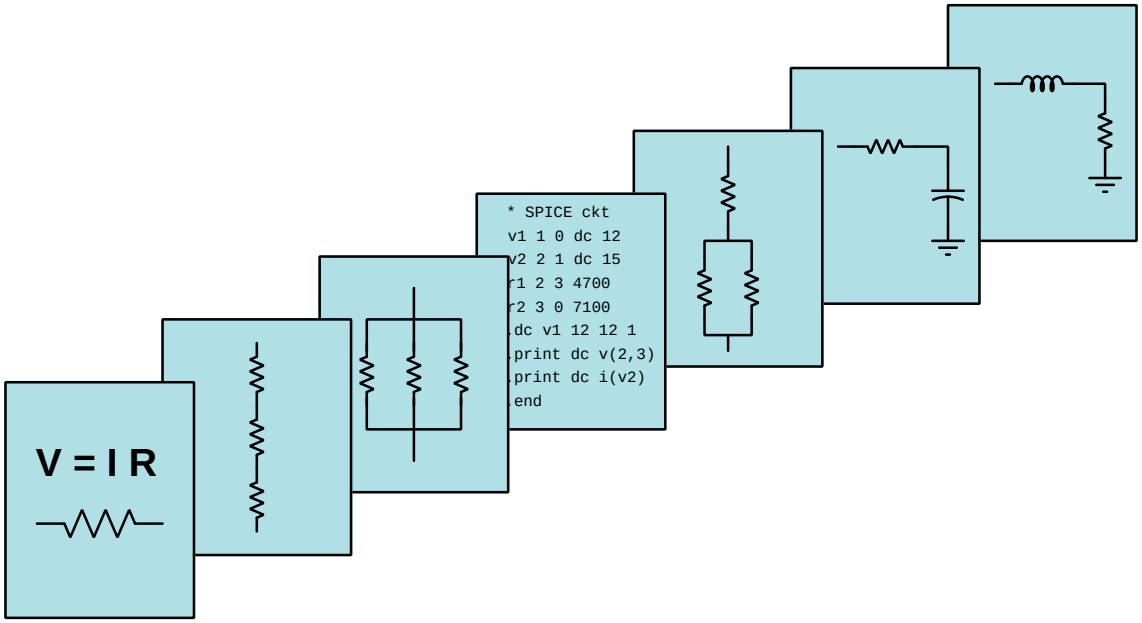
Modular Electronics Learning (ModEL) project



Semiconductor Logic Gates

Last update = 19 November 2024

Contents

Chapter 1

Introduction

1.1 Recommendations for students

Logical functions such as AND and OR are mathematical abstractions, and may be implemented using multiple technologies. Any semiconductor-based circuit implementing a logical function is called a *logic gate*. They usually take the form of an *integrated circuit* (IC), where a multitude of circuit components are etched onto a single piece of silicon material. A typical logic IC contains more than one gate, with metal pins provided for DC power supply connections and connections for input and output signals. Logic gate signals are typically ground-referenced voltages, with zero voltage representing a "low" or 0 or "false" logic state, and full source voltage representing a "high" or 1 or "true" logic state.

Important concepts related to logic gates include **logic states**, **logic levels**, **high** and **low** logic states, **logic functions**, **truth tables**, **pullup** and **pulldown** resistors, **sinking** versus **sourcing** current, **propagation delay**, **bipolar junction transistor** behavior, **field-effect transistor** behavior, effects of **opens** versus **shorts**, **static electricity**, **push-pull** transistor pairs, logic IC **families**, **open-collector** and **open-drain** logic, **tri-state** logic, **floating** conductors, **bus contention**, **steering diodes**, **Schmitt trigger** gates, and **hysteresis**.

Here are some good questions to ask of yourself while studying this subject:

- How might an experiment be designed and conducted to measure the accepted voltage levels for "high" and "low" states at the input of a logic gate? What hypothesis (i.e. prediction) might you pose for that experiment, and what result(s) would either support or disprove that hypothesis?
- How might an experiment be designed and conducted to measure the threshold voltage levels for a Schmitt trigger logic gate? What hypothesis (i.e. prediction) might you pose for that experiment, and what result(s) would either support or disprove that hypothesis?
- How might an experiment be designed and conducted to measure the propagation delay of a logic gate? What hypothesis (i.e. prediction) might you pose for that experiment, and what result(s) would either support or disprove that hypothesis?

- What are some practical applications of logic gates?
- How do analog and digital quantities differ from one another?
- What does a *truth table* represent?
- What is a *logic function*?
- What is the fundamental characteristic of an OR logic function?
- What is the fundamental characteristic of an AND logic function?
- What is the fundamental characteristic of a NOT logic function?
- What is the fundamental characteristic of a NOR logic function?
- What is the fundamental characteristic of a NAND logic function?
- What are some advantages of TTL logic gates over CMOS?
- What are some advantages of CMOS logic gates over TTL?
- What are some important performance parameters for logic gates?
- What causes hysteresis, and what are some examples of it in action?
- Why must CMOS integrated circuits be handled with special care?
- Why is it generally bad practice to connect multiple gate outputs together?
- What is the purpose of a "pullup" or a "pulldown" resistor?
- What logic state does a TTL gate input assume when it floats?
- What logic state does a CMOS gate input assume when it floats?
- What is the purpose of logic gates with "tri-state" output capability?
- What are some practical applications of Schmitt trigger gates?
- What does it mean to classify logic ICs into different "families"?

1.2 Challenging concepts related to digital signal integrity

The following list cites concepts related to this module's topic that are easily misunderstood, along with suggestions for properly understanding them:

- **Sourcing versus Sinking output currents** – a common misconception is that since the output of a logic gate is called "output" it must mean that current only ever *exits* that terminal. This is untrue. All that "output" actually signifies is the fact that the gate is outputting *information* consisting of voltage values measured between that terminal and ground. Sometimes the assertion of a "low" (zero-voltage) logical state requires that the gate actually draw current *in* through its "output" terminal!

- **Gates require DC power** – since logic gates are really just transistor amplifier circuits, those treansistors require constant DC voltage applied between their power terminals to function properly. Gates are *not* powered through their input terminals – these terminals only receive *information* in the form of "high" and "low" logic states, and ideally pass negligible current.

- **Pullup and pulldown resistors** – in digital circuits, resistors are often used to provide a secure logic state when an input device (such as a switch) goes to a high-impedance (open) mode. Students often have difficulty figuring out exactly where these resistors should go in a circuit. The most common mistake I've seen is to place one of these "pullup" or "pulldown" resistors in *series* with a gate input, which will accomplish absolutely nothing. The "trick" to getting this placement right, if you can call it a trick at all, is to literally follow the word "pullup" or "pulldown". A *pullup* resistor pulls the logic state of a wire up to the positive supply rail, and so must connect between the gate input and +V. A *pulldown* resistor pulls the logic state of a wire down to ground potential, and so must connect between the gate input and ground. In either case, the resistor provides a sure path to the opposite power rail that the input device connects to when active (closed).

- **Logic level voltages** – in any positive-logic circuit, a "high" state is ideally any ground-referenced voltage at or near the positive power supply rail voltage, and a "low" state is ideally any ground-referenced voltage at or near zero. However, the presence of noise, interference, and loading effects means logic circuits must be able to tolerate imperfect "high" and "low" voltage levels in order to reliably function in real-world scenarios. Thus, logic gate "families" are designed to operate such that "high" signals are defined as any ground-referenced voltage falling within a standardized *range* up to and including the positive rail potential, and "low" signals are defined as falling within a standardized *range* down to and including zero. These voltage requirements must be honored especially in cases where we need to interface two or more different families of logic gate, or gates powered by differing supply voltages, in the same circuit.

1.3 Recommendations for instructors

This section lists realistic student learning outcomes supported by the content of the module as well as suggested means of assessing (measuring) student learning. The outcomes state what learners should be able to do, and the assessments are specific challenges to prove students have learned.

- **Outcome** – Demonstrate effective technical reading and writing

  Assessment – Students present their outlines of this module's instructional chapters (e.g. Case Tutorial, Tutorial, Historical References, etc.) ideally as an entry to a larger Journal document chronicling their learning. These outlines should exhibit good-faith effort at summarizing major concepts explained in the text.

- **Outcome** – Apply the concept of a truth table to a practical logic gate circuit

  Assessment – Develop a truth table describing the status of one or more load devices in a logic gate circuit as a function of switches or other on/off inputs to that gate circuit.

- **Outcome** – Apply the concepts of transistor operation to logic gate internal circuitry

  Assessment – Predict the on/off states of all transistors within a given logic gate circuit based on input logic states; e.g. pose problems in the form of the "Discrete analysis of a bipolar AND gate" Conceptual Reasoning question.

- **Outcome** – Design a logic gate circuit to fulfill a specified purpose

  Assessment – Sketch wire placement in a pictorial diagram for a circuit employing a packaged set of logic gates to perform a given function; e.g. pose problems in the form of the "Sketching an OR gate circuit" Conceptual Reasoning question.

- **Outcome** – Diagnose a faulted logic gate circuit

  Assessment – Determine the probability of various component faults in a logic gate circuit given symptoms and measured values; e.g. pose problems in the form of the "Malfunctioning security alarm system" Diagnostic Reasoning question.

- **Outcome** – Independent research

  Assessment – Locate logic gate datasheets and properly interpret some of the information contained in those documents including supply voltage range, input logic voltage levels, output logic voltage levels, maximum switching speed, maximum output current, internal schematic diagrams (not available in all datasheets), etc.

Chapter 2

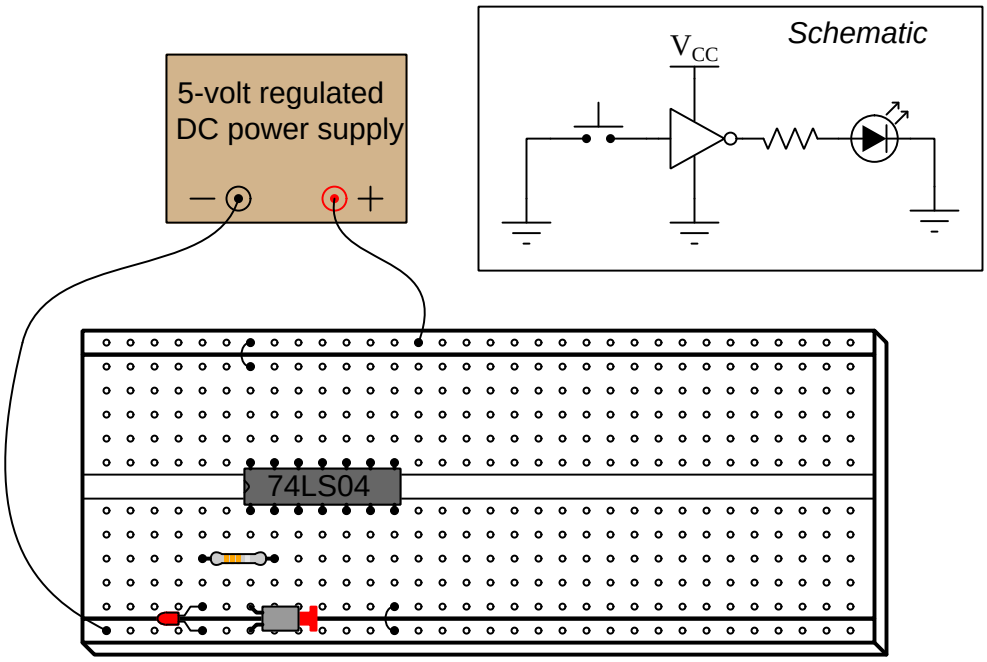
Case Tutorial

The idea behind a *Case Tutorial* is to explore new concepts by way of example. In this chapter you will read less presentation of theory compared to other Tutorial chapters, but by close observation and comparison of the given examples be able to discern patterns and principles much the same way as a scientific experimenter. Hopefully you will find these cases illuminating, and a good supplement to text-based tutorials.

These examples also serve well as challenges following your reading of the other Tutorial(s) in this module – can you explain *why* the circuits behave as they do?

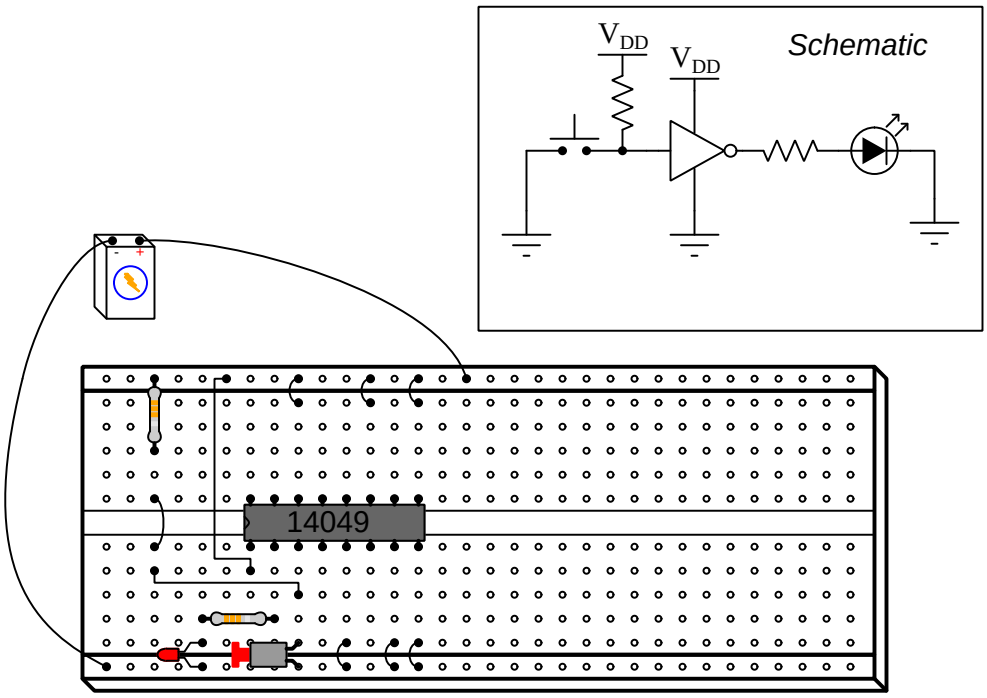
2.1 Example: inverter demonstration circuits

The DIP circuit is a TTL hex inverter (it contains *six* "inverter" or "NOT" logic gates), but only one of these gates is being used in this circuit. Being TTL, it requires a regulated power supply voltage of 5 Volts:



Research a manufacturer's *datasheet* for this logic gate IC to see the "pinout" diagram showing which pins on the IC package connect to which inverter gate terminals inside.

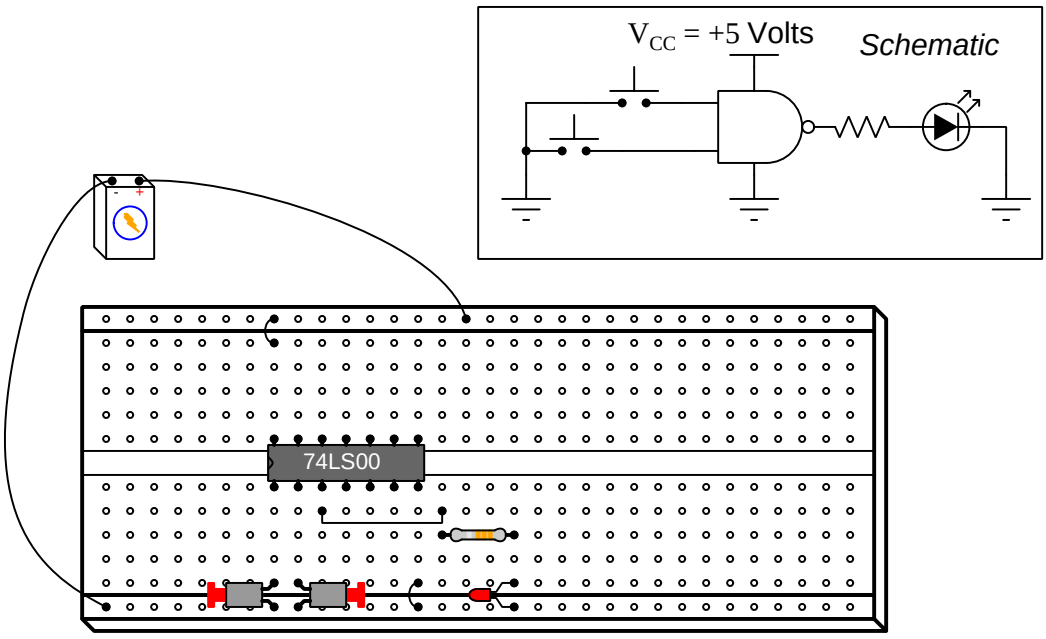
Next we have a CMOS version of the same circuit. Being CMOS, the power supply voltage requirements are not as strict, but we must use a pull-up resistor to ensure a "high" logic state at the input terminal when the pushbutton switch contacts are open. This is the same reason that all unused input pins are tied either to +V ("high") or ground ("low"). Unlike TTL, whose inputs default to a "high" state when floating, CMOS gate inputs assume no default state and therefore must be given connections to either V_{DD} (+V) or V_{SS} (ground) in order to avoid ambiguous states:



Note the unusual locations of the +V power supply pin on this IC: pin 1 rather than the highest-numbered pin (upper-left) as we more commonly find.

2.2 Example: NAND gate demonstration circuit

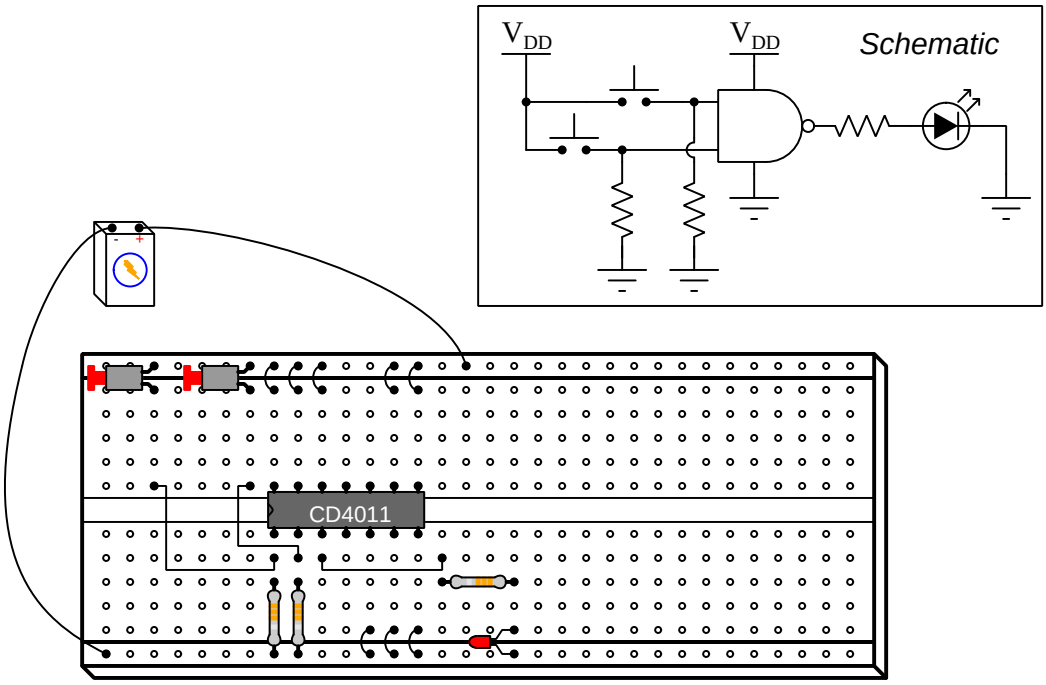
Below we see a schematic diagram as well as pictorial illustration of a NAND gate demonstration circuit using the model 74LS00 quad 2-input NAND gate IC. This particular IC uses bipolar (PNP and NPN) type transistors with low-power Schottky diode architecture (hence the "LS" designation for "Low-power Schottky"), and requires a tightly voltage-regulated power supply at 5 Volts in order for those transistors and diodes to function well together. An interesting characteristic of bipolar-type semiconductor logic gates is that any unconnected input terminal will assume a "high" logical state, which explains why the switches connect to ground. This means pressing each normally-open pushbutton switch will assert a "low" logical state at that input, while releasing each switch lets the gate's input return to its default "high" state:



Research a manufacturer's *datasheet* for this logic gate IC to see the "pinout" diagram showing which pins on the IC package connect to which NAND gate terminals inside. Feel free to research datasheets and build demonstration circuits like this one using types of TTL logic gates other than the 74LS00 quad 2-input NAND:

- 74LS08 – quad 2-input AND gate
- 74LS32 – quad 2-input OR gate
- 74LS02 – quad 2-input NOR gate
- 74LS86 – quad 2-input XOR gate

Next we see a CMOS version of this same circuit using the model CD4011 quad 2-input NAND gate IC. Being CMOS, the power supply voltage limits are quite wide which allows for battery power, but we must use pull-down resistors to ensure "low" logic states at the input terminals when the pushbutton switch contacts are open. This is the same reason that all unused input pins are tied either to +V ("high") or ground ("low"). CMOS gate inputs assume no default state when floating and therefore must be given connections to either V_{DD} (+V) or V_{SS} (ground) in order to avoid ambiguous states:

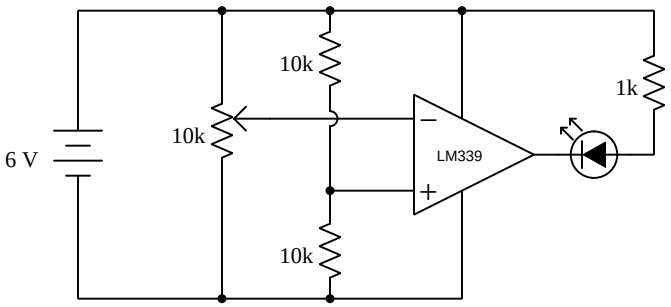


Research a manufacturer's *datasheet* for this logic gate IC to see the "pinout" diagram showing which pins on the IC package connect to which NAND gate terminals inside. Feel free to research datasheets and build demonstration circuits like this one using types of CMOS logic gates other than the CD4011 quad 2-input NAND:

- CD4081 – quad 2-input AND gate
- CD4071 – quad 2-input OR gate
- CD4001 – quad 2-input NOR gate
- CD4070 – quad 2-input XOR gate

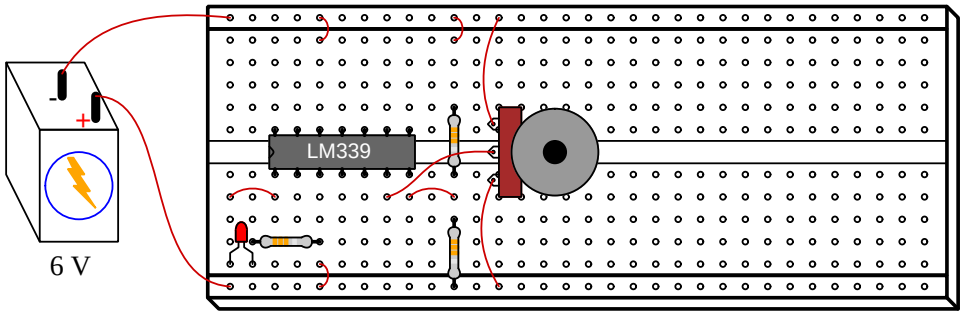
2.3 Example: comparator demonstration circuit

This circuit uses a model LM339 comparator (just one of the four comparators contained in the LM339 IC) to demonstrate basic comparator operation:



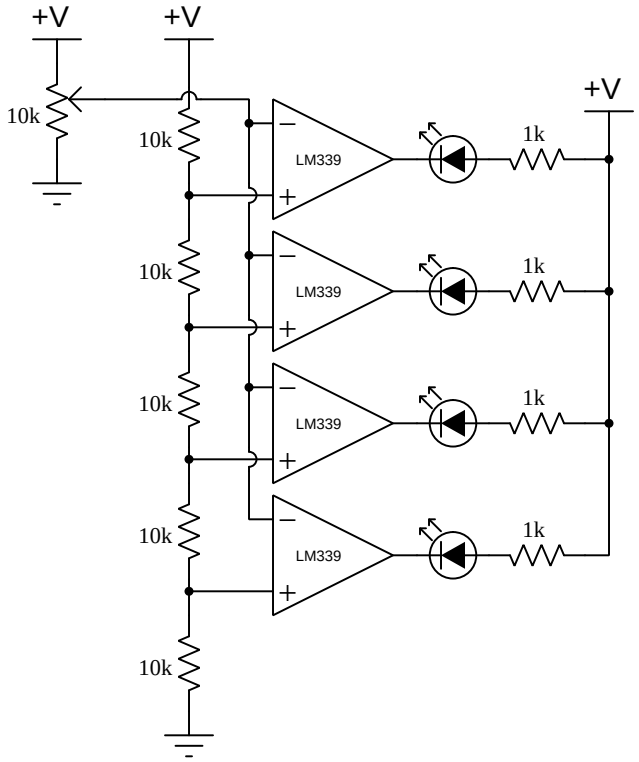
The purpose of the potentiometer is to serve as an adjustable voltage divider, to take the 6 Volt battery's voltage and divide it down to some adjustable value between 0 Volts and 6 Volts, inclusive. The two 10 kilo-Ohm fixed resistors also form a voltage divider, providing a fixed (non-variable) voltage equal to one-half of the battery's voltage.

