bubbles on every signal.

I have just covered five examples of how to use physical truth tables to describe behavior, the use of logical truth tables to interpret the physical behavior in a Boolean sense, and how to create a schematic symbol that depicts the Boolean functionality associated with the logical interpretation. Keep these principles in mind and you're 95% of the

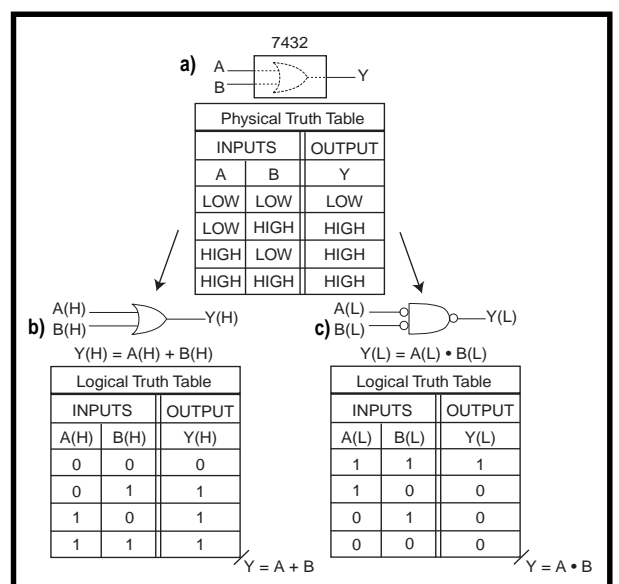


Figure 5a—A physical truth table describes the voltage behavior of a device. b—This logical truth table implies an OR operation. c—This logical truth table implies an AND operation.

way to understanding the mixed-logic system.

Notice that Figures 1d and 1e, 2b and 2c, 3b and 3c, 4b and 4c, and 5b and 5c are duals. This duality should not come as a surprise. DeMorgan's Laws tell us to expect exactly this type of behavior.

As the dual representations of physical devices are so closely tied to DeMorgan's Laws, these alternate logical interpretations of physical devices are usually called DeMorgan equivalents. Even most schematic capture tools support DeMorgan equivalent gates for ease of concise drafting.

LOGIC INCOMPATIBILITIES

Figures 2–5 give us AND and OR gates in all the flavors we could possibly need. But what about NOT? Figures 1d and 1e are shown as logic-level converters. And Figures 1b and 1c show unconventional schematic symbols. How do we affect a Boolean NOT operation?

The answer lies in “logic incompatibilities.” A logic incompatibility exists

when an input to a device and the input requirement of the device are of opposite activation levels.

For example, if an active-high signal is connected to a device that has an active-low input, we say a logic incompatibility exists. Likewise, if an active-low signal is connected to a device's active-high input, a logic incompatibility exists.

From our definition, logic incompatibilities never exist at the output of a gate. All outputs are taken to be at the assertion level indicated by the gate's schematic symbol. Logic incompatibilities only occur at the inputs of gates.

When a logic incompatibility exists at the input of a gate, we put a small triangular flag (point down) near the input of the gate. This flag serves two purposes: it alerts readers that a logic incompatibility exists, and it tells readers that you mean for the incompatibility to exist. It acts as a sort of explicit type cast, if you will. The reader is being told that the

signal and gate-assertion levels are different so the signal name (a Boolean variable) will be complemented in the output.

A logic incompatibility causes the incompatible signal's complement to show up in the device's output. Consider Figure 6a. The signal A(H) is connected to the active low input of the OR gate. This logic incompatibility causes the Boolean expression at the output to be $Y(H) = *A(H) + B(L)$, or more simply, $Y = *A + B$.

Notice that Y(H), A(H), and B(L) are polarized mnemonics, and as such correspond directly to the Boolean variables Y, A, and B. Y(H), A(H), and B(L) (or Y, A, and B) are Boolean interpretations of the physical voltages in the circuit. This means Y(H), A(H), and B(L) (or Y, A, and B) can only take on Boolean values of 1 and 0, not voltage levels of HIGH or LOW.