Below is a pictorial representation of the LM339 IC and other components inserted into a solderless breadboard:

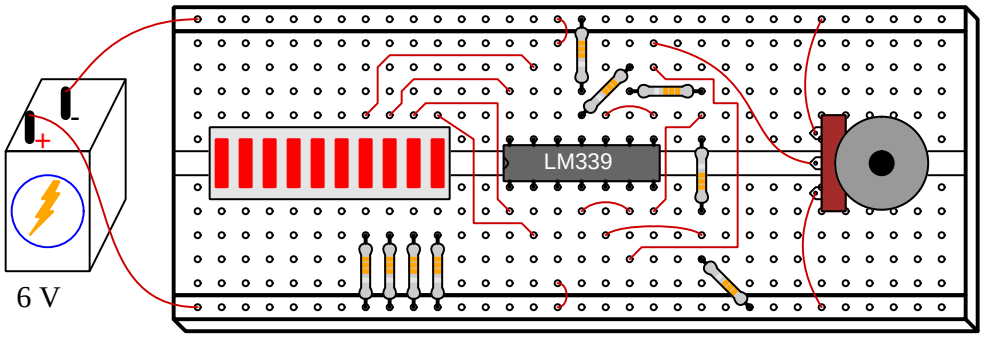


Note the unusual location of the LM339's power supply terminals: (+) on pin 3 and (−) on pin 12. Also note that the model LM339 cannot source output current, but can only sink. This is why the LED must be connected the way it is, sending current *into* the comparator's output terminal when energized.

2.4 Example: simple bargraph display circuit



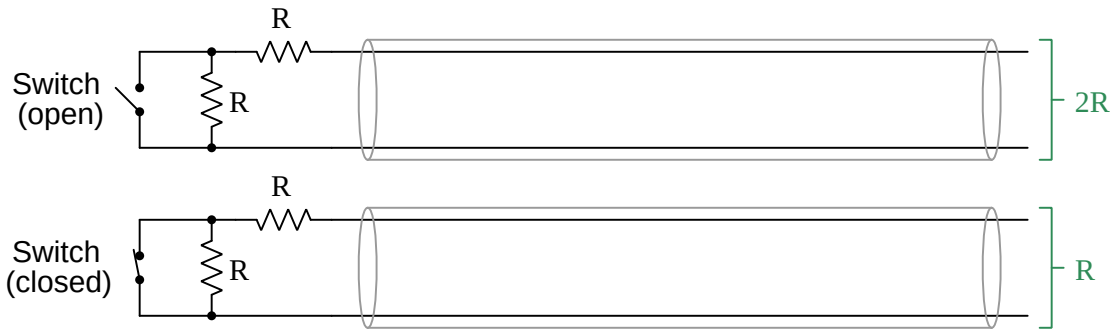
While individual LEDs will suffice, a ten-segment LED module looks much nicer:



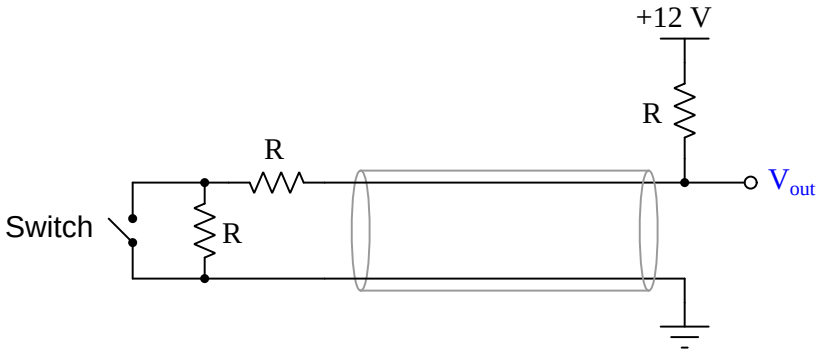
Note the unusual location of the LM339's power supply terminals: (+) on pin 3 and (−) on pin 12. Also note that the model LM339 cannot source output current, but can only sink. This is why the LEDs must be connected the way they are, sending current *into* the respective comparator output terminals when energized.

2.5 Example: supervised switch reading circuit

A *supervised* switch circuit is one where a remotely-located switch's status is communicated by analog resistance values rather than by the extreme limits of closed (0 Ohms) and open (infinite Ohms), in order to differentiate between the switch's legitimate states and the connecting cable being failed open or shorted. These are used in fire alarms and other electronic systems where high reliability is extremely important:

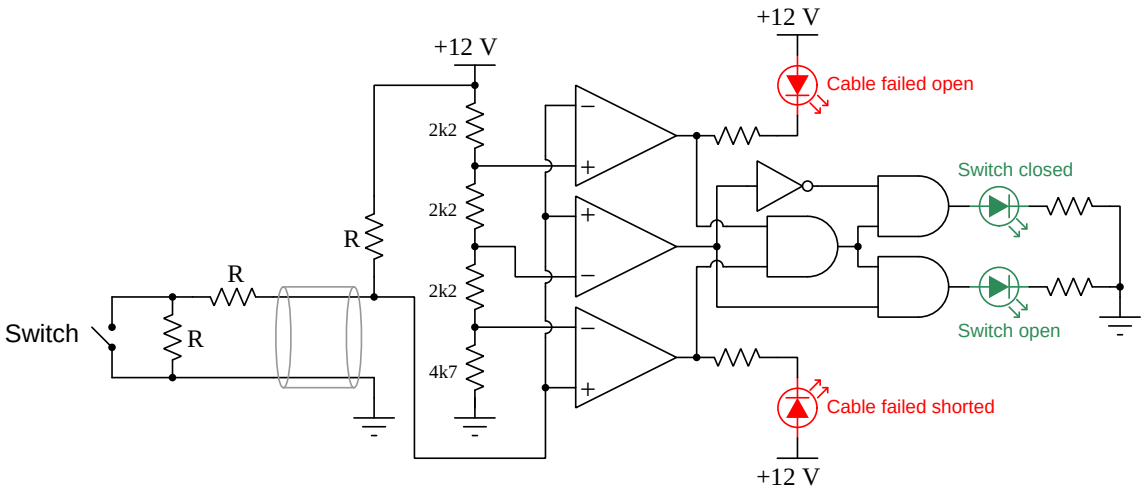


If connected to a DC voltage source and third resistor, we find four distinct voltage values (measured between the output terminal and ground) as we analyze the circuit for its six possible conditions:



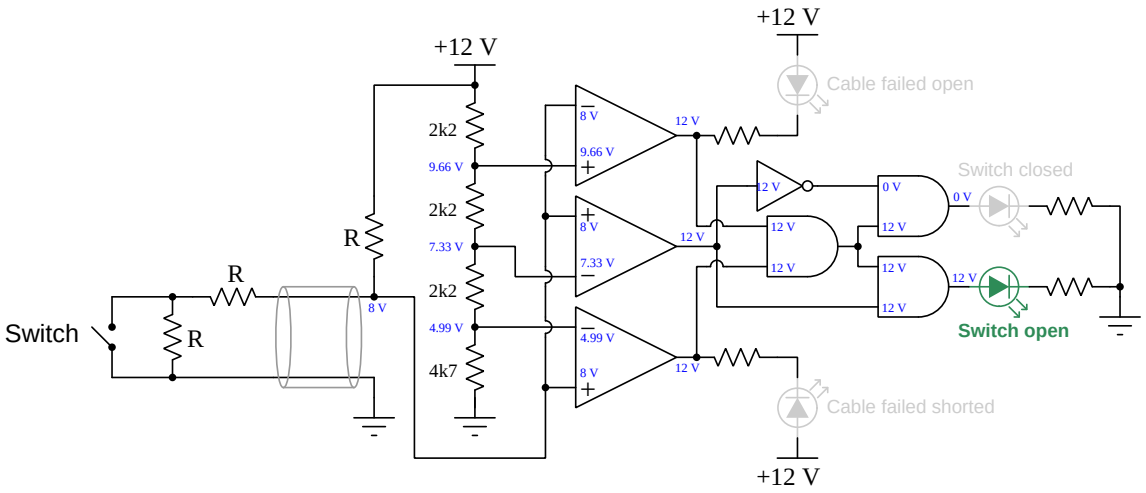
Switch state	**Cable state**	V_{out}
Open	Healthy	8 Volts
Closed	Healthy	6 Volts
Open	Failed open	12 Volts
Closed	Failed open	12 Volts
Open	Failed shorted	0 Volts
Closed	Failed shorted	0 Volts

The following comparator-and-gate circuit interprets these resistance values to activate LEDs indicating the status of the switch *and* cable:

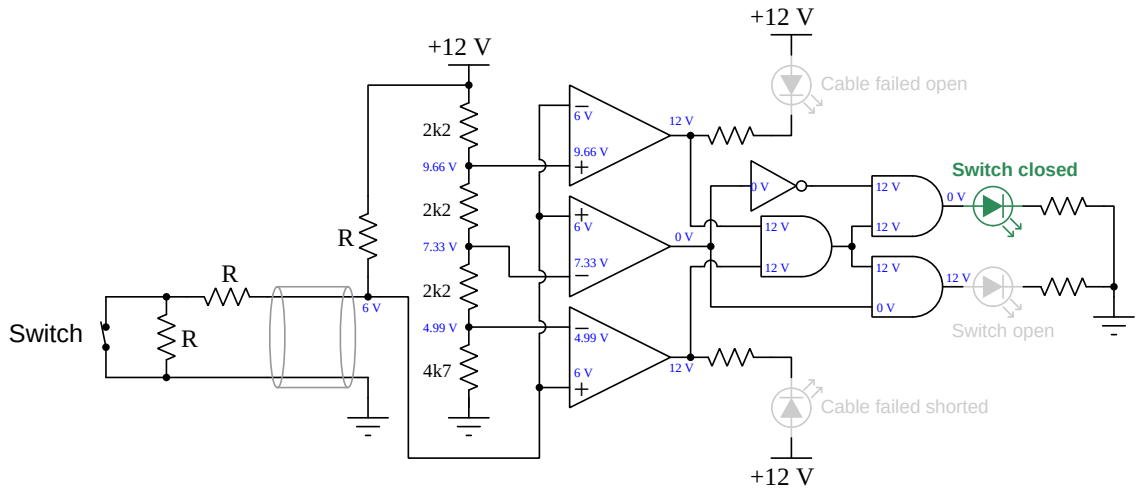


Note the four LEDs illuminated by this circuit: two of them showing healthy switch states (open or closed) and two of them ready to show faulted-cable conditions should they occur. This is the value of a supervised switch circuit: its ability to comprehensively report the condition of the remotely-located switch, so that any wiring problems may be detected as soon as they occur and remedied before they compromise the operation of the critical system.

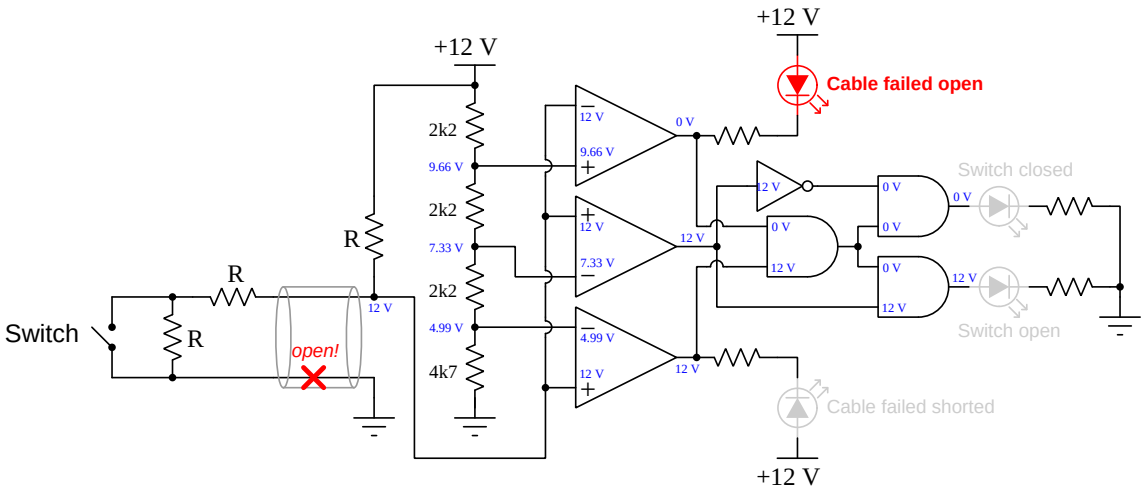
We may understand this circuit by annotating the diagram with all ground-referenced voltage values, for each of the relevant conditions. Assuming that all ICs are powered by the same +12 Volt DC power source, first we will show the circuit with a healthy cable and the switch open:



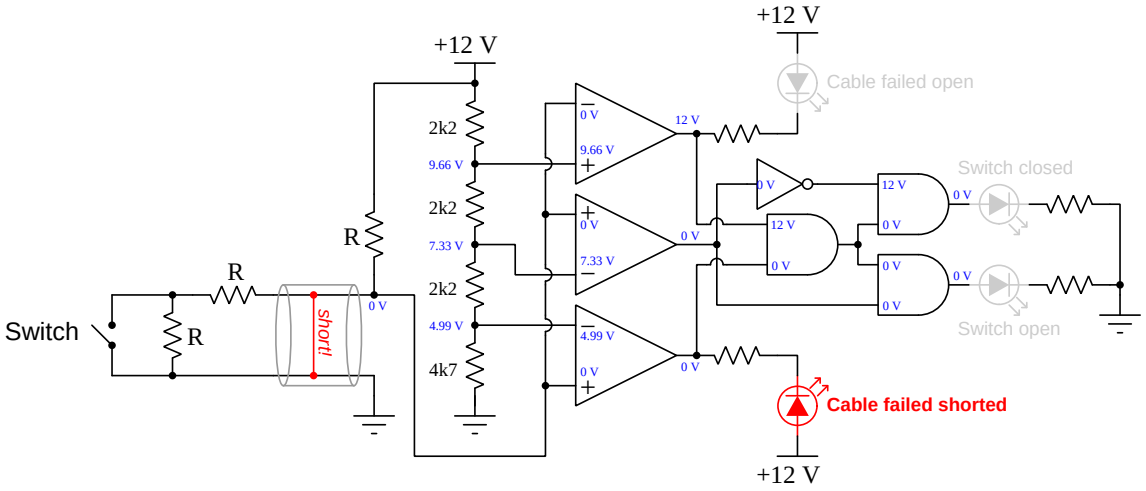
Next, showing the circuit's voltages with reference to ground with a healthy cable and the switch closed:



Next, showing the circuit's voltages with reference to ground with a failed-open cable (switch state is irrelevant):



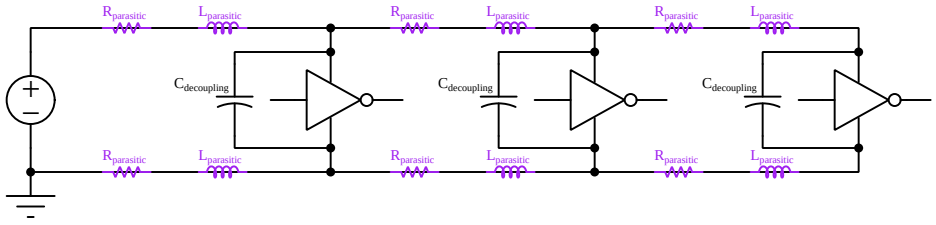
Finally, showing the circuit's voltages with reference to ground with a failed-shorted cable (switch state is irrelevant):



2.6 Example: decoupling capacitors

Many forms of electronic circuits are especially sensitive to variations in DC power supply voltage resulting from pulsations of current drawn from the supply over time, the DC terminal voltage tending to "sag" or "droop" as current surges. A highly effective safeguard against this problem is to attach a capacitor as closely as possible to the power supply terminals of the affected circuit, the capacitor serving as a reservoir of electrical energy to temper these transient voltage events.

The nature of the problem may be seen in the following schematic diagram, where we have three separate digital logic "gates" all powered by a common DC voltage source. The wires or printed circuit board traces carrying DC power to these three loads inherently possess both resistance (R) and inductance (L), both of which cause sudden voltage drops when current suddenly increases:



If we imagine the far-right logic gate suddenly drawing current because its output switches state, all the parasitic R and L elements between it and the DC voltage source will manifest additional voltage drop[1], resulting in all three logic gates experiencing a momentary decrease in supply voltage if not for the stabilizing influence of the capacitors connected in parallel to each gate's power supply terminals.

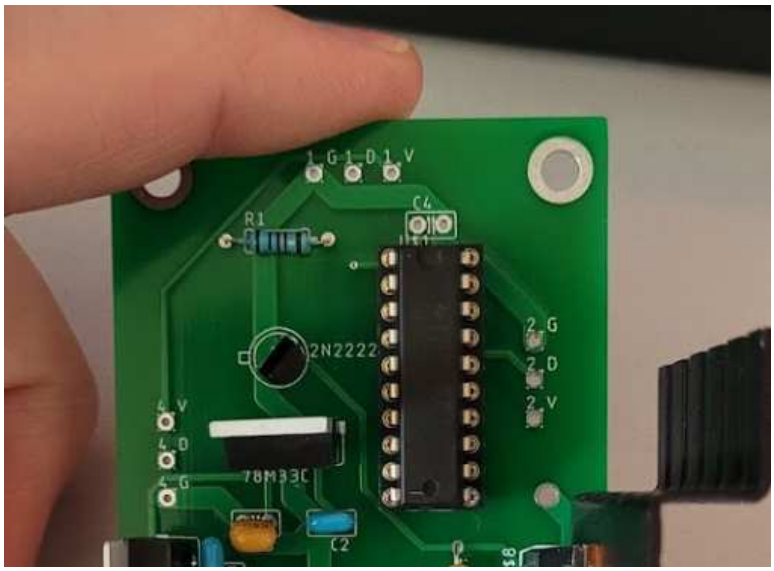
The tendency for DC supply voltage to suddenly dip in a circuit due to effects such as this is sometimes referred to as *ground bounce* or V_{DD} *bounce*[2]. The practice of connecting capacitors in parallel with the integrated circuits (ICs) is called *decoupling*, because it helps prevent the switching action of one load from interfering or "coupling" to another load. Such "bounce" may severely degrade a circuit's performance, causing noise problems in analog circuits and all manner of problems in digital circuits (e.g. erratic signal timing, false counts).

Decoupling is a good practice in any system where we anticipate rapid on/off switching of the components, and it is best to use capacitors with low equivalent series resistance (ESR) so they will be able to source high levels of transient current if necessary. For very high-frequency printed circuit boards a good practice for DC power distribution is to devote two or more of the layers in a multi-layer board as *power planes*: sheets of copper sandwiched between layers of insulating board material acting as "busses" for each rail of the DC power source. Not only does this make DC power easy for each device to access (simply install a "via" which is an inter-layer connector between the desired power plane layer and the device power terminal) but the natural capacitance existing between power planes separated by board insulation contributes to the decoupling offered by discrete capacitors installed adjacent to each device.

[1] Resistance drops voltage proportional to current, which is simply Ohm's Law in action ($V = IR$). Inductance drops voltage in response to a rise in current over time ($V = L\frac{dI}{dt}$), the voltage drop's magnitude being proportional to the *speed* at which current increases.

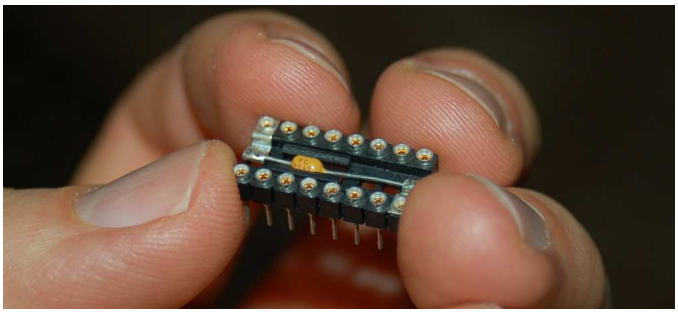
[2] In CMOS digital circuits the positive pole of the DC voltage source is commonly referred to as V_{DD} in honor of its connection to the Drain terminals of many transistors within the IC.

In the following photograph you can see two holes in a printed circuit board ready to accept a small capacitor (C_4) immediately above the 20-pin integrated circuit. When soldered into this incomplete PCB, C_4 will stabilize the DC power supply voltage to that IC so that it experiences less "bounce" on its power supply terminals during switching events:



Note how close to the IC the decoupling capacitor C_4 will be located. The rationale for positioning the decoupling capacitor immediately adjacent to the protected IC is to minimize the amount of length for any copper traces connecting it to the IC's power supply pins, because greater conductor length means more resistance and more inductance, all other factors being equal. Recall how it is the presence of R and L between the load and DC power source that aggravates the "bounce" problem, and so we need our decoupling capacitor to have as little impedance as possible between it and the load whose DC voltage we intend to stabilize.

A clever way of incorporating decoupling capacitors into circuits is found in the following photograph of a 14-pin DIP socket. Here the decoupling capacitor is built-in, connected between pins 7 and 14 which are commonly the negative and positive DC power pins for a great many TTL and CMOS logic ICs:



As a general rule, the best decoupling capacitor types are ceramic, mica, and polystyrene due to their naturally low equivalent series resistance (ESR). Tantalum electrolytic capacitors, although blessed with relatively high capacitance in small packages, have fairly high ESR which limits their effectiveness as decoupling capacitors.