Consider the circuit shown in 6b. Here we have the same

incompatibility as in Figure 6a. The fact that the output of the OR gate is active low is irrelevant to the interpretation of the equation. The circuit yields $Y(L) = *A(H) + B(H)$, or more simply, $Y = *A + B$.

The circuit shown in Figure 6c has a logic incompatibility where B(L) enters the OR gate's active-high input. This means B(L) will show up in the output complemented. The circuit yields $Y(H) = A(H) + *B(L)$, or more simply, $Y = A + *B$.

The circuit in Figure 6d has the same logic incompatibility as in Figure 6c. The fact that the output of the OR gate is active low is irrelevant to the interpretation of the equation. The circuit yields $Y(L) = A(H) + *B(L)$, or more simply, $Y = A + *B$.

Notice that the circuits in Figures 6a and 6b both have the same logical interpretation. The only difference is the activation level of the output signal. A similar situation exists for the circuits shown in Figures 6c and 6d. Figures 6e–h show the duals of the circuits shown in Figures 6a–d.

In Figures 6e and 6f, the logic incompatibility exists where an active low B (i.e., B(L)) is connected to the AND gate's active-high input. This causes B(L) to show up in the output complemented. The circuits in Figures 6e and 6f perform the same logical operation on A(H) and B(L), $Y = A \cdot *B$. The only difference is the assertion level of the AND gate's output—low in 6e, high in 6f.

In Figures 6g and 6h, the logic incompatibility exists where the active high A (A(H)) is connected to the AND gate's active-low input. This causes A(H) to show up in the output complemented.

These circuits perform the same logical operation on A(H) and B(L), $Y = *A \cdot B$. The only difference is the assertion level of the AND gate's output—low in 6g, high in 6h.

Beginning to see a pattern? The last piece of the puzzle to cover is the logic-level converter. Although “inverter” is somewhat of a misnomer, I will use that name when referring to the physical device. I do this primarily because the word “inverter” is so widely used (or misused) to describe the physical device

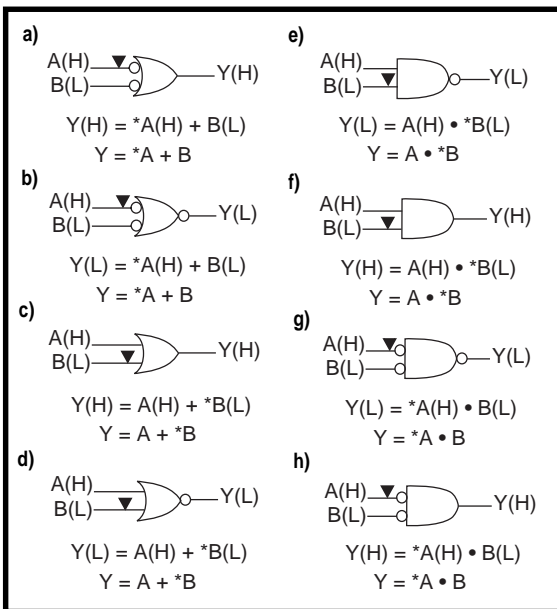


Figure 6a—Logic incompatibilities at a gate's input mean that variable is complemented in the output $Y = *A + B$. **b**—The activation level of an output, low in this case, has no effect on how the Boolean expression is derived. $Y = *A + B$ is the same expression derived in 6a. **c**—Logic incompatibilities at a gate's input mean that variable is complemented in the output $Y = A + *B$. **d**—The activation level of an output, low in this case, has no effect on how the Boolean expression is derived. $Y = A + *B$ is the same expression that was derived in 6c. **e**—Logic incompatibilities at a gate's input mean that variable is complemented in the output $Y = A \cdot *B$. **f**—The activation level of an output, high in this case, has no effect on how the Boolean expression is derived. $Y = A \cdot *B$ is the same expression that was derived in 6e. **g**—Logic incompatibilities at a gate's input mean that variable is complemented in the output $Y = *A \cdot B$. **h**—The activation level of an output, high in this case, has no effect on how the Boolean expression is derived. $Y = *A \cdot B$ is the same expression that was derived in 6g.

found in a 7404.