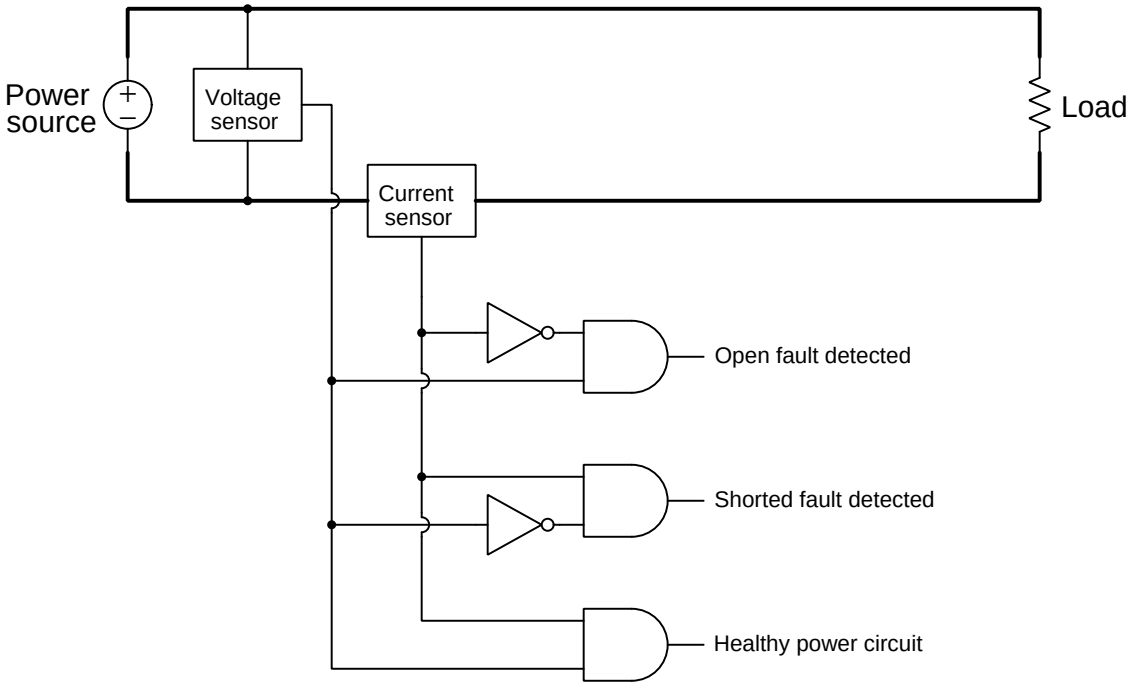
2.7 Gallery of logic gate applications

Semiconductor logic gates have many, many practical applications. The following subsections show just a few!

As is customary with most logic gate schematic diagrams, DC power terminals for the logic gates themselves have been omitted in order to reduce clutter on the diagrams. Know, however, that logic gates are transistor circuits which require DC power to function!!

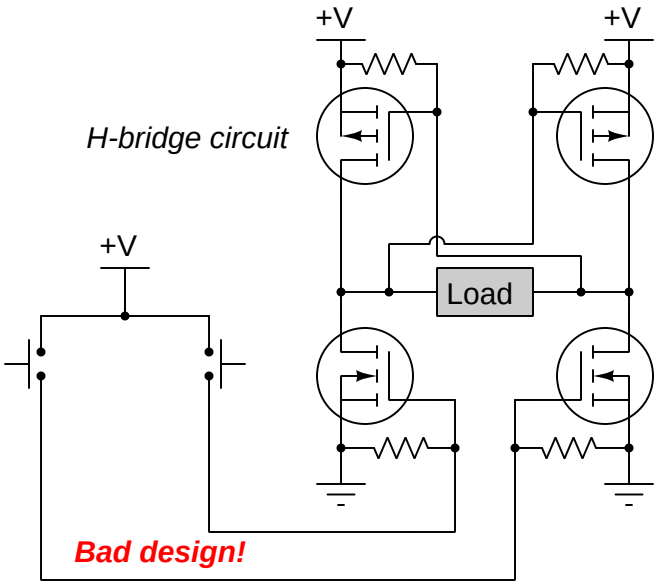
2.7.1 Power circuit fault detector

A power circuit equipped with voltage and current sensors providing boolean ("true" or "false") indications of voltage and current may be equipped with a logic circuit to take those sensor signals and from their values determine if there is any fault in the power circuit:

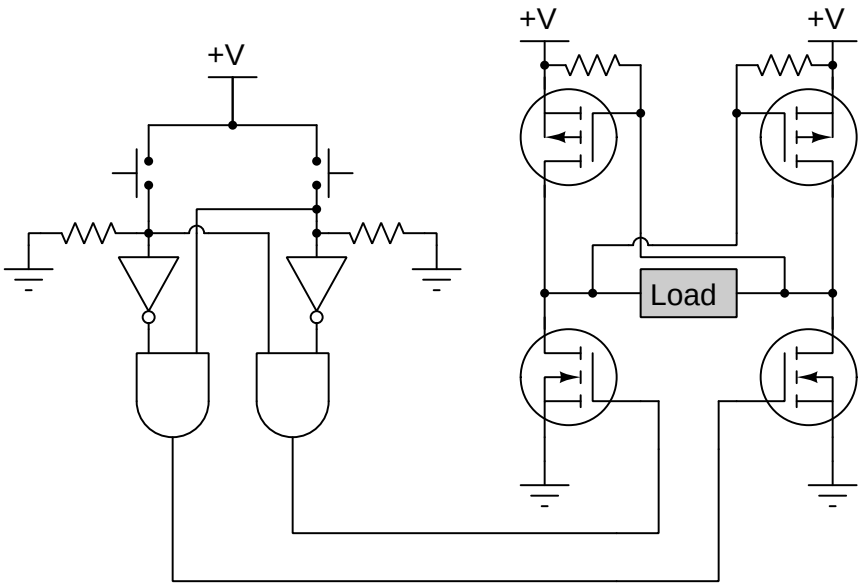


2.7.2 H-bridge driver circuit

H-bridge circuits are networks of four power transistors useful for controlling power to a load. By activating either one of the lower transistors in the bridge, the appropriate upper transistor becomes activated as well, passing current through the load in one direction or the other. However, the simplistic H-bridge circuit shown below is a bad design because all four transistors will activate (and short-circuit the +V source) if anyone happens to press both pushbuttons happen at the same time:

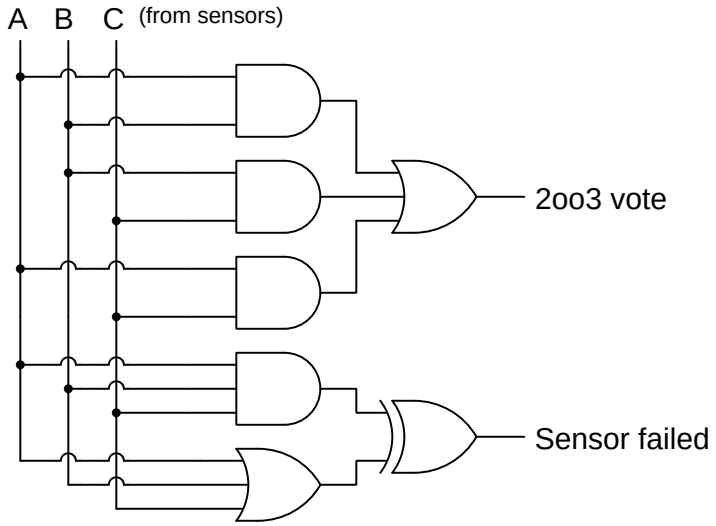


The addition of some logic gates will safeguard against this improper mode of operation, by activating the H-bridge only if *one* of the two pushbuttons is pressed but not if both are simultaneously pressed:



2.7.3 Two-out-of-three voting circuit

High-reliability systems often use redundant components to achieve fault tolerance. For example a system may use three identical sensors (A, B, and C) all detecting the same physical stimulus, all three sensors reporting their statuses to an electronic "voting" logic circuit providing a reliable output signal based on a two-out-of-three ("2oo3") vote:

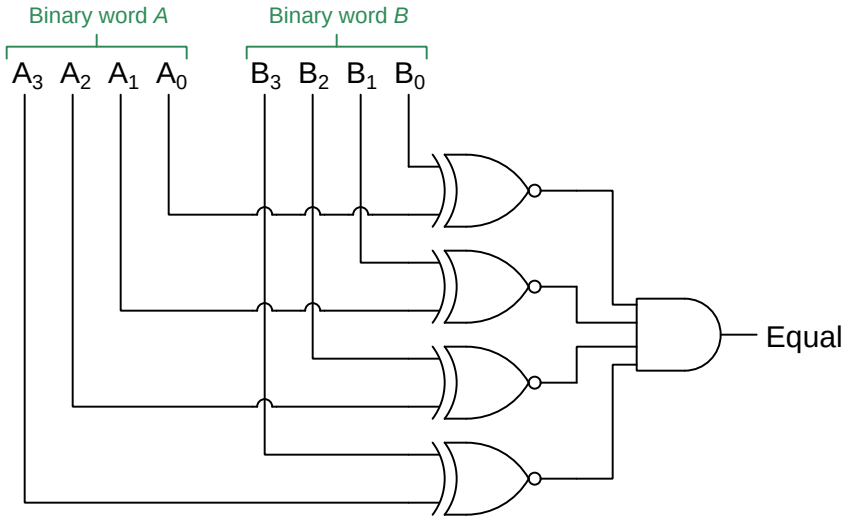


A	B	C	2oo3 vote	Sensor failed
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	1
1	0	0	0	1
1	0	1	1	1
1	1	0	1	1
1	1	1	1	0

All it takes is for at least two of the three redundant sensors to agree with each other to generate a reliable "high" or "low" output from the voting circuit, and if there is any disagreement at all between the three sensors we will know by the "sensor failed" output going high. This provides a fault tolerance of one, which means any one of the three sensors could fail in any state and the voting circuit would still reliably indicate the true status of the measured stimulus.

2.7.4 Binary word comparator

A *binary word* is a collection of *bits* representing a numerical quantity or a symbolic code[3]. This circuit compares two 4-bit binary words to check for equality:

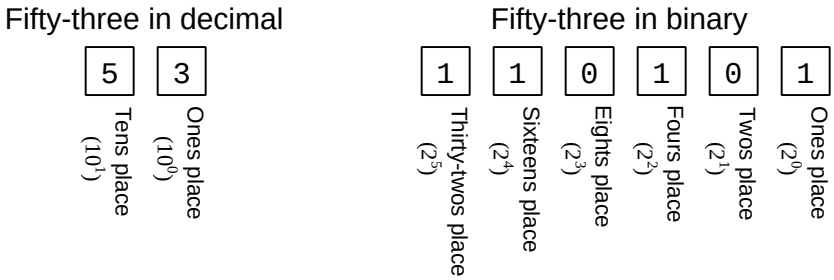


Each Exclusive-NOR gate compares a pair of respective bits in words A and B, outputting a "high" signal if those two bits are equal in state. The AND gate outputs a "high" signal only if every one of its input lines is also "high", which can only happen when the two 4-bit words are identical.

[3]For example, the American Standard Code for Information Interchange (ASCII) uses 7-bit words to represent all numerical and alphabetical characters on a standard English computer keyboard.

2.7.5 Binary decoder circuits

Binary numeration represents numerical quantities using multiple *bits*, each bit capable of being either "high" (1) or "low" (0). Basic whole-number representation in binary is simple – each bit has a place-weight that is some power of two, as opposed to *decimal* numeration where each place-weight is a power of ten. For example, compare the decimal versus binary representations of the number *fifty-three*:

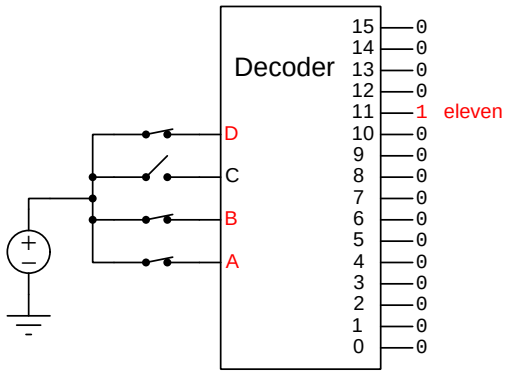


$$\text{Fifty-three} = (5 \times 10^1) + (3 \times 10^0) = 50 + 3$$

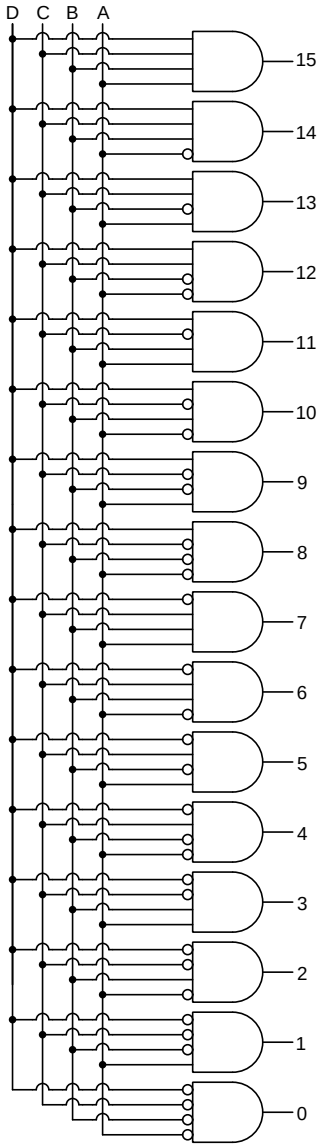
$$\text{Fifty-three} = (1 \times 2^5) + (1 \times 2^4) + (0 \times 2^3) + (1 \times 2^2) + (0 \times 2^1) + (1 \times 2^0)$$
$$\text{Fifty-three} = (1 \times 32) + (1 \times 16) + (0 \times 8) + (1 \times 4) + (0 \times 2) + (1 \times 1)$$

Modern digital computers use binary as the means of representing numbers because each bit of a binary number may be unambiguously symbolized using Boolean-state (true/false, on/off, 1/0) logic circuits.

Any circuit designed to input a binary number and activate one or more outputs selected by the value of that binary number is generally referred to as a *decoder*. For example, in the illustration below we see a 4-line to 16-line binary decoder decoding the number eleven:



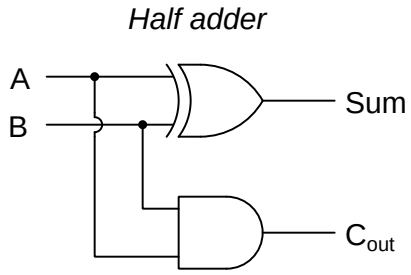
Binary decoders are really nothing more than a collection of AND and inverter (NOT) gates. In the example circuit below, inverters placed on the input of each AND gate appear as "bubbles" for the sake of compactness:



Only one of these AND gates will have its input conditions satisfied to generate a "high" output, for any given combination of bit-states in the four-bit binary input.

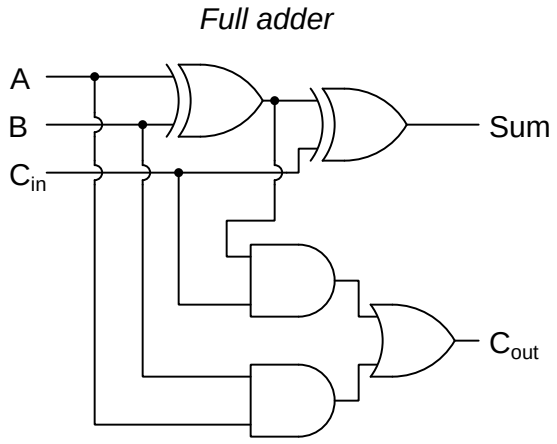
2.7.6 Binary adder circuits

Gate circuits may be built to perform simple arithmetic operations on binary numbers. Shown below is a binary *half-adder* circuit, able to add two binary bits to produce sum and a carry outputs:



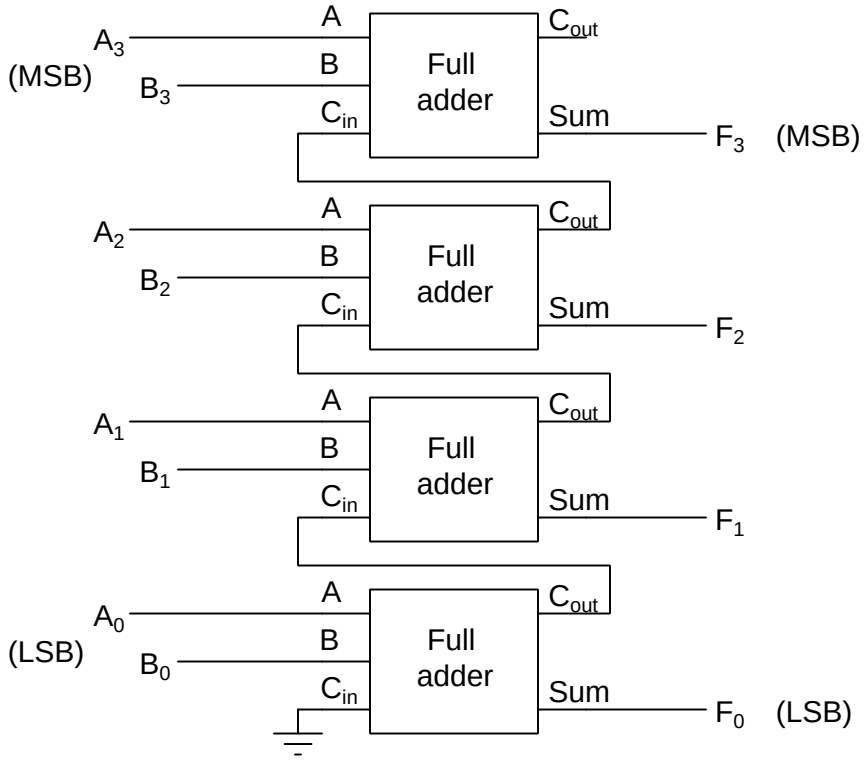
A	B	Carry out	Sum	Explanation
0	0	0	0	$0 + 0 = 0$
0	1	0	1	$0 + 1 = 1$
1	0	0	1	$1 + 0 = 1$
1	1	1	0	$1 + 1 = 2$

Next is a binary *full-adder* circuit, able to add two binary bits as well as a carry-in signal to produce sum and a carry outputs:



Carry in	A	B	Carry out	Sum	Explanation
0	0	0	0	0	$0 + 0 + 0 = 0$
0	0	1	0	1	$0 + 0 + 1 = 1$
0	1	0	0	1	$0 + 1 + 0 = 1$
0	1	1	1	0	$0 + 1 + 1 = 2$
1	0	0	0	1	$1 + 0 + 0 = 1$
1	0	1	1	0	$1 + 0 + 1 = 2$
1	1	0	1	0	$1 + 1 + 0 = 2$
1	1	1	1	1	$1 + 1 + 1 = 3$

Neither a half-adder nor a full-adder circuit is very useful on its own. In order to actually add two binary numbers of any significant magnitude together we must cascade multiple full-adder networks together. Below we see a full adder network for two 4-bit binary numbers ($A + B = F$):



Binary adders are a fundamental building-block of Arithmetic Logic Units (ALUs), which perform numerical operations on binary numbers within microprocessor systems. Most readers of this document will do so using some digital electronic device, and rest assured each and every one of those devices will contain at least one ALU!

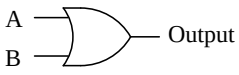
Chapter 3

Tutorial

3.1 Logic function review

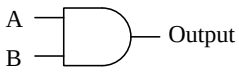
Digital logic is based on signals having one of two possible states: true or false, high or low, 1 or 0. A logic function is any system built to act on one or more input states to generate a single output state based on some set pattern. Logic functions are abstractions in the same way that mathematical functions (e.g. $y = x^2 + 5$) are abstractions. Some of the most common logic function types are listed below, along with *truth tables* showing their input/output patterns:

OR



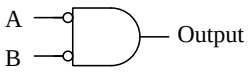
A	B	Output
0	0	0
0	1	1
1	0	1
1	1	1

AND



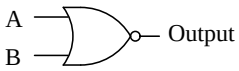
A	B	Output
0	0	0
0	1	0
1	0	0
1	1	1

Neg-AND



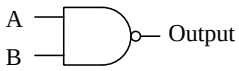
A	B	Output
0	0	1
0	1	0
1	0	0
1	1	0

NOR



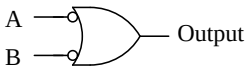
A	B	Output
0	0	1
0	1	0
1	0	0
1	1	0

NAND



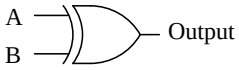
A	B	Output
0	0	1
0	1	1
1	0	1
1	1	0

Neg-OR



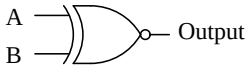
A	B	Output
0	0	1
0	1	1
1	0	1
1	1	0

XOR



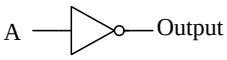
A	B	Output
0	0	0
0	1	1
1	0	1
1	1	0

XNOR



A	B	Output
0	0	1
0	1	0
1	0	0
1	1	1

NOT



A	Output
0	1
1	0

For each logic function, it is often possible to explain its behavior in more than one way. Take for instance the simple 2-input OR function. One way to describe its behavior is to say that its output will be true (i.e. 1, "high") if either one input is true *or* the other input is true. In fact, this descriptive statement is precisely where the OR function gets its name. Another functionally equivalent way of describing this same behavior, though, is to say that the output of an OR function is guaranteed to be true if *any* of the inputs are true.

When analyzing digital logic systems, it is often beneficial to think in terms of what input condition or set of conditions is sufficient to guarantee a certain output state. The following list does this for each of the logic functions previously shown:

- **OR** function – output is guaranteed to be true (1, high) if any input is true (1, high)
- **AND** function – output is guaranteed to be false (0, low) if any input is false (0, low)
- **NOT** function – output is guaranteed to be the opposite state as the input
- **XOR** function – output is guaranteed to be true (1, high) when the input states differ
- **NOR** function – output is guaranteed to be false (0, low) if any input is true (1, high)
- **NAND** function – output is guaranteed to be true (1, high) if any input is false (0, low)
- **XNOR** function – output is guaranteed to be false (0, low) when the input states differ

A close examination of the preceding logic functions reveals the fact that certain logic functions, although different in symbol and in name, actually have identical truth tables. These would be Negative-AND and NOR functions, as well as Negative-OR and NAND functions. The logical principle behind these equivalencies is *DeMorgan's Theorem*, which states that inverting all inputs of an AND or OR logic function is equivalent to inverting the output of the opposite type (e.g. adding inversion bubbles to the inputs of an OR function causes it to have the same truth table as an AND function with an inverted output).

Another interesting fact about logic functions is that some of them are inherently universal, which means you can build up any function using combinations of that one function type. NOR and NAND functions are both universal in this way. The key to this universality is the fact that both NOR and NAND functions may be used as inverters (i.e. NOT functions). A NOR function may be used as an inverter simply by connecting all of its inputs together to be a single input, or by making all unused inputs false (0, low). A NAND function becomes an inverter by either using all its inputs as a single input (just like the NOR function), or by making all unused inputs true (1, high). Then, by combining one or more of these NAND- or NOR-based inverters together with other NAND or NOR gates you can derive other types of functions by exploiting DeMorgan's Theorem.

Just as it is possible to build electric circuits, metal mechanisms, fluid networks, or any other physical system to implement certain mathematical functions, there are a variety of ways we might turn these abstract logic functions into physical reality. Throughout the rest of this tutorial we will explore how to do this using solid-state transistor circuitry, with the transistors used a on/off switching elements.

3.2 Bicycle headlamp alarm

An effective way to introduce new concepts is by practical example, and so we will now focus on a specific application for which an electronic logic gate fulfills a practical purpose.

As every bicycle commuter knows, the greatest danger of bicycling comes from sharing the road with automobiles. For this reason, some bicyclists ride with their headlamps turned on during daytime hours in order to be better seen by automobile drivers. Riding during the daytime with one's headlamp turned on poses another problem, however: the possibility of forgetting to turn off the headlamp once the bicycle is parked.