I will use “logic-level converter” when I explicitly mean that Boolean interpretation of the physical device. I will also point out when “inverter” is an appropriate name for a Boolean interpretation.

Figure 7 shows the possible configurations for an inverter. The circuit shown in 7a is a logic level converter. $Y(L) = A(H)$, or $Y = A$ means if $A(H)$ is a 0 then $Y(L)$ will be a 0. If $A(H)$ is a 1 then $Y(L)$ will be a 1.

The circuit in 7b is also a logic-level converter. Again $Y = A$, but this time, Y is active high and A is active low.

Figures 7c and 7d show an inverter interpretation of the physical device referred to as an inverter. The signal-naming conventions indicate there is a logic incompatibility at the input of the gate. This means the signal A shows up in the output complemented, $Y = *A$.

Generally, we use the physical device as a logic-level converter to introduce a logic incompatibility subsequent to the logic-level converter. For example, in Figure 8a, $A(H)$ is converted to $A(L)$ with the logic level converter (U1). Remember, $A(H)$ and $A(L)$ have the same Boolean value (see Figure 1d for a truth table). U1 presenting an active low $A(L)$ to U2 causes a logic incompatibility at the input of U2. This is shown explicitly by the flag at the input of U2, which means $A(L)$ will show up in the output of U2 complemented. Thus $Y(H) = *A(L) \cdot B(H)$, or $Y = *A \cdot B$.

Figures 8b and 8c show slightly more interesting examples. In Figure 8b, there is a logic incompatibility introduced by the logic-level converter in the signal path of A . $A(H)$ is an input to the circuit. $A(L)$ is produced by the logic-level converter and presented to the active-high input of the OR gate, which means A shows up complemented in the output of the OR gate.

$D(H)$ is simply $B(H) \cdot C(H)$, or $D = B \cdot C$. Because $D(H)$ is presented to the active-high input of the OR gate, there is no logic incompatibility, so the output of the OR gate does not have D complemented. Thus, $Y = *A + D$. But because $D = B \cdot C$, we get $Y = *A + B \cdot C$.

Figure 8c depicts a circuit almost identical to that shown in Figure 8b. However, the output of the AND gate is

active low. This introduces a logic incompatibility at the active-high input of the OR gate, and D shows up in Y as being complemented, $Y = *A + *D$. Because $D = B \cdot C$, we get $Y = *A + *(B \cdot C)$. Using DeMorgan’s Laws, we can further reduce this to $Y = *A + *B + *C$.

Don’t be confused by all the (H) and (L) designations. These are part of the variable name and are meaningless once we actually write an equation. The polarized mnemonics are only used to tell the reader how to interpret the voltage level of signals in the physical circuit. Once we write an equation such as $Y(H) = *A(H) + *(B(H) \cdot C(H))$ in Figure 8c, the (H) and (L) designations can be dropped and just the signal name (Y , A , B , and C) used.

The polarized mnemonic is a variable that assumes a Boolean value of 1 (asserted) or 0 (deasserted). Once the logic incompatibilities have been identified and an equation written, the (L) and (H) become superfluous.

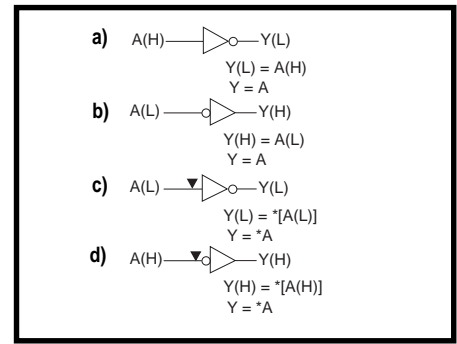


Figure 7a—The output of a logic-level converter assumes the same Boolean value as the input. However, the output is of the opposite assertion level than the input. **b**—When $A = 1$, $Y = 1$. When $A = 0$, $Y = 0$. The output follows the input. **c**—The logic incompatibility at the input of the logic-level converter causes the output to assume the opposite Boolean state of the input. **d**—When $A = 0$, $Y = 1$. When $A = 1$, $Y = 0$. The logic incompatibility causes a complement to be formed in the output.

Figures 8d and 8e both introduce a logic incompatibility where $C(L)$ enters the active high input of the AND gate. This means C shows up complemented in the output of the AND gate, $D = B \cdot *C$.

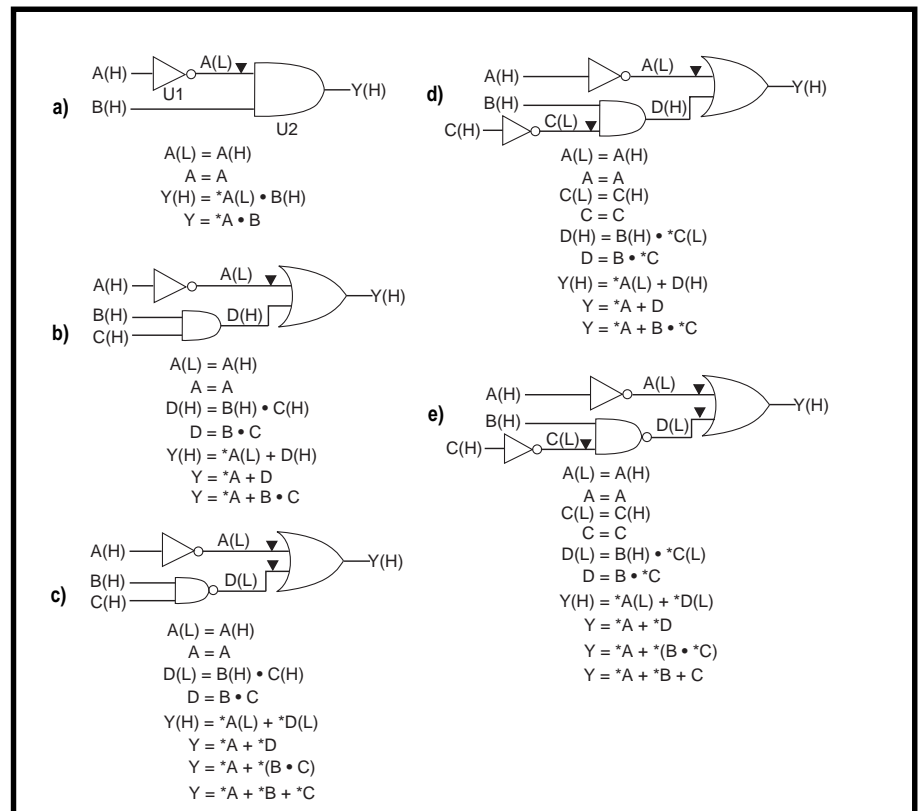


Figure 8a—The logic incompatibility at the input of the AND gate causes A to be complemented in Y . U1 is a logic-level converter. **b**—The logic-level converter is used to introduce a logic incompatibility at the input of the OR gate in A ’s signal path. A will be complemented in the OR gate’s output. **c**—Both $A(L)$ and $D(L)$ have incompatible activation levels with the input of the OR gate. Both A and D will show up complemented in Y . **d**—Two logic-level converters are used to introduce two logic incompatibilities in this circuit. **e**—Even in a circuit where half the signals are active high and half are active low, the mixed-rail system lets us easily keep them all straight.

The only difference between the circuits shown in Figure 8d and 8e is the activation level on the output of the AND gate. In Figure 8e there is a logic incompatibility introduced where D(L) enters the OR gate's active-high input. So, D shows up complemented in Y, $Y = *A + *D$. However, in Figure 8d there is no logic incompatibility, so D is not complemented in Y, $Y = *A + D$.