Suppose we wished to augment a bicycle headlamp with an alarm, to signal the rider that they left their headlight turned on when parking the bicycle. In order for such an alarm to work, it must be aware of both the headlamp's status (on or off) and the rider's status (riding or not), and then use some logical function to activate the alarm when the bicycle's headlamp is on *and* the bicycle is not being ridden. An electronic *logic gate* is a sensible option for this application, being an integrated semiconductor circuit designed to receive voltage signals as inputs while outputting a voltage signal representing the logic function's determination.

The headlamp's status is easy enough to electrically sense: the voltage across the lamp terminals serves nicely as a signal for one of the logic gate's inputs. The rider's status is a bit more challenging, requiring some form of sensor to detect the rider's presence on the bicycle and to represent that presence as a voltage signal. Pressure-activated switch contacts[1] installed at points where the rider's body touches the bicycle will suffice, a switch installed on the bicycle's handlebar to signal when the handlebar is being gripped, and a switch on the saddle to signal when the rider is seated. Now we need to determine which combination(s) of statuses should trigger the headlamp alarm.

[1] A suitable sensor for this application is a *force-sensitive resistor*, or *FSR*. These are pressure-sensitive plastic strips with two terminals, the electrical resistance between those terminals decreasing as force is applied perpendicular to (i.e. squeezing) the strip.

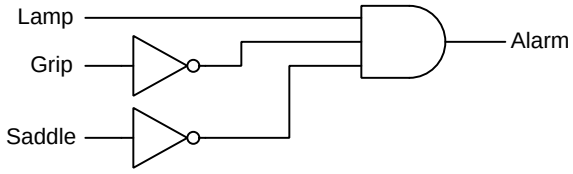
The following truth table shows our three logic gate input signals (lamp, grip, saddle) and the desired state of the alarm. With three discrete inputs, each one having just two possible states, the truth table will have eight rows to represent all eight possible combinations ($2^3 = 8$) of input states:

Lamp status	**Grip status**	**Saddle status**	**Alarm status**
Off	Untouched	Unseated	off
Off	Untouched	Seated	off
Off	Touched	Unseated	off
Off	Touched	Seated	off
On	Untouched	Unseated	ON
On	Untouched	Seated	off
On	Touched	Unseated	off
On	Touched	Seated	off

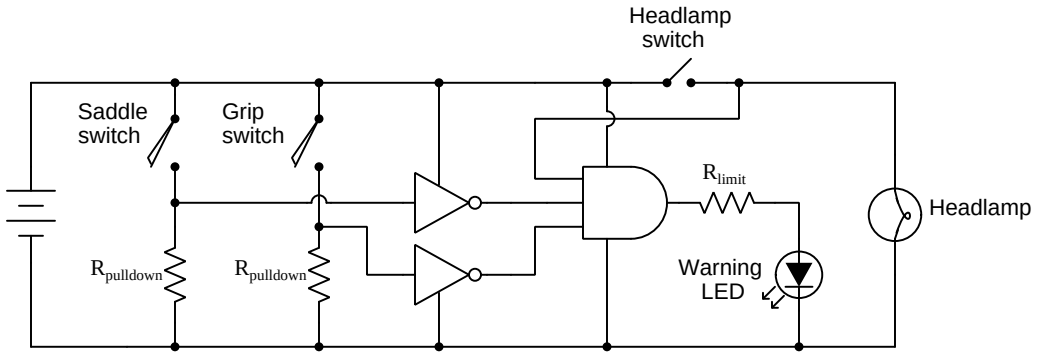
Re-writing this table using Boolean quantities (e.g. 0 = False and 1 = True):

Lamp	**Grip**	**Saddle**	**Alarm**
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	0

Our desire is to have the alarm activate if the lamp is on *and* the grip released *and* the saddle unoccupied. From this simple statement we can see that an AND function will be necessary. However, since the desired "1" output state occurs when both the grip and saddle sensors are in their "0" states, we must *invert* those inputs prior to sending them to the AND function. The following logic diagram shows how these functions will interconnect:



Now that we have a clear idea of the necessary logic functions, we may begin designing a logic *circuit* to implement that function. Logic *gates* are integrated circuits designed to receive and output voltage signals, zero voltage representing a "0" state and full supply voltage representing a "1" state. Each logic gate, in addition to having input and output terminals for the electrical logic signals, also has two terminals for connection to a DC power source. Since this circuit will be installed on a bicycle, it makes sense to use the headlamp's battery as the DC power source for the logic circuit. A complete schematic diagram is shown here:



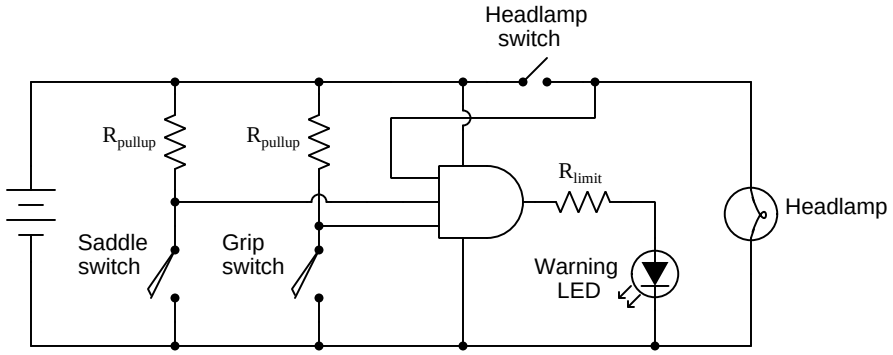
All gate symbols in this diagram are drawn such that input terminals are on the left and output terminals are on the right. Top and bottom gate terminals signify power supply connections. Note how the two inverter (NOT) gates' power connections are drawn: the upper gate with only the connection to (+) shown, and the lower gate with only the connection to (−) shown. This is typical when multiple gates are contained within one integrated circuit, because the IC as a whole only has two power terminals for all gates contained within.

Note also the two *pulldown* resistors, necessary to provide a reliable "low" (0) logic state whenever the respective switch opens. Without these pulldown resistors in place, the inverter gates' input terminals would be left in an electrically *floating* condition (i.e. connected to nothing), which may result in the gate receiving false signals due to static electric charges or other interference. These resistors' values are identical, sized large enough that they do not waste undue energy when the switch is closed and sized small enough to rapidly drain off any static energization from the intrinsic capacitance of the gate's input when the switch is open.

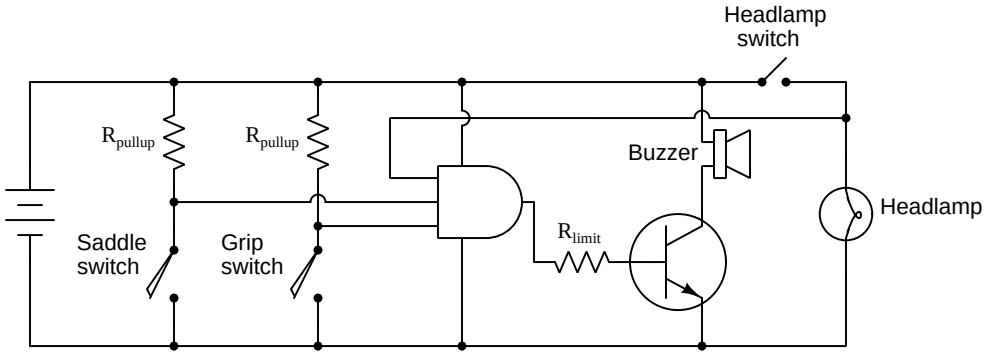
The current-limiting resistor connected to the output terminal of the AND gate simply limits current to the LED alarm indicator, and is sized appropriately to allow the LED to receive its rated voltage at its rated current given the battery's nominal voltage.

With this, we have a functioning headlamp alarm circuit. The alarm LED will energize only if the headlamp is on *and* the saddle is unoccupied *and* the grip is released.

A good design principle is to eliminate unnecessary components, as this makes the system more reliable and (generally) dissipate less energy. This is possible if we think carefully about the states of all inputs to the AND gate. Recall that the point of the AND gate is to trigger the alarm whenever the lamp is on and the grip is released and the saddle is unoccupied. If we could re-wire the switches in such a way that a released grip and unoccupied saddle resulted in "1" states rather than "0" states, we could dispense with the two inverters and just use a single AND gate. This is certainly possible, and all it requires is shifting the locations of the two switches, using their resistors to *pull up* the logic states to "1" when the contacts open rather than to *pull down* the logic states to a "0" when the switches are at rest:

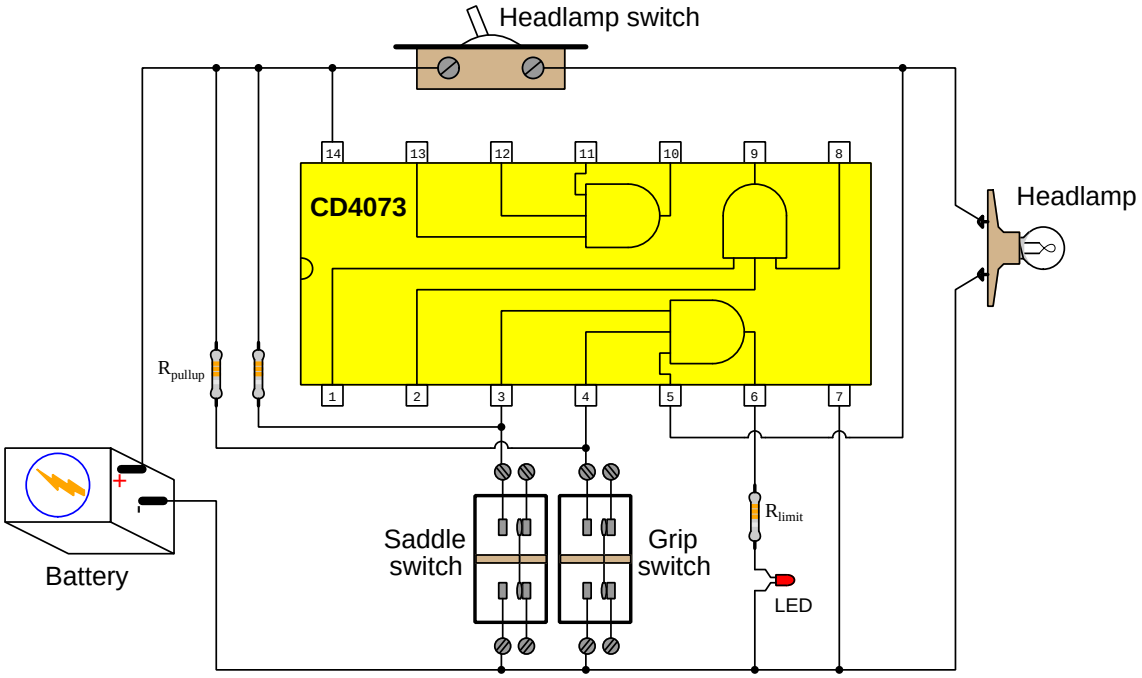


Like most integrated circuits, electronic logic gates are typically quite limited in output current. A typical AND gate might only be capable of driving a single LED (approximately 20 mA of current), and not much else. If we need to drive a higher-power load, we must use a power transistor or some other interposing component between the gate's output terminal and the load. An example of this is shown below, allowing the alarm to use an audible buzzer rather than an LED:



Now the current-limiting resistor needs to be sized to let just enough current through the transistor's based terminal to saturate it when the gate's output goes to a "high" (1) state. If we were to use a power MOSFET instead of a bipolar junction transistor, there would be no need for a current-limiting resistor at all, and the gate output current would be reduced to essentially zero.

In this pictorial diagram we see one possible implementation of this alarm circuit using a model CD4073 CMOS integrated logic circuit:



A practical detail not shown in this illustration is that the input terminals for the two un-used gates within the CD4073 IC should be connected either to +V or Ground rather than left "floating" (i.e. not connected to anything, as shown)[2]. Those un-used AND gates have no effect on the operation of our headlamp alarm, of course, but securing their input terminals to well-defined "high" or "low" states is important for the health of the IC in general.

[2]Ensuring well-defined "high" or "low" logic states for *all* gate inputs is especially important for CMOS circuits whose high-impedance inputs may assume intermediate states in the presence of nearby static-electric charges. For example, with a 5 Volt DC power supply, a gate input left "floating" may assume any potential at all when influenced by nearby static charges, and if that potential happens to drift mid-way between 0 and 5 Volts with reference to ground it may cause the complementary N- and P-channel MOSFETs within that gate to simultaneously turn on instead of being in complementary states (i.e. one on and the other off). This will waste energy and possibly even harm the IC! Another reason for connecting unused input terminals to one of the two DC power "rail" points is to guard against IC damage caused by accidental electrostatic discharge. Unused gate *output* terminals, however, should always be left floating so as to allow them to assume any logic state dictated by the respective gates' inputs.

3.3 Logic gate limitations

This bicycle headlamp alarm circuit is just one example of how electronic logic gates may be used to perform practical functions. For any digital logic design application, a number of concerns must be addressed when selecting logic gates, some of which are listed here:

- **DC supply voltage requirements** – electronic logic gates are semiconductor circuits, designed to operate over a specified range of supply voltages. Some gates have very narrow supply voltage limits while others are quite wide. For our bicycle headlamp alarm circuit, we would need to ensure the AND gate would function for any reasonable voltage we would expect the headlamp battery to output under typical conditions.

- **Logic voltage levels** – the ideal "high" (1) state is full supply voltage (measured with reference to power supply ground), while an ideal "low" (0) state is zero voltage, but in some circuits it may not be possible to provide ideal voltage levels to gate inputs. For example, if our pressure-sensitive switches exhibited finite resistance values instead of zero resistance when closed and infinite resistance when open, the voltage levels for those two inputs would be something greater than zero when "low" and less than supply voltage when "high", a possible result being unpredictable behavior from the gate. Careful analysis of the input circuitry and comparison against the gate's advertised tolerances will reveal potential problems. Every "family" of electronic logic gate has its own standards for minimum and maximum voltage levels for high and low states expected at output terminals (V_{OH} and V_{OL}), as well as minimum and maximum voltage levels acceptable as high and low states at input terminals (V_{IH} and V_{IL}). The *Derivations and Technical References* chapter contains sections listing acceptable voltage levels for digital logic states.

- **Input terminal current** – an ideal logic gate's inputs are like the inputs of an ideal operational amplifier: they draw zero current. Real logic gates exhibit some finite (albeit small) amount of current essential to the operation of the gate's internal transistor circuitry, and this input current may affect the voltage levels. For any application where pullup or pulldown resistors are used to provide a default logic state to a gate's input(s), the resistors must be sized such that they may carry this input current while still providing a voltage level within the gate's acceptable limits.

- **Output terminal current** – as mentioned previously, electronic logic gates are limited in the amount of current they may handle at their output terminals. If a gate's output current limitations is less the current required by the driven load, some form of interposing is necessary between the gate and the load.

- **Current direction** – the direction(s) of current through signal terminals on an electronic gate are not consistent between different models of gate. A terminal "sources" current when the direction (conventional) flows out of the gate toward the external world, and "sinks" current when the conventional-flow current flows into the gate from the outside world[3]. Some gate outputs have the ability to both source and sink current, while others can only sink current. Legacy TTL logic gate inputs always source current and never sink. When the output of

[3] An engineer once described this to me using the colorful terms "blowing" and "sucking" current, respectively.

one gate circuit drives the input of another, the roles of sourcing and sinking must always be complementary: one gate sources while the other sinks.

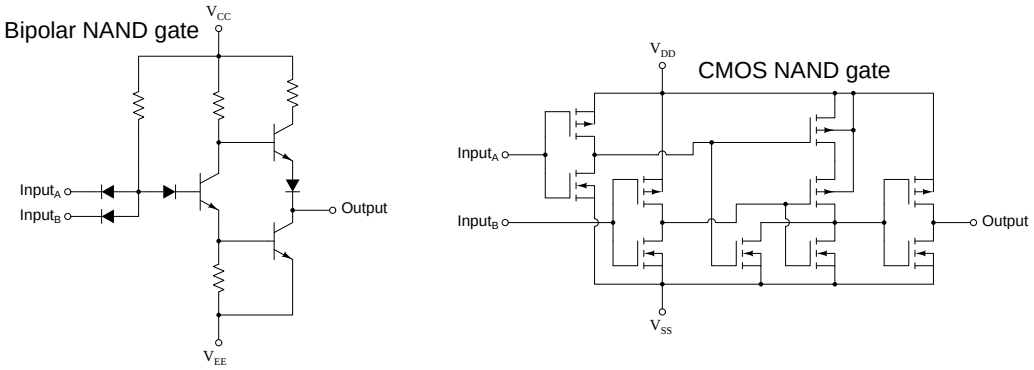
- **Switching speed** – though not a concern in our example headlamp alarm circuit, logic gates are limited in the speed at which their output states may change. This parameter is analogous to the *slew rate* limitations of an amplifier, where the amplifier's output signal is limited in its rising and falling speeds, and is typically expressed as a delay time called *propagation delay*: the amount of time required for an input state-change to propagate through the circuit and manifest as an output state-change.

- **Number of inputs, and number of gates per IC** – every logic gate has a limited number of input terminals. AND gates, for example, are commonly available with 2 inputs, 3 inputs, or 4 inputs. If more are needed, or perhaps just more than offered by the gates on hand, it may be necessary to combine multiple gates to achieve the required number of inputs. For example, with our bicycle headlamp alarm circuit which needed a three-input AND function, we could have used two 2-input AND gates, the output of one gate feeding the input of the other.

3.4 TTL versus CMOS logic

Semiconductor logic gates may be built from bipolar junction transistors (BJTs) or field-effect transistors (MOSFETs) – sometimes referred to as *bipolar* (also known as *TTL*) and *CMOS*[4], respectively. Some logic gates incorporate both types of transistor in one integrated circuit called *BiCMOS*.

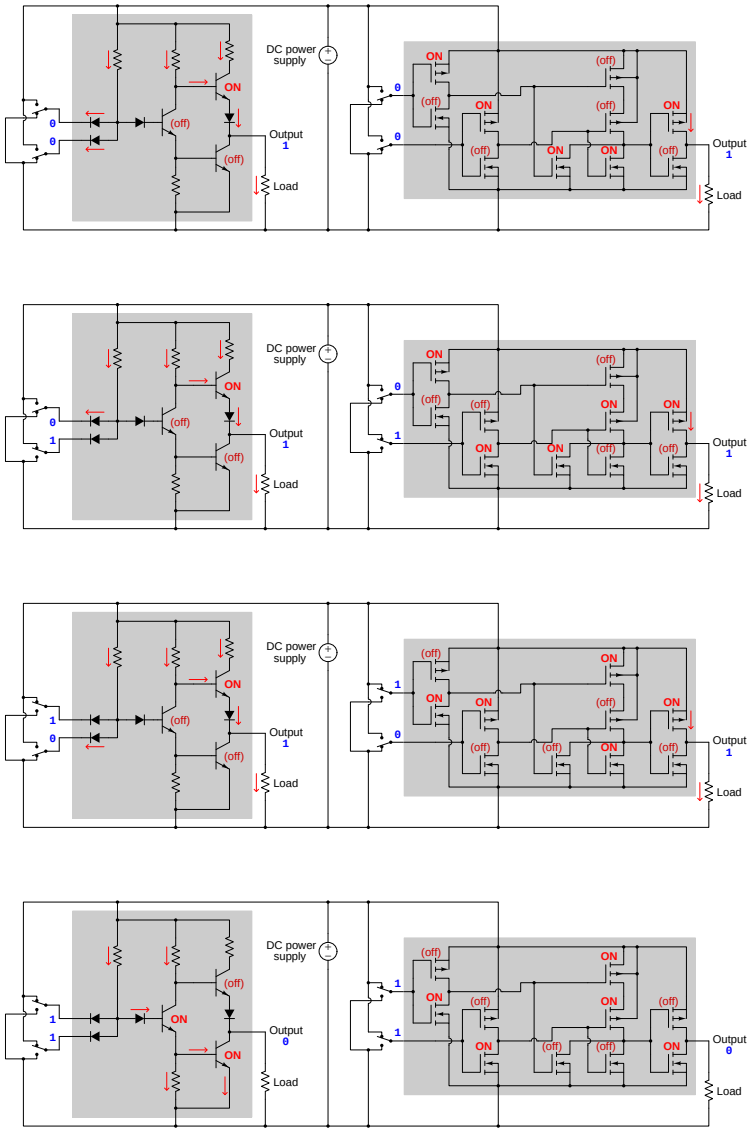
The following schematic diagrams contrast basic bipolar versus CMOS implementations of a two-input NAND gate. Note how the bipolar gate circuit uses resistors, diodes, and NPN transistors while the CMOS gate circuit uses nothing but N-channel and P-channel enhancement-type MOSFETs:



Note also the DC power supply terminals, and how they are labeled. On the bipolar gate the positive power supply terminal is labeled V_{CC} because it is on the side of every transistor's *collector* terminal, and the negative terminal is labeled V_{EE} because it is on the side of every transistor's *emitter* terminal. Labeling of power supply terminals on the CMOS gate is a little less consistent: following the same pattern, V_{DD} (+) ought to mean on the side of the transistors' *drain* terminals and V_{SS} (−) ought to mean on the side of the transistors' *source* terminals, but this only makes sense for the gate's N-channel transistors. This is a case where a common convention doesn't make as much sense as it ought to, but nevertheless V_{DD} universally represents the positive (+) power supply terminal and V_{SS} the negative (−) terminal for CMOS integrated circuits.

[4]This acronym stands for Complementary Metal-Oxide-Semiconductor, because the circuitry is based on complementary pairs of N-channel and P-channel MOSFETs.

Below we see schematic diagrams showing the bipolar and CMOS NAND gates in all four possible combinations of input states, showing how their internal transistors turn on and off with these states to produce the correct output state for a NAND function[5]. It is highly recommended as an active reading strategy to closely examine each of these diagrams and prove to yourself why each of the gates' transistors are in their labeled states:



The only transistor passing current in the CMOS gate is the P-channel output transistor, and only when driving a "high" (1) state to the load. None of the other MOSFETs carry current, since

[5] A NAND function outputs a "high" (1) state if *any* of its inputs are "low" (0).

one transistor in each P/N-channel pair will always be off. Bipolar gate circuits by contrast *always* draw some current regardless of logic state. This explains one of the major differences between bipolar and CMOS logic gates, being the *quiescent* current consumption, which refers to the amount of current drawn from the DC power supply when the gate is holding in any one logical state. CMOS logic gates draw current only when *transitioning* between "low" and "high" states, because only during those brief time periods will you ever find two complementary transistors partially on. CMOS power dissipation, therefore, is a function of signal *frequency*. For a "static" application such as our bicycle headlamp alarm circuit where input states remain stable for long periods of time, CMOS logic power consumption is negligible.

Another important difference between bipolar and CMOS logic gate circuitry is allowable DC power supply voltage. Bipolar gate circuits typically operate with a 5.0 Volt DC power supply, with narrow limits of tolerance (generally $\pm$ 0.25 Volt DC), while CMOS gate circuits tolerate a much wider range of power supply voltages. This favors CMOS for our hypothetical bicycle headlamp circuit. Recent developments in digital gate technology have produced several "families" of CMOS logic designed to operate reliably on ever-smaller supply voltages in order to serve popular demand for battery-powered digital devices such as cellular telephones and computer "tablets" which must function under conditions of waning battery voltage.