DRAFTING AND READING MIXED-LOGIC NOTATION

Drawing schematics is an art the same way mathematics is an art. In the world of mathematics, for a given theorem, there are convoluted meandering proofs and there are succinct proofs. Both are equally valid, but the latter is preferred. Figure 9 contains several examples of both convoluted and succinct graphical expressions of logic equations.

Figures 9a and 9b are both illustrations of the same physical circuit. The circuit generates a bus reset signal (BusReset). The output should be asserted when SystemReset is asserted or when a microcontroller-generated bus reset signal ($\mu CBusReset$) is asserted. This means the bus will be held in reset (presumably a safe state) while the system is being reset and the microcontroller can at any time force a reset.

Many engineers hold fast to the belief that because the 7408 is described as a quad two-input AND gate, the clearest way to draw the physical device is as an AND gate (see Figure 9a).

Some engineers don't seem to understand that a physical device is equally well represented several different ways. Figure 9a is not an uncommon representation of the circuit, nor is it technically wrong. After all, it does describe the operation of the physical circuit.

Let's examine the logic equations for the homologous circuit shown in Figure 9a. The reader cannot know whether the intended function of the 7408 is an AND or an OR function except by looking at the gate. Figure 9a naturally leads to the convoluted equation:

$$*BusReset = *SystemReset \cdot * \mu CBusReset$$

which equation reads, "the bus is NOT being reset when the system is NOT being reset AND the microcontroller is NOT commanding a bus reset." Although this is true, it certainly is a meandering way to get the point across. The clever reader might do a DeMorgan's equivalent on the equation by complementing both sides, then simplifying, thus obtaining:

$$BusReset = SystemReset + \mu CBusReset$$

which reads, "the bus will reset when the system is in reset or the microcontroller commands a bus reset." Now, I'd say that's a bit more succinct than the previous equation.

Figure 9b shows how to use mixed-logic notation to clearly express the equation on the schematic. A reader looking at Figure 9b knows the 7408 is being used to generate an OR function. There are no logical incompatibilities to cause complements to be generated. Thus the reader may immediately write the equation:

$$BusReset = SystemReset + \mu CBusReset$$

Always draw the schematic to communicate the logical function of the circuit, regardless of what name the manufacturer gives to the physical device. Figure 9c is an example of a common decoding for a parallel input/output chip (a PIO).

The logic equation is rapidly deduced, and we can quickly see that the PIO is selected when I/O Request (IOREQ) is asserted and $A_{15} = 0$, $A_{14} = 1$, and $A_{13} = 1$. In this example, the intended AND function of the 7420 happened to correspond to the common representation shown on the manufacturer's datasheet.

Figure 9d shows one last example of both a good and a poor graphical representation of the logical operation of the circuit. The schematic with the AND function lends itself to rapid deciphering by the reader. Clearly the RAM is selected when Memory Request (MEMREQ) is active AND $A_{15} = 0$ and $A_{14} = 1$.

The alternate schematic in Figure 9d represents the identical physical circuit,

but the major combinatorial gate (the 7427) is drawn as a NOR gate (an OR function with active-high inputs and an active-low output). This leads to the rather ugly equation:

$$CS = * (*MEMREQ + A_{15} + *A_{14})$$

which reads, "the RAM is NOT selected when memory is NOT selected OR A_{15} is asserted or A_{14} is NOT asserted." Although technically correct, this is a rather obscure way to describe the circuit behavior. The equation from Figure 9d's AND representation, reads "RAM is selected when memory is selected and $A_{15} = 0$ and $A_{14} = 1$," which is much clearer.

This result can be derived algebraically from the equation deduced from the OR representation, but why confuse readers with an OR function and hope they're quick enough to know you really mean AND, and then require them to work algebra? Clearly the best method is to represent the 7427 as an AND function with active-low inputs and an active-high output.