Bipolar logic gates historically outpaced CMOS in terms of switching speed, although developments in CMOS technology have all but closed this gap. One of the reasons why a CMOS gate circuit might be slower to switch states than a bipolar gate is the fact that each MOSFET exhibits capacitance between the metal gate terminal and the semiconductor channel, and the stored electric charge within this capacitance must be drained and replenished in order to make the transistor switch states. Any series resistance in a CMOS logic gate's input path (e.g. the "on" resistance of the output transistor in the gate driving that input) therefore creates a resistor-capacitor (RC) time delay. BJTs operate on a completely different principle, and so this limitation does not apply.

Bipolar gates also enjoy greater tolerance to static electricity than CMOS. The extremely high voltage levels generated by feet scuffing on carpet or fabric rubbing together is not just a nuisance on dry days (causing small electric shocks to the person), but these stored charges have the ability to easily destroy integrated circuits, especially circuits containing MOSFETs. MOSFET gates are insulated from their channels by a microscopically thin layer of metal oxide, and the extreme thinness of this oxide layer gives it a relatively low breakdown voltage. A standard precaution when working with any digital logic ICs (especially CMOS) is to wear an anti-static wristband which connects the person to earth ground through a flexible wire cord through a high resistance[6]. Many CMOS ICs come equipped with protection diodes internally connected to the input terminals, which clamp the input terminal voltage to a range within the DC power supply's voltage, but this is no guarantee of protection against electrostatic discharge (ESD).

Another important difference between bipolar and CMOS logic is the default logical state of any unconnected (floating) input terminal. A "low" state on any bipolar gate input terminal is achieved by some device sinking current to ground from that input terminal, effectively "steering" bias current

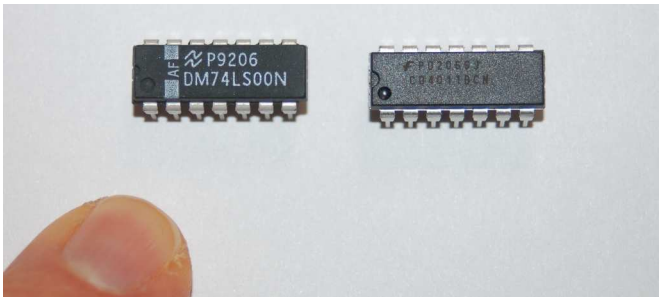
[6] While a direct "electrically common" connection would work just as well to drain static electric charges, it would also pose an elevated electric shock hazard since any contact made with a "live" terminal would result in current passing through the person's arm due to the wristband. This is why anti-static wristbands always have a resistor (in the mega-Ohm range) in series with the grounding wire, so that the connection to ground is high enough resistance to severely limit current in the event of accidental contact with a "live" circuit. Such a resistance, even millions of Ohms, is still low enough to drain static electric charges very rapidly because the capacitance of the human body is so very low, and therefore the RC time constant (τ) is short.

away from the base of the first transistor in that gate. An unconnected input terminal cannot either source or sink current, which means all of the bias resistor's current reaches the first transistor's base terminal to turn that transistor on. This means unconnected bipolar gate inputs always assume a logical "high" state. CMOS gates behave quite differently because there no current needs to be sourced or sunk at any input terminal, ever. When a CMOS gate input is left unconnected, its logical state is *random*. This is why we typically find either pullup or pulldown resistors used at CMOS gate inputs when driven by a SPST switch contact: to provide a reliable default logic state when that switch contact is open.

To summarize, here are some important differences between bipolar (TTL) and CMOS logic circuits:

- Quiescent-state power supply current is negligible for CMOS, but substantial for bipolar
- Power dissipation is a function of signal frequency for CMOS, but is relatively steady for bipolar
- CMOS logic can tolerate a wider range of power supply voltage than bipolar
- Bipolar logic has traditionally enjoyed faster switching speed than CMOS
- Bipolar logic tolerates higher levels of electrostatic discharge (ESD) than CMOS
- Unconnected inputs always go to a "high" logic state for bipolar, but are random and unreliable for CMOS

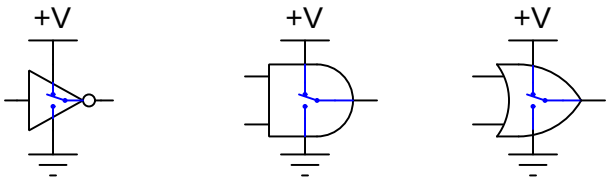
TTL and CMOS logic gates are visually indistinguishable when packaged in integrated circuit (IC) form, and may be identified only by part number. Here we see two physically identical 14-pin Dual Inline Package (DIP) ICs situated next to each other, the left IC being a model 74LS00 and the right IC being a CD4011, both quad 2-input NAND gate ICs:



Additional prefix or suffix characters attached to the base part number reflect manufacturer codes and sub-models which may differ in terms of operating temperature range or other performance parameters.

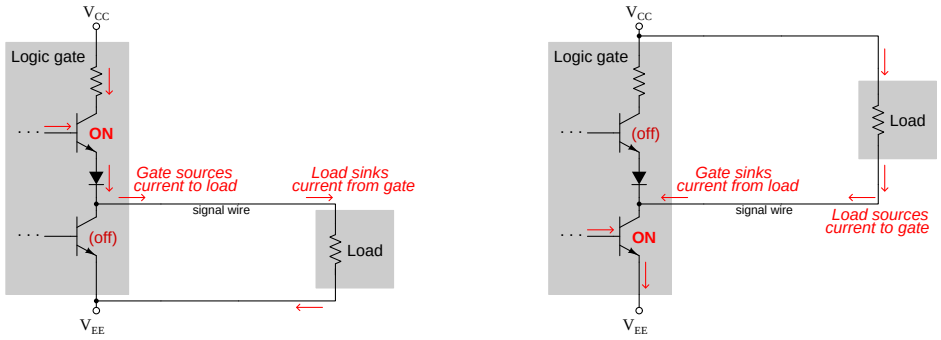
3.5 Logic gate currents

Semiconductor logic gates are fundamentally *amplifier circuits* which take in voltage signals and output voltage signals in response to those inputs. Being digital devices, the output signals consist of the discrete values of either *zero* or *full* (DC supply) voltage. A helpful "mental model" of a logic gate's output – regardless of its particular logical *function* (e.g. NOT, AND, OR) – is that of a SPDT switch where the output terminal either directly connects to the IC's ground terminal or to its +V terminal:



In the illustration shown above, each of the logic gates is outputting a "high" signal, where the output terminal is connected to the +V DC power rail terminal by means of a transistor inside the IC. When a gate's output switches to a "low" state, this analogy is of the SPDT switch flipping to the downward position so as to connect the output terminal to the Ground power rail terminal instead. In reality there is no mechanical switch inside of a semiconductor logic gates, but typically a pair of transistors where one is always turned on and the other always turned off.

An important concept common to all forms of digital logic circuitry is the notion of current *sourcing* and *sinking*. This simply refers to the direction of current through any terminal of a logic gate, input or output. Simply defined, sourcing refers to (conventional flow notation) current *exiting* the signal terminal of a device, while sinking refers to current *entering* the signal terminal of a device. The following (partial) schematic diagrams illustrate these concepts by showing the final output stage of a bipolar logic gate IC driving different loads in different states:

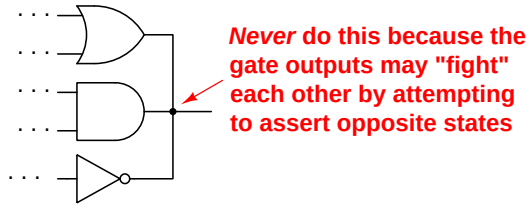


Another way commonly used to describe direction of terminal current for an integrated circuit (IC) is to assign mathematical signs to the current values, following a standard called *passive sign convention*. "Passive sign convention" uses positive values to denote current *sinking* and negative values to denote current *sourcing*. If you see a *negative* current value specified in a logic IC datasheet, it means that current is being *sourced* by the IC terminal in question.

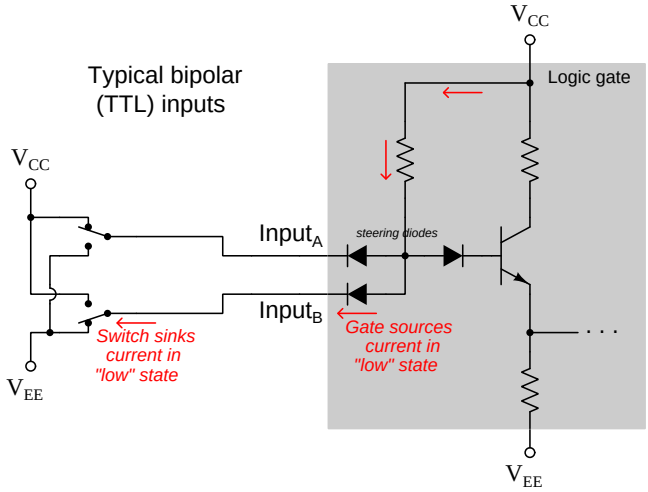
Note how in each of the circuits shown here, one device sources current while the other sinks current. It is impossible to have both gate and load sourcing current to each other, or both sinking current from each other, because the current directions would mutually contradict. Also note how these two different configurations result in the load being energized under different gate output states: when the gate sources current to a sinking load, the load energizes for a "high" (1) gate output state; when the gate sinks current from a sourcing load, the load energizes for a "low" (0) gate output state. Since the energization state of the load is what ultimately matters in a logic circuit, the ability to choose between these two configurations provides a simple means of logical inversion, alternative to adding a NOT function to the output of a gate. This is similar to what we did on the bicycle headlamp circuit design by swapping positions of the input switches and their respective resistors: we inverted the switches' logical states by having them sink current rather than source current to their resistors – this simple component swap obviated the need for two NOT gates, making a simpler and more reliable circuit.

A common term used to describe the two "stacked" transistors in the output stage of a logic gate is *totem pole*, analogous to a *push-pull* power stage in an analog amplifier circuit.

Multiple gates with totem-pole output stages should *never* have their outputs connected together, because if two of these connected gates ever tried to output opposite logic states, they would essentially short-circuit each other as they "fought" one another to assert their output state at that electrically common point:



Logic gate *inputs* also source and/or sink current, and this is important to understand in order to properly pair logic gate inputs with devices generating logic signals (e.g. switches, gate outputs). The bipolar gate example shown earlier is typical of the *TTL* (Transistor-to-Transistor Logic) logic family, which uses "steering diodes[7]" to direct bias current away from the base terminal of the input transistor when the respective input is in a "low" state.



As this diagram shows, a TTL gate's input sources current to any input device in a "low" state (that input device sinking current to the V_{EE} power supply rail[8]), but "high" input states result in no current. In other words, TTL gate inputs naturally "float" to a high state. CMOS gate inputs behave very differently: since MOSFET gate terminals are electrically insulated from the rest of the transistor structure and therefore cannot conduct a continuous current, a CMOS gate's input terminals neither source nor sink current in steady-state conditions. The only time an electric current passes through a CMOS gate's terminals is when the logic state *changes* and the MOSFETs' gate capacitance momentarily absorbs or releases energy (i.e. the gate-to-channel capacitance's electric field strength either grows or shrinks). Therefore, any device sending a *pulsing* signal to the input terminal of a CMOS logic gate must alternately source and sink transient[9] currents.

The lack of a definite logic state for a floating CMOS input is why all unused inputs on a CMOS integrated circuit must be connected to one of the power supply terminals. Leaving a CMOS input floating is an unacceptable design practice for two reasons: (1) the gate may behave randomly because the floating inputs may attain either a "high" or "low" state by means of stray electric fields from surrounding objects, and (2) a floating input may result in the complementary N-channel and P-channel MOSFET pair being partially on, unnecessarily drawing current from the power supply and dissipating excessive heat within the IC.

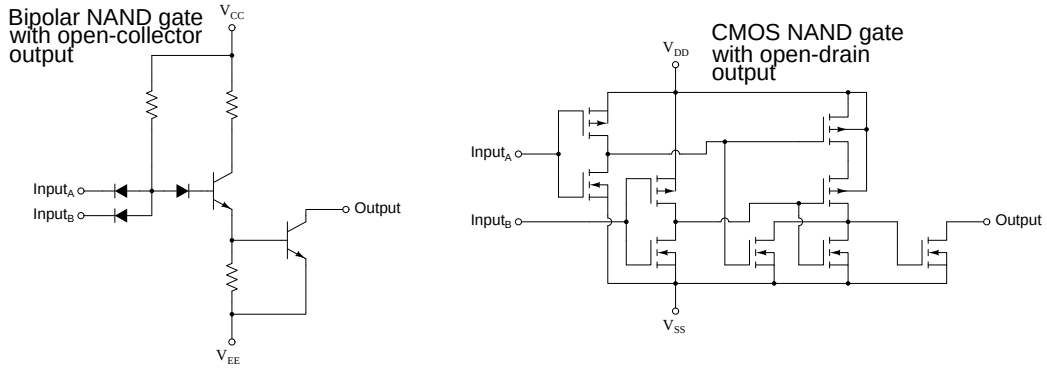
[7] The internal schematic diagrams of TTL logic gates often show these steering diode arrays as *transistors* because the back-to-back PN junctions required to form such an array is easier to build on the integrated circuit substrate as a transistor. This can be very confusing for anyone new to the internals of a TTL logic gate, as the "transistor" serves no amplification purpose.

[8] The "rails" or "busses" of a DC power supply refer to the + and − connections feeding electrical power to all components.

[9] A *transient* event is one that is momentary rather than continuous.

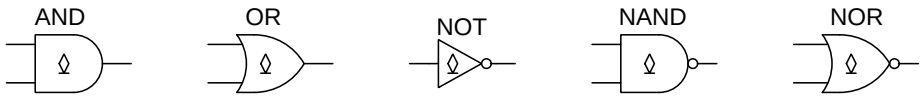
3.6 Open collector/drain logic gates

Some logic gate outputs can *only* sink current and not source because their output stage is missing a sourcing transistor. Bipolar gates of this design are called *open-collector* while CMOS gates of this design are called *open-drain*. The following schematic diagrams show examples of bipolar and CMOS NAND gates with such output stages:

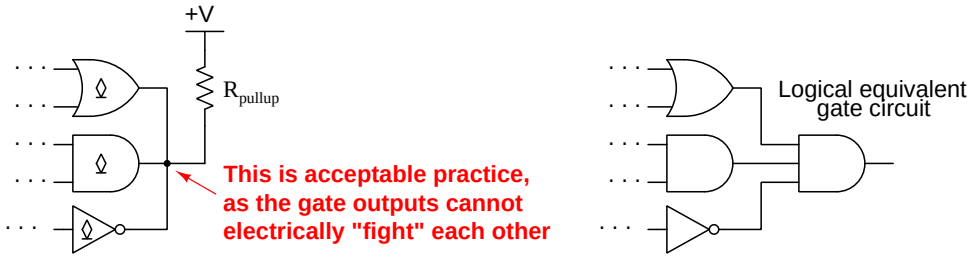


If you compare these schematic diagrams against those of NAND gates with "totem-pole" outputs, you will see the absence of an upper transistor on the output terminal to provide a sourcing current path. The "high" signal state for either gate is really a *floating* condition, which means an external *pullup* resistor is necessary to provide a definite voltage level for that "high" state.

Open-collector and open-drain logic gates are identified by a special symbol, resembling a diamond shape resting on a horizontal line segment:



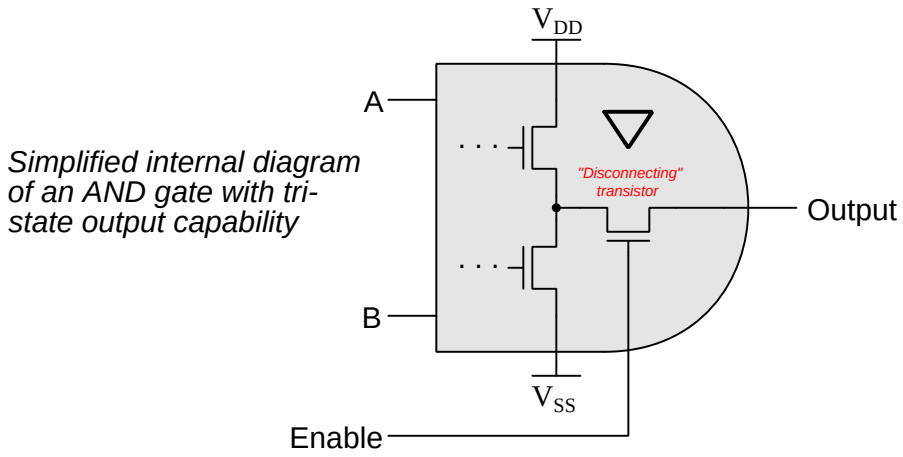
An interesting capability of open-collector and open-drain gate outputs is that they may be safely paralleled with no risk of interference or damage, because they are only capable of sinking current and not sourcing current. Doing so creates the equivalent of an AND function between the connected gates, because *any* "low" state from a gate output guarantees a "low" state for that electrically common point. This is sometimes referred to as a *wired-AND* function:



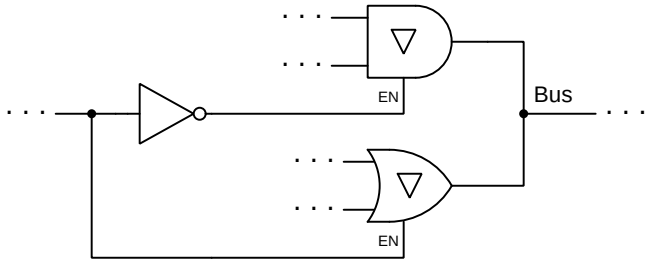
3.7 Tri-state output logic gates

Another solution to the problem of connected logic gate outputs "fighting" each other is an added feature found in some logic gates called *tri-state outputs*. A logic gate with tri-state output capability has an additional input line controlling a transistor "disconnect" between the internal totem-pole transistor stage and the output terminal. When disabled, the gate assumes a *high-impedance* or *high-Z* mode whereby the output terminal "floats" and can no longer source or sink current. When enabled, a tri-state gate behaves like any other logic gate of its functional type.

An inverted triangle symbol denotes a logic gate capable of tri-state operation. An additional "enable" line on the gate is another indication:



In the following example, an AND gate and an OR gate both drive signals to a common line called a *bus*. Their respective enable lines are controlled by a digital signal and its complement, to ensure only one of the two gates will ever be permitted to drive a high or low signal to the bus:

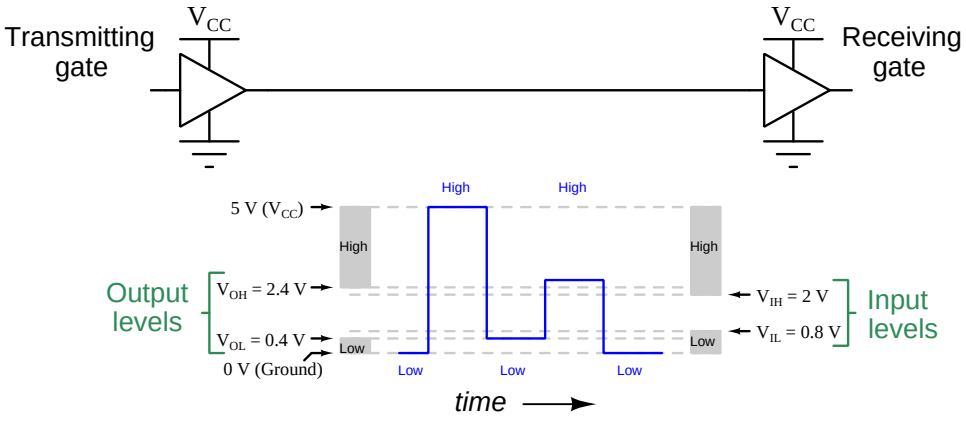


Digital electronic computer designs are largely *bus-oriented*, which means they contain many such "bus" lines shared one at a time by various logic gate outputs, much as a radio channel may be used by one person (talking) at any given time. This is why the potential problem of multiple logic gates "fighting" one another with opposing logic states is often referred to as *bus contention*: in a poorly-designed system the gates will literally contend with one another for control of the common bus. In complex digital systems where several or more gates connect to a common bus, the strategy employed to select just one of those gates at a time is called *bus arbitration*.

3.8 Logic levels

An expression sometimes heard among digital circuit designers is *"There is no such thing as a digital signal, just funny-looking analog signals"*. With digital logic what we're forced to do is take a fundamentally analog medium (e.g. DC voltage signals) and impose an arbitrary mask defining some voltage levels as "high" and others as "low". In reality, digital logic gates are really just high-gain analog amplifiers, driving themselves into cutoff or saturation when receiving "valid" input signals.

Standardized threshold voltages[10] defining "high" and "low" states makes digital signaling possible in a world where signal voltages may assume any value along a continuous (analog) range. Conservative design of logic circuits ensures *transmitted* voltage levels always surpass the minimum expectations for *received* signals, the difference between the transmitted and received thresholds called the *noise margin*. The following illustration shows a TTL logic gate transmitting a low-high-low-high-low pulse sequence to another TTL logic gate receiving that signal, where the data is properly interpreted even though the voltage levels for "high" and "low" states vary from ideal (full rail) to barely acceptable:



Noise margin may be easily understood by analogy to human speech: when we talk to another person we should always ensure we are speaking with enough volume to be clearly heard. If one person speaks at a volume just loud enough for the listener to barely hear the sounds, there is no margin of safety in the event of noise: even the slightest amount of noise introduced to the conversation may render accurate listening impossible. Therefore, it is courteous to raise one's voice to a level *slightly louder* than what is necessary for the listener to properly hear us, providing a *margin* large enough to compensate for unexpected noise and thereby facilitate reliable communication.

[10] Voltage level thresholds defining "high" and "low" logic states are well-defined for both TTL and CMOS logic gates operating on a 5 Volt DC supply. Refer to section 5.1 on page 72 for a discussion on 5 Volt TTL logic levels, and to section 5.2 on page 73 for a discussion on 5 Volt CMOS logic levels.

3.9 Voltage level translation

All voltage signals are inherently analog, which means they are capable of continuous variation. The "high" and "low" states of digital logic circuits are really just determinations based on voltage signals either being higher than some standardized upper threshold value or lower than some lower threshold value, respectively. What fundamentally defines these threshold values is the design of the transistor circuits inside digital logic gates, those design details dictating the amount of voltage required to saturate transistors in their fully-on states or cause them to completely turn off.

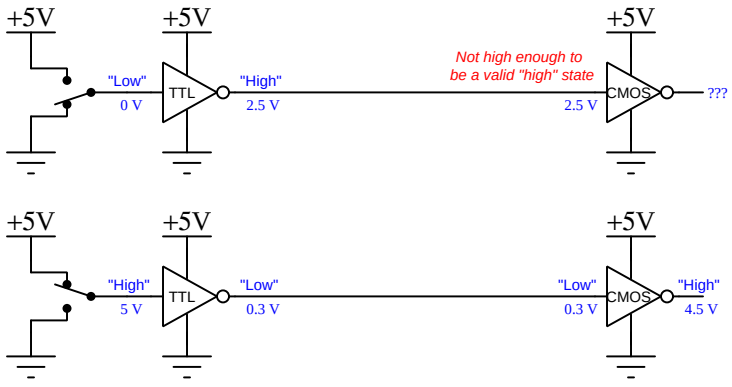
Not only does the magnitude of a logic gate's power supply voltage influence these threshold values, but so does the design "family" of that gate. For example, a classic TTL logic gate using bipolar junction transistors operating on a 5 Volt DC power supply exhibits the following logic levels defining "high" and "low" states:

Logic state	Guaranteed output	Acceptable input	Noise margin
High	$V_{OH} = 2.4$ Volts min.	$V_{IH} = 2.0$ Volts min.	$2.4 - 2.0 = 0.4$ Volts
Low	$V_{OL} = 0.4$ Volts max.	$V_{IL} = 0.8$ Volts max.	$0.8 - 0.4 = 0.4$ Volts

By contrast, a CMOS logic gate using MOSFET transistors operating on the same 5 Volt DC power supply expects different amounts of voltage for its "high" and "low" states:

Logic state	Guaranteed output	Acceptable input	Noise margin
High	$V_{OH} = 4.44$ Volts min.	$V_{IH} = 3.5$ Volts min.	$4.44 - 3.5 = 0.94$ Volts
Low	$V_{OL} = 0.5$ Volts max.	$V_{IL} = 1.5$ Volts max.	$1.5 - 0.5 = 1.0$ Volt

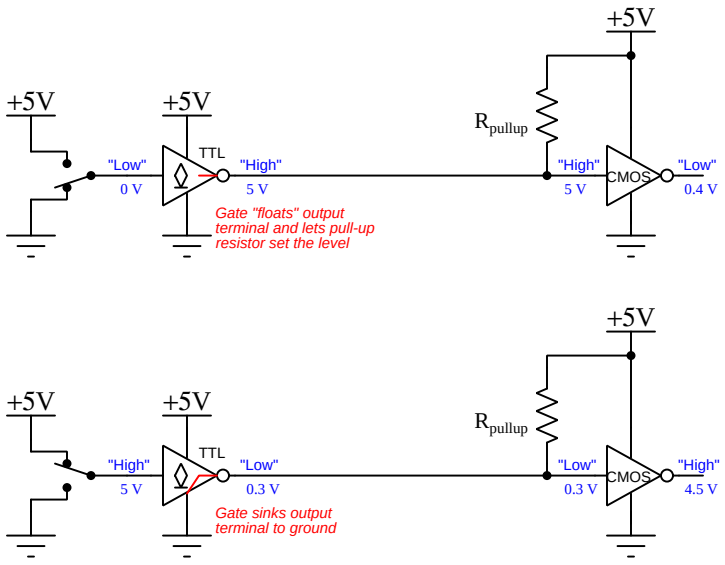
A close comparison of the threshold values for these two logic families reveals some incompatibilities. For example, consider a TTL gate driving the input of a CMOS gate, both operating on the same 5 VDC supply:



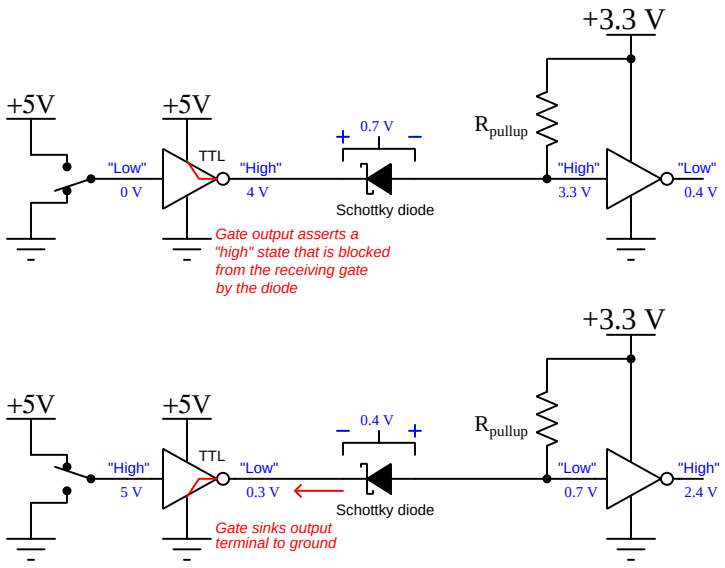
Here we see how a typical "low" voltage output by a TTL gate (with respect to ground) is definitely low enough to be reliably interpreted by the CMOS gate's input as a "low" logic state, but the "high" voltage signal output by that same TTL gate may not be high enough to satisfy

the CMOS gate's "high" threshold. Similar incompatibilities arise when the gates in question are powered by different DC power supply voltages, for example when classic TTL (requiring a 5 Volt supply) interfaces with classic CMOS (typically tolerating DC supply voltages ranging anywhere between 3 VDC and 18 VDC).

For logic families where the "low" guaranteed output voltage is reliably less than the "low" input threshold, a simple solution exists based on the use of open-collector or open-drain logic. This is where the logic gate's output terminal connects internally to a single transistor sinking to ground, but no transistor sourcing to +V. Such a logic gate is incapable of forcing a "high" output state, and relies on an external pull-up resistor to supply the "high" state. By connecting that pull-up resistor to the +V power supply terminal of the receiving gate, we guarantee that receiving gate gets just the voltage it needs for a "high" logic state without the transmitting gate having to meet the receiving gate's threshold. Using our TTL/CMOS gate pair as an example once more, we replace the TTL gate with an open-collector version:



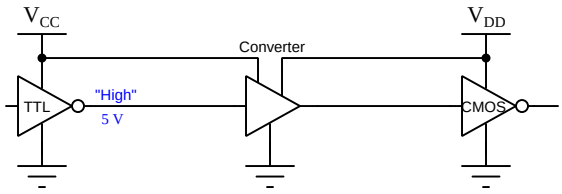
An alternative to using open-collector or open-drain gate outputs in conjunction with pull-up resistors to translate between otherwise incompatible "high" voltage levels is to add a diode to the output of a standard logic gate output so that it may only sink current and never source, and then use a pull-up resistor to provide the receiving gate its necessary "high" voltage level. The following example shows a 5-Volt TTL logic gate sending a signal to a 3.3-Volt CMOS gate using this diode-and-resistor strategy:



If not for this diode, the TTL gate's "high" output signal (which could in principle rise to a level as high as +5 Volts) could drive current into the 3.3 Volt CMOS gate's input and cause damage to that gate. No logic gate or amplifier should ever be exposed to a greater electrical potential than its positive power supply rail, in this case +3.3 Volts.

A Schottky diode is used for this application due to its low forward voltage drop (only 0.4 Volts, compared to 0.7 Volts for a standard silicon rectifying diode). A germanium diode (0.3 Volts V_F) would also be appropriate. The main disadvantage of this admirably simple approach is that the diode's added voltage drop raises the logical "low" voltage level at the receiving gate's input to a point that may be very close to or even exceed that gate's V_{IL} maximum specification. Carefully check your logic gates' output and input voltage level specifications before choosing this technique for voltage level translation!

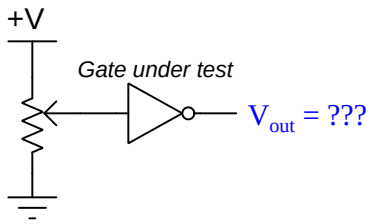
A more sophisticated solution to the problem of voltage level translation is to use a *logic level converter* gate designed for this purpose. Such gates typically have two +V power supply terminals (labeled differently) and a common ground terminal, one of those power supply terminals connecting to the +V supply of the transmitting gate and the other connecting to the +V supply of the receiving gate. The converter's input thresholds are defined by the first power supply voltage, while its output signals are defined by the second power supply voltage, thereby ensuring interoperability between the (otherwise) incompatible logic gates:



3.10 Schmitt trigger gates

What then, will a logic gate do if faced with a non-conforming input signal? That is, a signal whose voltage lies somewhere between V_{IH} and V_{IL}. The best answer is "no one knows", which is an informal way of saying the manufacturer of the logic gate will not certify its behavior with such input signals. Depending on the exact magnitude of the non-conforming voltage signal and upon internal design details of the logic gate circuit, the gate's output may go "high" or it may go "low" or it may even float to some value that also does not conform to the standard.

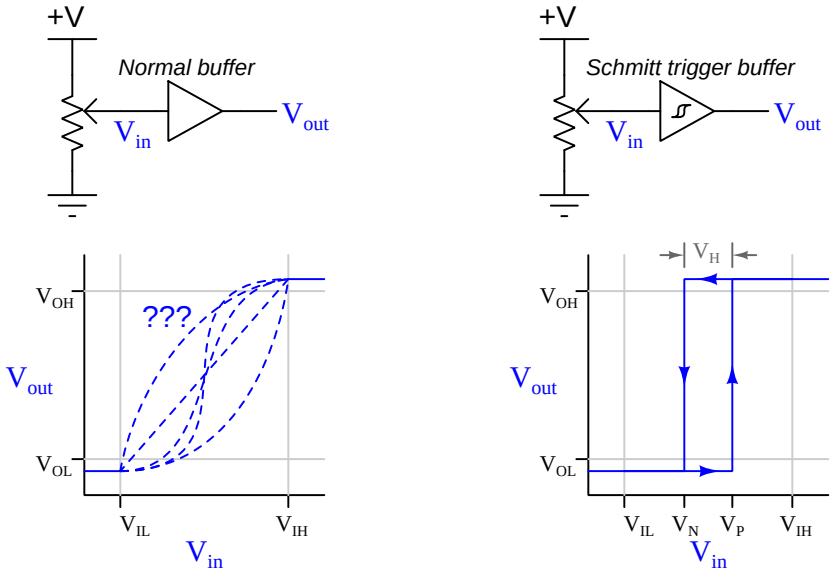
The following schematic diagram shows a test circuit for exploring the behavior of a single-input logic gate (in this case, an inverter or *NOT* gate). A DC voltmeter would be connected between the output terminal and ground to test the gate's response to the varying input voltage signal:



There are practical applications, though, where a logic gate may receive voltage signals not strictly conforming to the high/low thresholds specified in the standard for that "family" of logic design. This may occur as a result of excessive "noise" induced on the signal conductors by external sources, or it may occur as a result of a DC power supply with poor voltage regulation, or any other cause. A special type of logic gate designed to be more tolerant of non-conforming signals is the *Schmitt trigger*. The internal circuitry of a Schmitt trigger gate incorporates *positive feedback*[11] to give the gate a characteristic of *hysteresis*. This means the output "swings" to one of its two possible levels (fully "high" or fully "low") and will not switch states unless its input signal changes significantly.

[11] Feedback is an important electronic circuit design principle. Positive feedback is where the output of a circuit is "fed back" to one of its inputs in such a way that it tends to reinforce is current state. That is to say, once the circuit switches to its "high" state it works to keep itself in that state, and vice-versa.

The following illustration contrasts the behavior of a standard (left) and Schmitt trigger[12] (right) logic gate, the gate in this case being a simple *buffer* designed to output the same logic level given to its input:



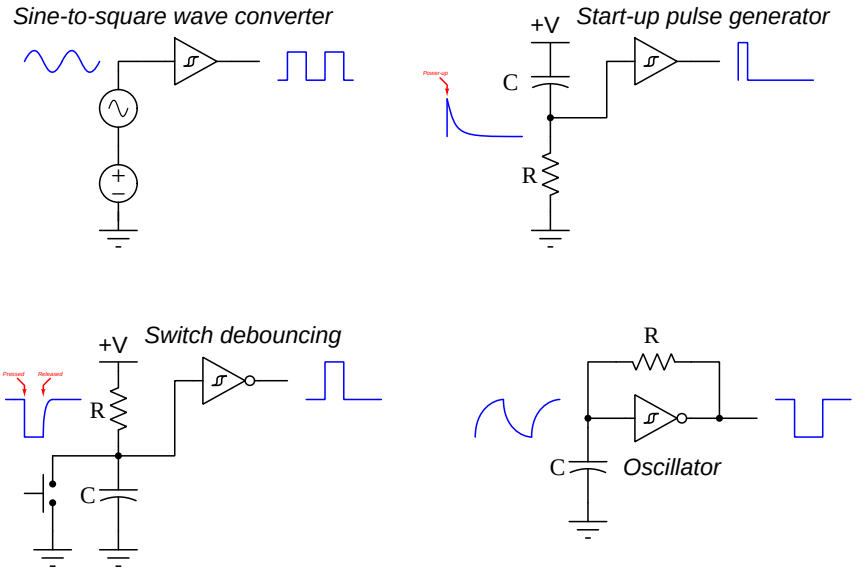
A Schmitt trigger gate does not acknowledge the input voltage signal as a "high" until it exceeds the V_P threshold *in the rising direction*, and it does not acknowledge the input signal as a "low" until it goes below V_N *in the falling direction*. The gap between the V_P and V_N thresholds is called the *hysteresis voltage* (V_H), and this provides a guaranteed degree of noise immunity for logic gates with Schmitt-trigger inputs.

One notable disadvantage of digital logic equipped with Schmitt-trigger inputs is longer propagation delay time. The additional internal circuitry necessary to incorporate positive feedback into the gate's behavior introduces extra delay time. For example, where a non-Schmitt inverter (NOT) gate might exhibit 55 nanoseconds of propagation delay, a comparable Schmitt-trigger inverter might exhibit 140 nanoseconds of delay.

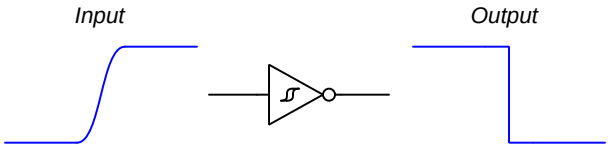
Schmitt-trigger are available in all logic families, both BJT-based and FET-based. Exact values of V_P and V_N are specified, as always, in the manufacturer's datasheet for the specific digital component.

[12] Note the small "hysteresis curve" symbol drawn inside the gate symbol, denoting that gate as having Schmitt-trigger input(s).

Schmitt triggers are so effective at interpreting indistinct voltage levels as digital states that you will often find them *intentionally* used in applications interpreting analog voltage signals. Here are a few[13], shown in schematic form:



Another common application of a Schmitt trigger logic gate is to "square up" a pulse signal with excessively long rise- or fall-time(s):



Some digital circuits, notably clock-triggered devices such as flip-flops and counter circuits, require pulse signals that rise and/or fall at some minimum speed (i.e. there is a maximum amount of time the pulse should take to transition from low to high or from high to low). If a pulse signal happens to be transitioning slower than the receiving flip-flop or counter would prefer, erratic behavior may result. Placing a Schmitt-trigger gate between the slow-rise/fall signal and the clock-triggered device's input will hasten those transitions and result in more reliable system performance.

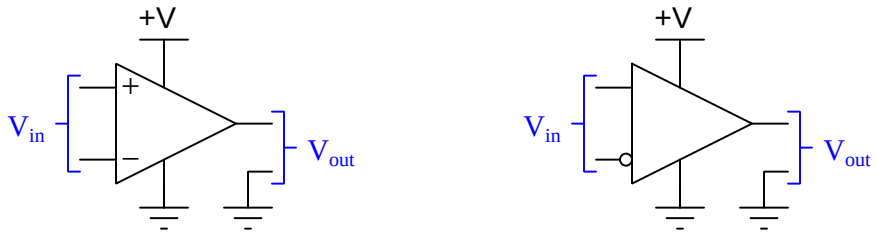
[13] Start-up pulse generators are useful in digital counting and shift-register circuits where you need to re-set the counter or register to a definite starting condition every time it is powered up. The RC network and Schmitt-trigger gate provides a single pulse at every start-up event useful for this resetting purpose. Switch debouncing circuits are also useful in digital counting and shift register circuits, where the inherently "noisy" closure of a mechanical switch may cause the digital circuit to falsely "count" multiple times for every intended closure of the switch contacts. With the RC network and the Schmitt trigger gate, the switch's "noisy" status becomes "filtered" into a clean pulse signal, one pulse per intended closure.

3.11 Comparators

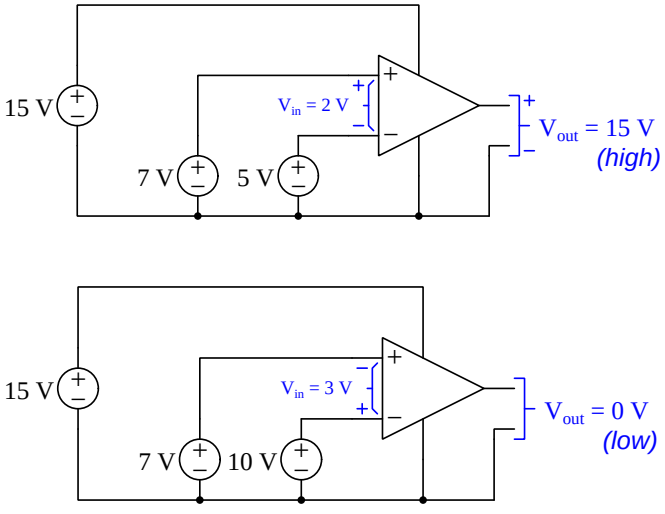
Logic gates are integrated circuits operating on signals that are discrete in nature: either "high" or "low". Analog circuits, by contrast, operate on signals representing a continuous range of voltage values (usually established by the circuit's DC power supply voltage "rails"). A *comparator* if a hybrid circuit designed to compare the voltages of two analog signals and output a discrete ("high" or "low") signal depending on which of those two analog input signals is larger. Comparators deserve mention along with logic gates because comparators are often used to "condition" analog signals for input into logic gates.

Comparators have two input terminals, marked "+" and "−" (non-inverting and inverting, respectively), one output terminal, and of course two terminals for connection to a DC power supply. Since a comparator's function is to *compare* the two ground-referenced input voltage signals, what it is really doing is measuring the difference in potential (i.e. differential voltage) between those two input terminals. The output's logical state depends strictly on the *polarity* of V_{in}:

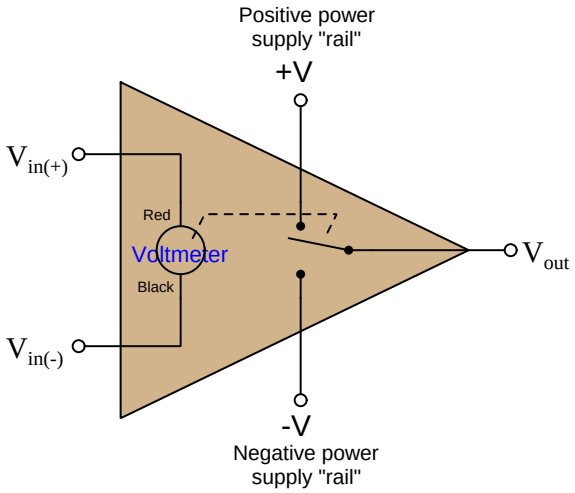
Common comparator symbols



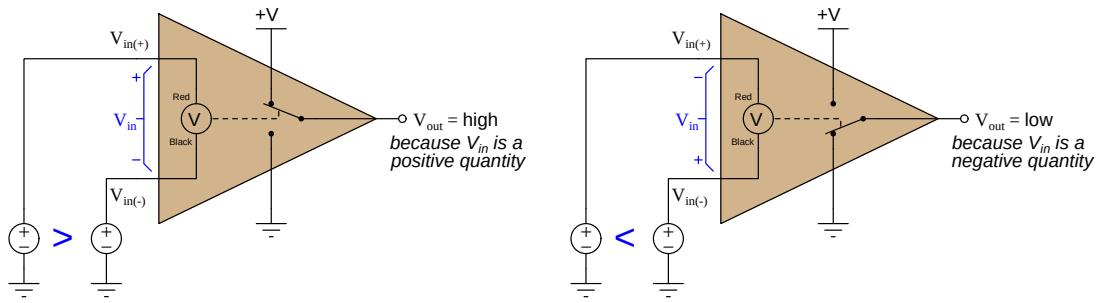
For example, consider these two comparators' output states versus the two DC voltages each are comparing. In each case the comparator is powered by a 15 Volt DC source, which defines the voltage level of its "high" state:



We may crudely model a comparator as a SPDT switch moved by the needle of an analog voltmeter connected between the two input terminals, the non-inverting (+) terminal being the meter's red test lead and the inverting terminal (−) being the black:

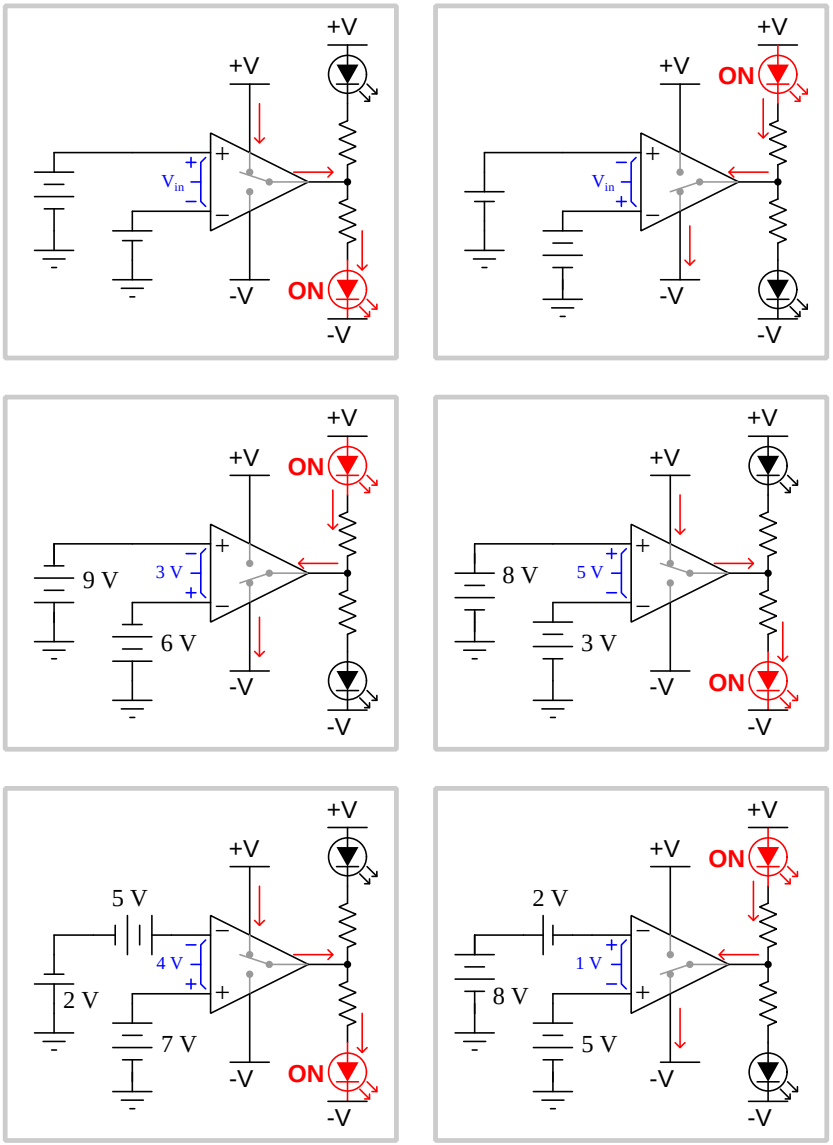


If the differential voltage between these two input terminals is of such a polarity that it would drive a voltmeter to register a positive value, the SPDT switch connects the output terminal to the positive DC rail; if the input polarity would drive a voltmeter negative, the SPDT switch connects the output terminal to the negative DC rail:

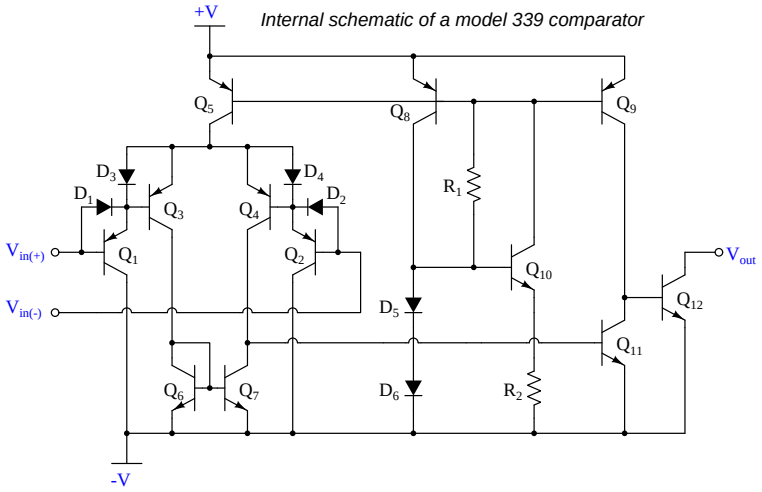


The V_{in} sensed by each comparator is the *difference* in potential between the two ground-referenced voltage sources, the output state in each case reflecting which of those two DC sources has the greater potential with respect to ground. A comparator, therefore, compares two analog signal voltages and indicates which is greater by asserting either a "high" or "low" digital logic state at its output terminal.