THE TERMINAL STATE

In these days of VHDL and Verilog, universities may be short-changing their students by not teaching them how to clearly and concisely communicate digital logic schematically. Well, shame on them. But the good news is that mixed-logic notation is simple to learn and easy to use.

Many engineers and professors have shied away from mixed-logic notation because it requires them to fully understand the ramifications of how we interpret voltages in digital circuits, how those interpretations interact with the Boolean abstractions, and how to clearly express these ideas schematically. Yes, details and more details. But in the end, mixed-logic offers a superior method for clearly expressing how a circuit operates.

The most important thing to keep in mind is that a schematic is a communication tool. The target audience consists of other engineers. Use a naming convention that clearly expresses activation level. Use the schematic symbols to indicate the logical Boolean operations that are performed. For additional clar-

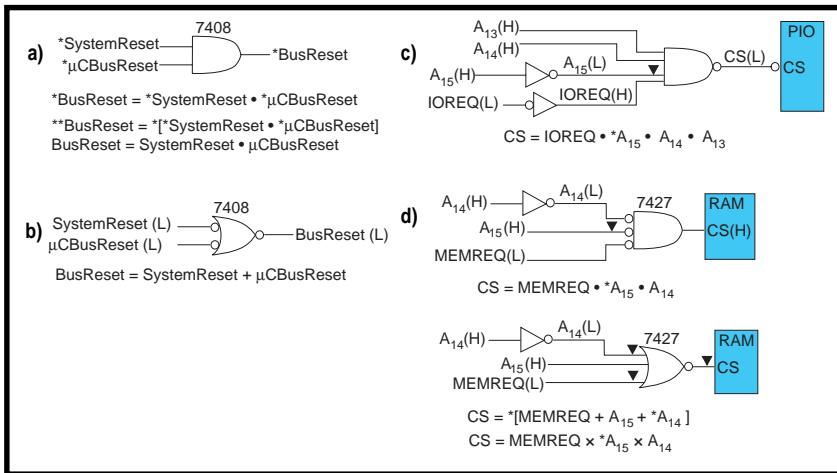


Figure 9a—This is an example of what not to do. Don't make reading schematics tough on the reader. **b**—This is physically equivalent to the circuit shown in 9a, but much easier to read. **c**—Using one logic-level converter to introduce a logic incompatibility and one logic-level converter to eliminate a logic incompatibility enables us to create a schematic from which you can effortlessly read the Boolean equation of interest. **d**—This shows both an easy to read schematic and a tough to decipher, but all too common, version.

ity, put the device's part number and reference designator near the gate. To aid in troubleshooting, always put the pin numbers for each pin on the schematic.

Tradeoffs and details are the stock and trade of engineers. Never fear the tradeoffs, and always consider the details.

Special thanks to Professor Richard F. Tinder [2] at Washington State University, Pullman, WA; Greg Young, Electrical Engineer, Z-World, Davis, CA; and Raymond D. Payne, Chief Electrical Engineer, Under Control Inc. Davis, CA.

Bob Perrin spends his days designing general-purpose C-programmable embedded controllers and troubleshooting customer system-level problems for Z-World (www.zworld.com). Over the last ten years, Bob has designed instrumentation for agronomy, soil physics, and water activity research. He was also the lead design engineer for an intrinsically safe line of workstations for use in explosive gas and particulate environments (class 1, divisions 1 and 2). For more articles by Bob, visit his online library at www.engineerbob.com.

REFERENCES

- [1] J.F. Wakerly, *Digital Design Principles and Practices*, Prentice Hall, Englewood Cliffs, NJ, 1994.
- [2] R.F. Tinder, *Digital Engineering Design: A Modern Approach*, Prentice Hall, Englewood Cliffs, NJ, 1991.

Circuit Cellar, the Magazine for Computer Applications. Reprinted by permission. For subscription information, call (860) 875-2199 or subscribe@circuitscellar.com.