It is instructive to analyze examples showing a comparator's response to various input signals. For each of the following examples, look carefully at the polarity of the differential voltage between the two input terminals and see how that polarity relates to the output's "high" or "low" state:



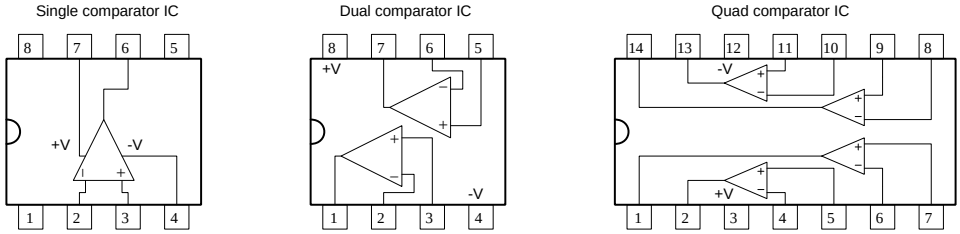
Of course, this voltmeter and SPDT switch model is merely an analogy. The internal circuitry of a comparator is, as one might expect, fairly complex. A good representative example of a comparator is the popular model 339 shown in the following diagram:



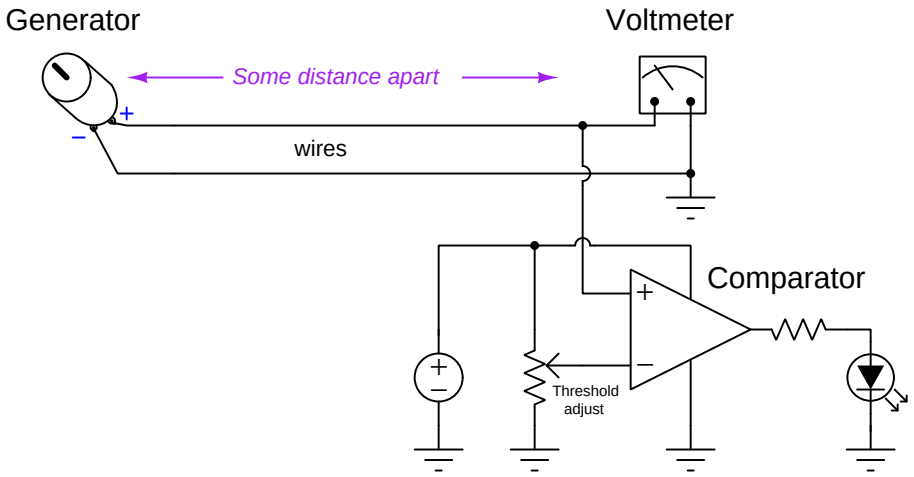
A noteworthy feature of the model 339 comparator is that it is capable only of sinking current at its output current, not sourcing current. This means a pullup resistor must be connected to its output terminal to provide a default "high" logic state when the 339's output transistor is turned off, just like an open-collector or open-drain logic gate. The 339's open-collector output lends itself well to interfacing with a broad range of logic families because the "high" voltage level may be user-defined by which +V power supply rail the pullup resistor connects to.

Comparators capable of both sourcing and sinking output current also exist, the model TLC3704 being one of these

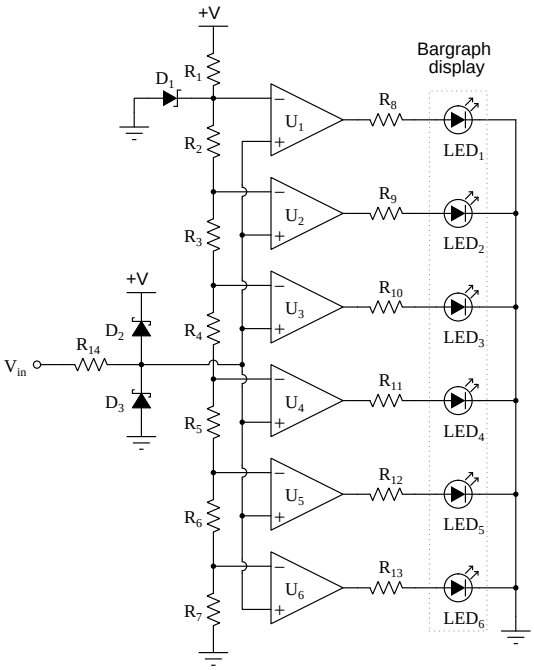
As integrated circuits, comparators are often packaged as single units, dual units, or even quad units per IC "chip". Common "pinout" assignments for DIP-packaged comparators are shown here:



In the following example circuit we see a comparator comparing the voltage output by a DC generator against the voltage dropped by a manually-adjusted potentiometer, such that the LED will energize if ever the generator's voltage exceeds the threshold limit set by the potentiometer:



Another example circuit shows how a set of comparators may be used to create a *bargraph* display for an analog voltage signal:



As with logic gates, comparators are often drawn in schematic diagrams without their power supply terminals shown, for simplicity. Comparators, of course, require DC power to function just

like any other active electronic component, so it should be clearly understood that the omission of power supply terminals in schematic diagrams is not literal.

Chapter 4

Historical References

This chapter is where you will find references to historical texts and technologies related to the module's topic.

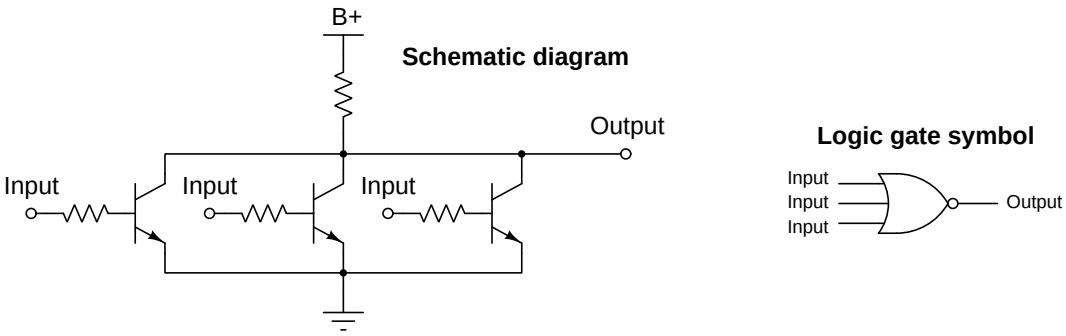
Readers may wonder why historical references might be included in any modern lesson on a subject. Why dwell on old ideas and obsolete technologies? One answer to this question is that the initial discoveries and early applications of scientific principles typically present those principles in forms that are unusually easy to grasp. Anyone who first discovers a new principle must necessarily do so from a perspective of ignorance (i.e. if you truly *discover* something yourself, it means you must have come to that discovery with no prior knowledge of it and no hints from others knowledgeable in it), and in so doing the discoverer lacks any hindsight or advantage that might have otherwise come from a more advanced perspective. Thus, discoverers are forced to think and express themselves in less-advanced terms, and this often makes their explanations more readily accessible to others who, like the discoverer, comes to this idea with no prior knowledge. Furthermore, early discoverers often faced the daunting challenge of explaining their new and complex ideas to a naturally skeptical scientific community, and this pressure incentivized clear and compelling communication. As James Clerk Maxwell eloquently stated in the Preface to his book *A Treatise on Electricity and Magnetism* written in 1873,

> It is of great advantage to the student of any subject to read the original memoirs on that subject, for science is always most completely assimilated when it is in its nascent state . . . `[page xi]`

Furthermore, grasping the historical context of technological discoveries is important for understanding how science intersects with culture and civilization, which is ever important because new discoveries and new applications of existing discoveries will always continue to impact our lives. One will often find themselves impressed by the ingenuity of previous generations, and by the high degree of refinement to which now-obsolete technologies were once raised. There is much to learn and much inspiration to be drawn from the technological past, and to the inquisitive mind these historical references are treasures waiting to be (re)-discovered.

4.1 NASA's Apollo Guidance Computer

The digital computer used for guidance functions in the 1960's era Apollo spacecraft (the one used to transport the first humans to Earth's Moon) built by NASA used bipolar transistor logic, consisting almost entirely of NOR logic gates. The schematic diagram for each three-input NOR gate was as follows, along with its corresponding logic gate symbol:



Note how the positive power supply terminal is labeled *B+*, an anachronistic reference to the positive terminal of a high-voltage *battery* (hence the letter "B") used to power vacuum tube circuits. Based on the knowledge that bipolar transistors are normally "off" devices, and require the base-emitter junction to be forward-biased in order to turn "on", we can tell if any input goes to a high state (i.e. connected to the positive rail of the DC power source), that respective NPN transistor will turn on and bring the output terminal's potential down (nearly) to ground. In other words, *any high input forces the output to be low*: the very definition of a NOR function.

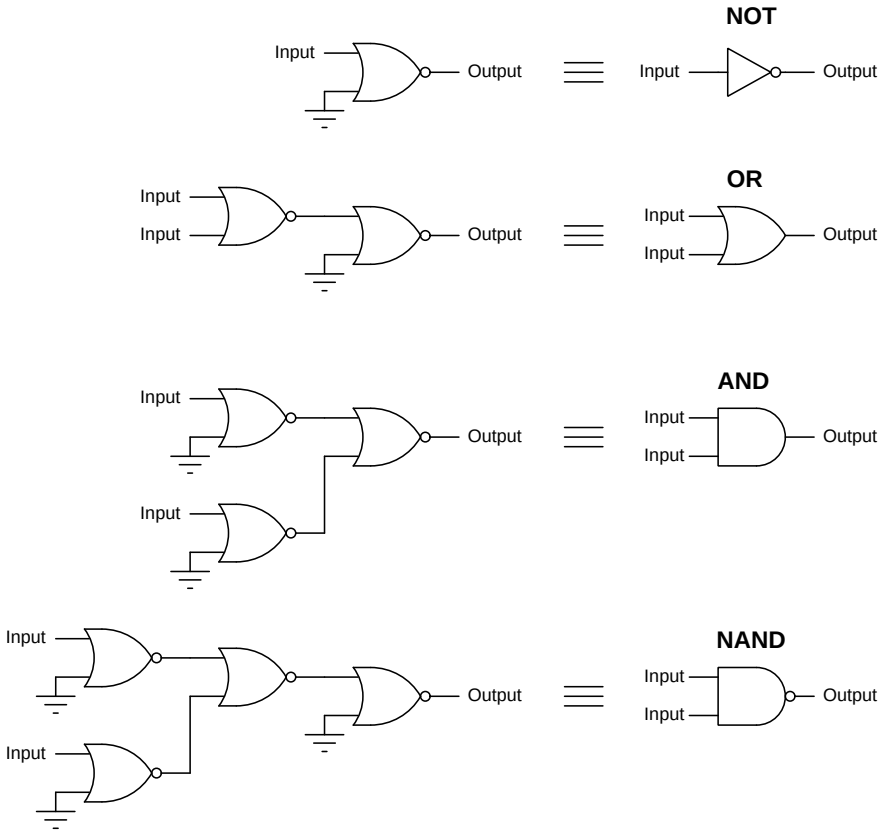
The three transistors can only *sink* current, and therefore this NOR gate's sourcing capability is limited by the resistor between the three collectors and the B+ power supply terminal. In other words, this NOR gate had a significantly greater current-sinking rating than its current-sourcing rating.

A rather short technical document entitled "A Case History Of The AGC Integrated Logic Circuits" describes how nearly the entire computer consisted of these three-input NOR logic gates:

> The standardization approach, which is particularly adaptable to digital computers, has been demonstrated with the Polaris flight computer and extended with integrated circuits to the Apollo Guidance Computer. Both computers were designed to use a three input NOR Gate as the only logic element. All logic functions are generated by interconnecting the three input NOR Gate with no additional logic blocks, resistors, or capacitors. At first glance, it appears that using only one type of logic block greatly increases the number of blocks required for the computer. But, by judiciously selecting and organizing the logic functions it is quickly apparent that few additional blocks are necessary. The few additional units required are greatly counterbalanced by the increased reliability gained during both the manufacturing of components and fabrication of the components into modules. [page 3]

It is possible to construct any digital logic function using nothing but NOR gates, because the NOR gate is one of two universal logic gate types (NAND being the other). The key to gate universality is the ability to function as an inverter (i.e. the NOT function) because inverting the input(s) and/or output of any logic function makes possible the transformation of that logic function into all other types. Any NOR gate will function as an inverter if the unused input(s) are fixed to "high" (1) logic states, the one remaining input controlling the gate's output. With a NOR gate such as the type used by NASA to build the Apollo Guidance Computers, the unused inputs may simply be left floating.

The following illustration demonstrates[1] how it is possible to use nothing but NOR gates to construct the other four basic logic functions (NOT, OR, AND, and NAND):

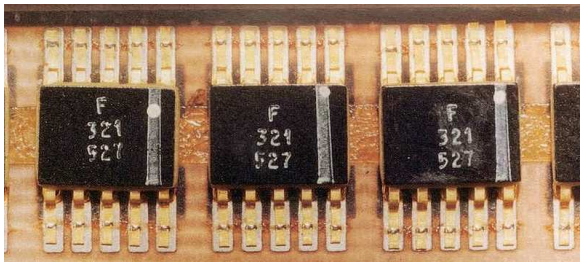


[1] This illustration itself, of course, does not actually *demonstrate* the universality of NOR gates. In order for a true demonstration to be complete, one must observe the system operating as intended. For this, it is left as an exercise to the reader to perform "thought experiments" on these four logic circuits, imagining the input terminals in their various possible states and following through to the consequent output states based on the truth table of a NOR function.

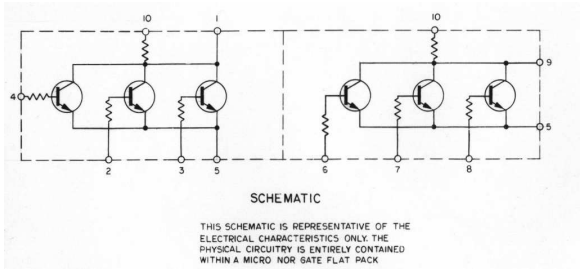
This next passage from the NASA document explains why NOR gates were chosen rather than NAND, and gives some technical specifications for the integrated circuits:

> The logic element utilized in the Apollo Guidance Computer is the three input NOR Gate as shown in Fig. 1. At the time that the decision was made to use integrated circuits, the NOR Gate, as shown, was the only device available in large quantities. The simplicity of the circuit allowed several manufacturers to produce interchangeable devices so that reasonable competition was assured. Because of recent process development in integrated circuits, the NOR Gate has been able to remain competitive on the basis of speed, power and noise immunity. This circuit is used at 3 V and 15 mW, but is rated at 8 V and 100 mW. Unpowered temperature rating is 150 ^{o}C. [page 4]

These NOR gate integrated circuits were enclosed in "flatpack" packages and soldered into printed circuit board assemblies. In the following photograph[2] we see three of these "flatpack" NOR gate ICs soldered into place:



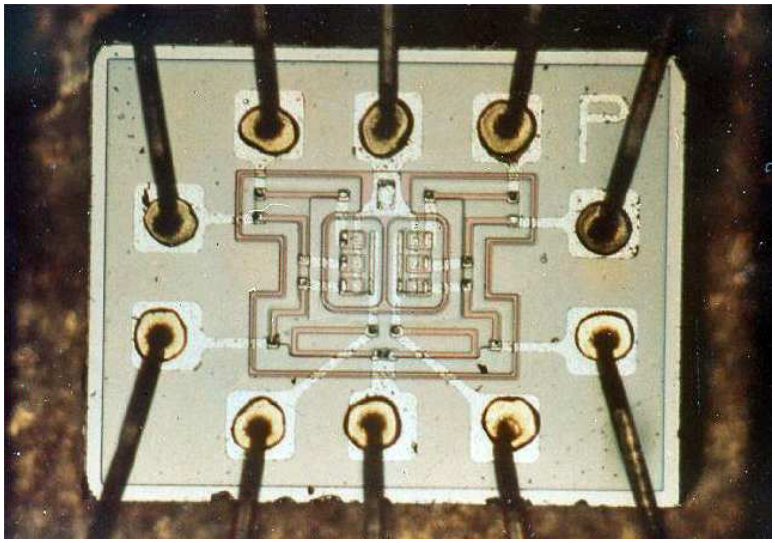
Each of these ten-terminal "flatpacks" contained two NOR gates, as shown in this NASA schematic[3]:



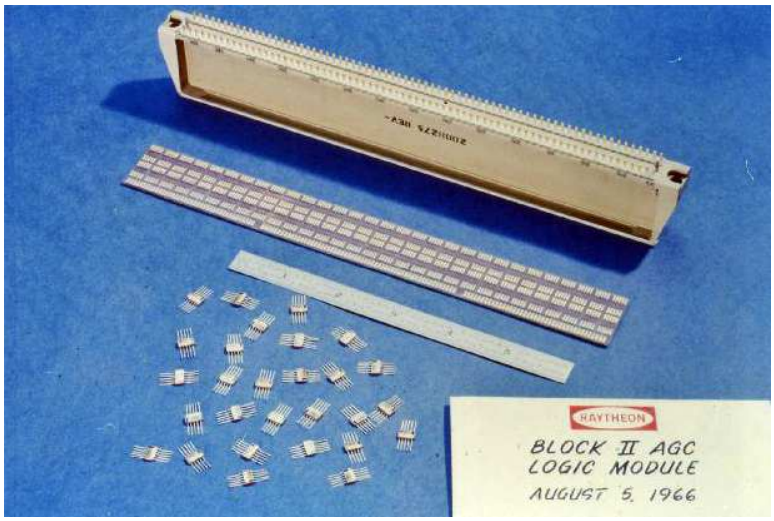
Note how, just as with modern multi-gate ICs, some of the terminals on this early example of an integrated circuit are shared in common with each of the internal gates. In this case we see terminal 10 (B+) common to both NOR gates, as well as terminal 5 (ground).

[2] This image was cropped from a public-domain photograph made courtesy of Grabert who posted it to Wikipedia.
[3] Another public-domain photograph courtesy of Grabert.

Here we see a close-up view[4] of the silicon wafer inside one of these dual-NOR gate IC packages:

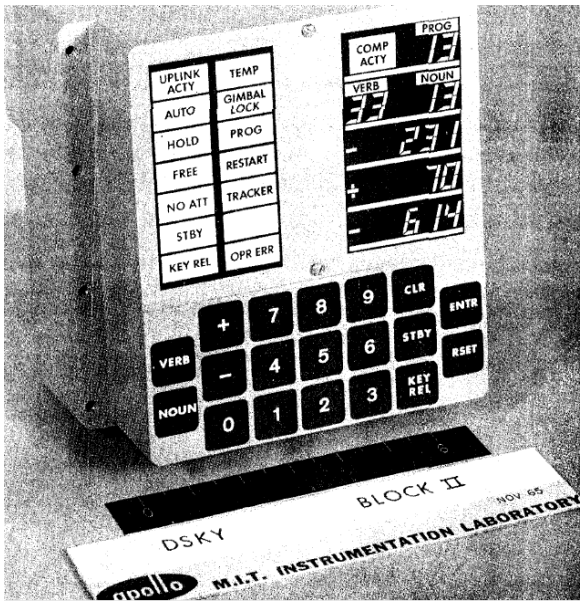


In another photograph[5] of the Apollo guidance computer assemblies, we see several of these NOR gate ICs lying on a table next to one of the module assemblies of the computer called "Block 2 AGC Logic":

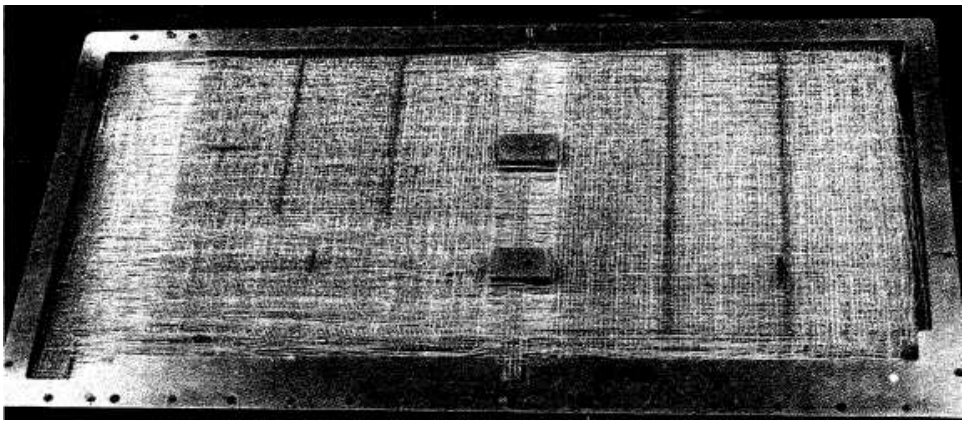


[4] Another photograph courtesy of Grabert, graciously released into the public domain.
[5] Another photograph courtesy of Grabert, graciously released into the public domain.

A photograph showing the operator console for this digital computer appears in the following photograph, from page 12 of the NASA *Case History* document:



Interconnections between NOR gates were quite extensive in this computer, the connections made by *wire-wrapping* on the side of the printed circuit board opposite of the ICs themselves. In other words, the ICs were soldered to one side of the circuit board but wire-wrap pins protruding through the other served as connection points for the wrapped wires connecting gates to each other. The following photograph, taken from page 11 of the NASA document shows wiring on the underside of the computer chassis. The quality of this photograph is too poor to see anything but a dense field of wire-wrap terminal pins and courses of thin wires interconnecting those pins:



Chapter 5

Derivations and Technical References

This chapter is where you will find mathematical derivations too detailed to include in the tutorial, and/or tables and other technical reference material.

5.1 TTL logic levels

Logic gates need to reliably communicate with each other, the voltage levels output by the "driving" gate being compatible with the input voltage levels of the "receiving" gate. These specifications are given as follows:

- V_{OH} = minimum voltage output by a gate in the "high" (1) state
- V_{OL} = maximum voltage output by a gate in the "low" (0) state
- V_{IH} = minimum voltage received by a gate to be interpreted as a "high" (1) state
- V_{IL} = maximum voltage received by a gate to be interpreted as a "low" (0) state

Classic "LS" family of TTL gate circuits using bipolar transistors operated on a 5 Volt DC power supply, outputting at least 2.4 Volts in the "high" state and 0.4 Volts in the "low" state. The same logic family would accept any input voltage signal greater than 2.0 Volts as a "high" and less than 0.8 Volts as a "low". The amount of *noise margin* is the difference between the guaranteed output voltage levels and the acceptable input voltage levels. Represented in table form for 5-Volt TTL gates:

Logic state	**Guaranteed output**	**Acceptable input**	**Noise margin**
High	$V_{OH} = 2.4$ Volts min.	$V_{IH} = 2.0$ Volts min.	$2.4 - 2.0 = 0.4$ Volts
Low	$V_{OL} = 0.4$ Volts max.	$V_{IL} = 0.8$ Volts max.	$0.8 - 0.4 = 0.4$ Volts

If an ordinary digital logic gate receives an input voltage signal lying somewhere between the threshold values of V_{IH} and V_{IL}, its output state will be unpredictable. The gate may "assume" either a high or a low state, or worse yet may try to process that signal in an analog fashion, generating an output voltage level below V_{OH} and above V_{OL}, thus propagating the problem to the next logic gate(s). For this reason it is very important to design logic circuits to avoid voltage levels below V_{IH} and above V_{IL}.

Classic TTL logic IC part numbers begin with either 54 (military-grade) or 74 (commercial grade), but not all 54- or 74- series ICs are of traditional TTL (bipolar transistor) design. For example, the "HC" series of 54- and 74-numbered ICs utilize MOSFETs rather than bipolar transistors, but are otherwise designed to be pin-for-pin compatible with classic TTL logic ICs. These "high-speed CMOS" ICs are designed to function nearly identically to their bipolar-transistor counterparts, but with much lower current requirements and a slightly wider power supply range (2 to 6 Volts typical). Like classic 4000-series CMOS logic ICs, this means the minimum and maximum acceptable voltage levels for 54HC- or 74HC- series ICs "high" and "low" signals vary with supply voltage. By contrast, since classic TTL digital logic ICs are constrained to a very narrow range of DC power supply voltage (typically 4.75 to 5.25 Volts) their acceptable voltage levels for "high" and "low" signals is correspondingly fixed.

5.2 CMOS logic levels

Logic gates need to reliably communicate with each other, the voltage levels output by the "driving" gate being compatible with the input voltage levels of the "receiving" gate. These specifications are given as follows:

- V_{OH} = minimum voltage output by a gate in the "high" (1) state
- V_{OL} = maximum voltage output by a gate in the "low" (0) state
- V_{IH} = minimum voltage received by a gate to be interpreted as a "high" (1) state
- V_{IL} = maximum voltage received by a gate to be interpreted as a "low" (0) state

Classic "CD" family of CMOS gate circuits using complementary MOSFET transistors operating on a 5 Volt DC power supply has much wider greater noise margins than "LS" series TTL, outputting at least 4.44 Volts in the "high" state and 0.5 Volts in the "low" state, while accepting any input voltage signal greater than 3.5 Volts as a "high" and less than 1.5 Volts as a "low". Represented in table form for 5-Volt CMOS gates:

Logic state	**Guaranteed output**	**Acceptable input**	**Noise margin**
High	$V_{OH} = 4.44$ Volts min.	$V_{IH} = 3.5$ Volts min.	$4.44 - 3.5 = 0.94$ Volts
Low	$V_{OL} = 0.5$ Volts max.	$V_{IL} = 1.5$ Volts max.	$1.5 - 0.5 = 1.0$ Volt

CD-family CMOS logic was not limited to a 5 Volt DC power supply as was the LS (TTL) bipolar family of logic gates. Typical DC power supply limits ranged from 3 Volts to 18 Volts, with acceptable input voltage levels varying as a function of power supply voltage. Output voltage levels for a CD-family logic gate also followed power supply voltage, typically within 0.5 Volts of the power supply rails.

If an ordinary digital logic gate receives an input voltage signal lying somewhere between the threshold values of V_{IH} and V_{IL}, its output state will be unpredictable. The gate may "assume" either a high or a low state, or worse yet may try to process that signal in an analog fashion, generating an output voltage level below V_{OH} and above V_{OL}, thus propagating the problem to the next logic gate(s). For this reason it is very important to design logic circuits to avoid voltage levels below V_{IH} and above V_{IL}.

5.3 Logic families

Many possible circuit designs exist to create digital logic gates and associated logic circuitry. For example, one could design and build a NAND gate using nothing but bipolar (NPN and/or PNP) transistors; alternatively, one could make a NAND gate using nothing but MOSFETs. And, for each of these transistor types there are many variations of circuit design possible, such that any logic gate made according to a particular circuit design standard would have unique characteristics, some of which are listed here:

- DC power supply voltage range
- Acceptable "high" and "low" signal voltage levels at gate input terminals
- Guaranteed "high" and "low" signal voltage levels at gate output terminals
- Typical propagation delay times
- Typical output current limitations

When selecting logic ICs to form larger, more complex digital logic systems, it is important for the circuit designer to know that those logic components will function well with each other: that their DC power supply voltage ranges are compatible, that their output signal voltage levels will comply with their input signal voltage requirements, etc. In order to facilitate compatible device selection, logic IC manufacturers label their products with part numbers and codes designating each device's membership within different *families* of logic circuits. Each of these "families" is guaranteed to be interoperable within itself, meaning that any logic component from one family will be fully compatible with any other logic component belonging to the same family. Some families are interoperable between each other, too, but compatibility amongst members of a single IC logic family is the basic purpose of having these "family" classifications.

The early years of digital logic circuit manufacturing saw emergence of families such as Resistor-Transistor Logic (RTL), Diode-Transistor Logic (DTL), and Emitter-Coupled Logic (ECL), the first two now considered obsolete. Later emerged the Transistor-Transistor Logic (TTL) family based on NPN and PNP bipolar transistor circuitry, this family identified by part numbers beginning with either 54 or 74, the 54-series ICs having military-grade specifications (e.g. operating temperature limits) and the 74-series ICs having commercial-grade specifications. After that came the Complementary Metal-Oxide Semiconductor (CMOS) family based on N-channel and P-channel MOSFETs rather than bipolar transistors, this family identified by part numbers beginning with 4 or 14[1].

IC logic families soon developed into sub-families having different characteristics such as power consumption and switching speed, some of these sub-families designated by letters in the middle of the part number. For example, the classic 7411 is a triple 3-input AND gate IC based on TTL (bipolar transistor) technology, but in the years to follow the classic 54/74 TTL family's introduction there developed the "LS" sub-family (e.g. 74LS11 triple 3-input AND gate IC) using Schottky diodes within the internal circuitry to reduce power consumption, and then later the "HC" sub-family (e.g.

[1] The prepending of a "1" to the 4000-series part number was typical of ICs manufactured by Motorola, just to make things confusing!

74HC11 triple 3-input AND gate IC) using MOSFETs rather than bipolar transistors but otherwise designed to be backward-compatible with legacy 74 and 74LS sub-families.

More recent developments in IC logic have resulted in logic circuit designs optimized for lower and lower DC supply voltage ranges, this design trend addressing the need for more advanced consumer electronic products powered by chemical batteries. For any given amount of current, a lower operating voltage means less power dissipation, and this in turn means a battery of any given size will be able to energize that logic circuit for a longer period of time. Portable computers, mobile telephones, and other personal electronic devices naturally benefit from this technological trend.

An exhaustive list of IC logic families would be beyond the scope of this reference, and frankly is better left to the manufacturers themselves. However, here I will provide a listing of some of the more common digital IC logic families at the time of this writing (2023):

- **5400/7400 classic TTL family** – uses bipolar transistor technology; operates on 5 Volt DC power supply with a tight margin, typically 4.75 Volts minimum and 5.25 Volts maximum for the 7400 commercial-grade series, 4.5 Volts minimum and 5.5 Volts maximum for the 5400 military-grade series

- **4000 classic CMOS family** – uses complementary (N- and P-channel together) MOSFET technology; operates on a wide range of DC power supply voltages, typically 3 Volts minimum to 18 Volts maximum, often standardized at 5 Volts, 10 Volts, or 15 Volts; notably slower in switching speed than classic TTL but operates at a *far* lower power dissipation[2]

- **5400/7400 ALS, AS, S, and LS sub-families** – uses Schottky diodes within the internal TTL circuitry to help avoid transistor saturation and thereby increase maximum switching speeds; same DC power supply range as classic 5400/7400 TTL

- **5400/7400 F sub-family** – this is a "fast" variant of TTL logic designed for low propagation delay times and high-speed operation; limited to the same DC power supply voltage range as classic 5400/7400 TTL devices but with significantly higher current requirements and consequently higher power dissipation

- **5400/7400 HC sub-family** – uses MOSFETs rather than bipolar transistors internally, but designed to mimic the operation of classic TTL devices at much-reduced power dissipation; enjoys a wider DC power supply range than classic TTL, typically 2 Volts minimum and 6 Volts maximum

- **5400/7400 HCT sub-family** – similar to the HC sub-family in its use of MOSFETs rather than bipolar transistors, but designed to be fully interoperable with classic TTL devices; limited to the same 5400-series classic TTL power supply range of 4.5 Volts minimum and 5.5 Volts maximum

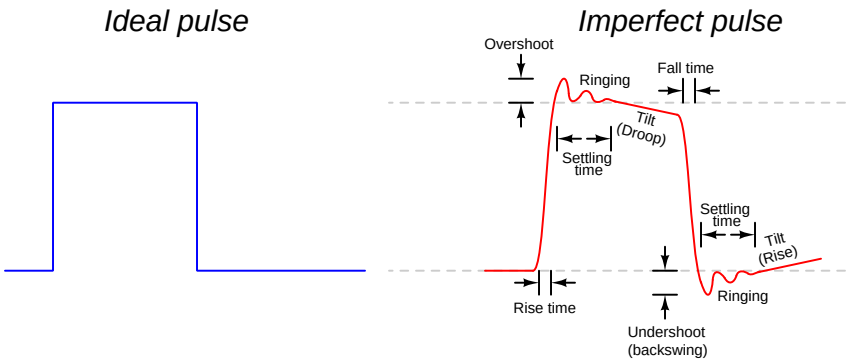
- **5400/7400 BCT sub-family** – uses a combination of bipolar transistors and MOSFETs internally (BiCMOS); limited to the same 5400-series classic TTL power supply range of 4.5 Volts minimum and 5.5 Volts maximum

[2] This trade-off between device speed versus device power dissipation is a common one in digital electronics. Often we need to sacrifice one to achieve superior performance in the other.

- **5400/7400 LVC sub-family** – this "low-voltage CMOS" sub-family operates with a considerably lower DC power supply range than previous 5400/7400 digital logic sub-families, typically 2 Volts minimum and 3.6 Volts maximum, often standardized at 3.3 Volts
- **5400/7400 LVT sub-family** – this "low-voltage BiCMOS" sub-family uses a combination of bipolar transistors and MOSFETs internally and operates on a DC power supply voltage range of 2.7 Volts minimum and 3.6 Volts maximum, often standardized at 3.3 Volts
- **5400/7400 AVC sub-family** – this "advanced low-voltage CMOS" sub-family extends the operating DC power supply voltage to even lower levels with 1.4 Volts being minimum, often standardized at 3.3 Volts, 2.5 Volts, or 1.8 Volts
- **5400/7400 AUC sub-family** – this "advanced ultra-low voltage CMOS" sub family pushes the DC power supply voltage envelope down even further, with 0.8 Volts minimum and 3.6 Volts maximum, often standardized at 2.5 Volts, 1.8 Volts, and 1.2 Volts

5.4 Digital pulse criteria

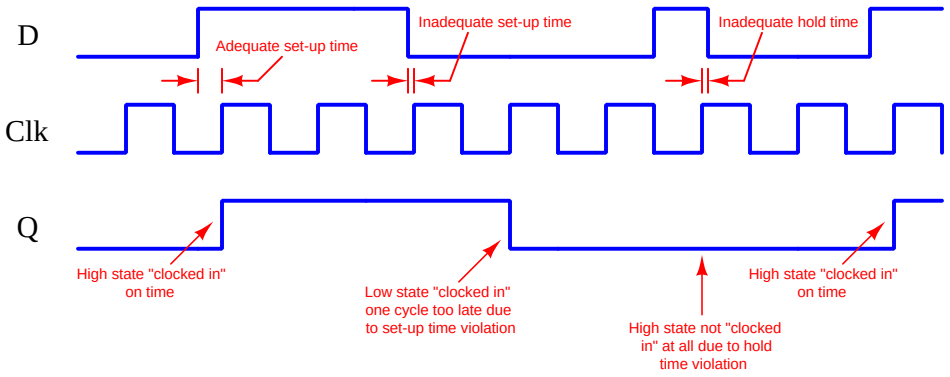
Ideal "pulse" signals have infinitely-steep rise and fall times, level "high" and "low" states well within the specified voltage ranges, and no other imperfections or artifacts. Real pulses always deviate from ideal, often in multiple ways:



Rise and fall times are strongly influenced by parasitic capacitance existing on the signal line with reference to ground, as well as the current-sourcing and current-sinking capability of the logic gate or other device generating the pulse signal. The capacitive "Ohm's Law" formula $I = C\frac{dV}{dt}$ predicts how much current will be necessary to create a linear rate-of-rise or rate-of-fall of voltage for a given capacitance. Note that to achieve infinitely steep rise or fall times an *infinite* amount of current would be necessary to charge/discharge whatever capacitance happens to exist on that signal line!

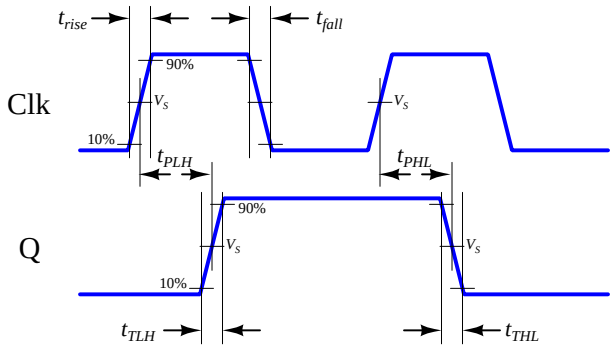
Ringing is caused by *resonance* occurring between parasitic capacitance and parasitic inductance, those two phenomena naturally exchanging energy back and forth with each other to produce the oscillatory waves we call "ringing". Overshoot and undershoot are also the result stored energy within these parasitic L and C circuit properties, and since parasitic capacitance and parasitic inductance can never be fully eliminated from any real circuit it means the effects of over/undershoot and ringing is likewise unavoidable. If you don't see these effects in your pulses, you just aren't viewing them at a fast enough time scale!

Clock-synchronized digital logic circuits such as counters, shift registers, and microprocessors require their input signals to be at stable states immediately before and immediately after the clock pulse arrives. For example, the following timing diagram shows input and output states for a D-type flip-flop circuit (positive-edge triggered), showing the effects of some signal timing violations:



Datasheets for digital circuits often provide timing diagrams showing criteria related to pulse signal timing and logic states. These diagrams don't typically show ideal square-edged pulses, but rather *trapezoidal* pulse profiles intended to exaggerate realistic features such as rise and fall times, propagation delays, and minimum set-up/hold times. Such diagrams usually confuse students who are accustomed to seeing square-edged pulses in their textbook timing diagrams. This technical reference will show some typical timing diagrams and explain what they represent.

For example, consider this timing diagram for a positive-edge-triggered JK flip-flop having both its J and K inputs tied high so as to maintain the circuit in its "toggle" mode. As such we would expect its output (Q) to change state with every rising edge of the clock pulse:



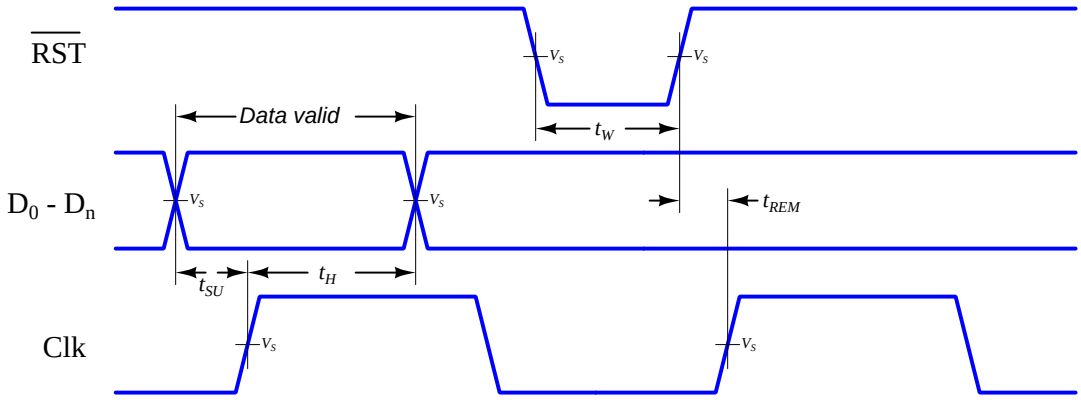
Each of the labels found in this diagram is defined as follows:

- t_{rise} = Rise time of input signal, typically measured from 10% of signal amplitude to 90% of signal amplitude
- t_{fall} = Fall time of input signal, typically measured from 90% of signal amplitude to 10% of signal amplitude
- t_{TLH} = Low-to-High transition time of output signal, typically measured from 10% of signal amplitude to 90% of signal amplitude (the same concept as rise time, but applied to the output signal instead of the input signal)
- t_{THL} = High-to-Low transition time of output signal, typically measured from 90% of signal amplitude to 10% of signal amplitude (the same concept as fall time, but applied to the output signal instead of the input signal)
- t_{PLH} = Propagation delay time of output signal when switching from low to high
- t_{PHL} = Propagation delay time of output signal when switching from high to low
- V_S = Switching threshold voltage, typically defined as 50% of signal amplitude

This timing diagram shows how a digital logic circuit reacts to a single input signal, in this case the clock pulse. Although this example happens to be for a JK flip-flop in toggle mode, the same type of timing diagram with its exaggerated rise/fall times and propagation delays could be applied to any digital logic gate whose output state depended solely on the state of a single input.

For synchronous digital logic circuits where input signals must coordinate with the clock pulse signal in order to be properly accepted by the circuit, we typically find timing diagrams comparing these input states to each other, often without showing the output(s) at all. Instead of showing us how the digital logic circuit will react to an input signal, this sort of timing diagram shows what the digital logic circuit *expects* of its multiple input signals.

The example is shown here for a positive-edge-triggered D register[3] having multiple data lines (D_0 through D_n), one asynchronous[4] reset line ($\overline{RST}$), and one clock input. The arbitrary logic levels of the multiple data lines are shown as a pair of complementary-state pulse waveforms, the only relevant features being the *timing* of the data and not the particular voltage levels of the data signals:



Labels shown in this diagram refer to *minimum* time durations the logic circuit requires for reliable operation:

- t_{SU} = Minimum set-up time before the arrival of the next clock pulse
- t_H = Minimum hold time following the last clock pulse
- t_W = Minimum width (duration) of the asynchronous reset pulse
- t_{REM} = Minimum removal time before the arrival of the next clock pulse

Violations of any of these minimum times may result in unexpected behavior from the logic circuit, and is an all-too-common cause of spurious errors in high-speed digital circuit designs. The assessment of digital pulse signals with regard to reliable circuit operation is generally known as *digital signal integrity*.

[3] In this case, a "D register" is synonymous with multiple D-type flip-flops sharing a common clock input, passing data through from each D input to each corresponding Q output synchronously with each clock pulse.

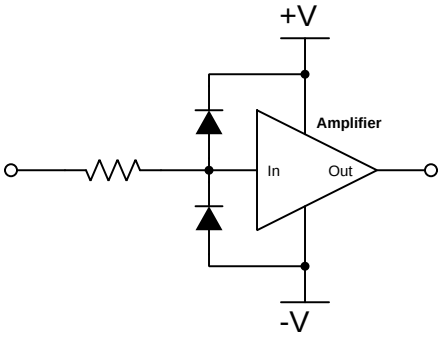
[4] To review, a *synchronous* input depends on a clock pulse while an *asynchronous* input is able to affect the circuit independent of the clock pulse.

5.5 Protecting logic gate inputs from over-voltage

Electronic amplifier circuits are extremely useful devices, comprising portions of many practical integrated-circuit (IC) electronic components such as digital logic gates, comparators, operational amplifiers, signal mixers, etc. Solid-state amplifier circuits use *transistors* to allow one electrical signal to control another, and these transistors tend to be susceptible to damage from excessive applied signal voltage. In the case of MOSFET transistors, an excess of signal voltage may puncture the extremely thin layer of metal-oxide insulation separating the transistor's gate terminal from its current-carrying channel. In the case of bipolar (NPN, PNP) transistors, an excess of signal voltage may break down reverse-biased PN junctions inside the transistor, often causing them to fail in "shorted" states. Excessive applied signal voltage may also cause damage to bipolar transistors if forward-biasing PN junctions to the extent that they conduct high amounts of current which may lead to thermal damage.

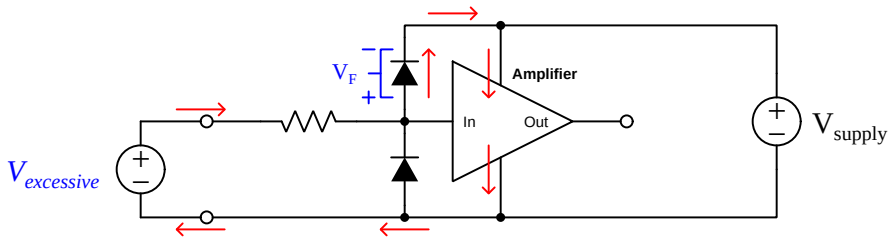
Sources of excessive signal voltage may be broadly categorized as *electro-static discharge* (ESD) or *electrical over-stress* (EOS). ESD happens when some object external to the circuit (including human bodies) accumulates an electro-static charge in its own capacitance, and this capacitively-stored charge is suddenly send to the circuit where it drops a voltage across circuit components high enough to cause damage. EOS is a more broad description of any over-voltage or over-current condition caused by one circuit sourcing energy to another, such as when an electrical test instrument is connected to a signal source that is too great for that instrument to handle.

A simple and effective way to limit both applied voltage and applied current to the input of any electronic amplifier is to connect a current-limiting resistor in series between the input terminal and the amplifier, as well as connect diodes between the input terminal and either DC power supply "rail" terminal:



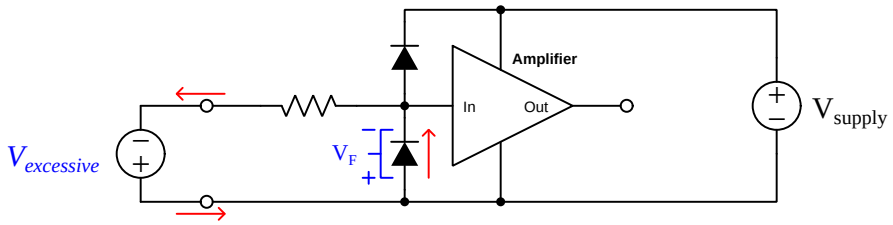
It matters not how the amplifier itself is internally constructed, nor what its larger purpose is – whether it be part of a digital logic gate, a comparator, an operational amplifier, or something even more complex. The purpose of these two diodes and the series resistor is to prevent any component within the amplifier from experiencing either too much voltage and/or too much current.

To understand how this protection network functions, consider the following circumstance where an excessive positive potential is applied to the input terminal by some external source. Here, "excessive" is defined as any voltage greater in magnitude than the DC power source energizing the amplifier:



The upper diode of the protection network forward-biases to permit current from the offending source to pass into the positive power bus of the amplifier. So long as the limiting resistor restricts this resulting current to a value less than what the amplifier requires for its normal operation, the voltage between the amplifier's input terminal and the negative rail cannot exceed the DC power supply's voltage plus the diode's forward-voltage drop ($V_{supply} + V_F$). With a normal silicon diode this means a voltage no greater than 0.7 Volts plus the positive DC rail voltage. If Schottky diodes are used in the protection network, it means a voltage no greater than 0.4 Volts beyond the positive rail. Such mild over-voltage conditions are unlikely to cause damage to any of the amplifier's internal transistors.

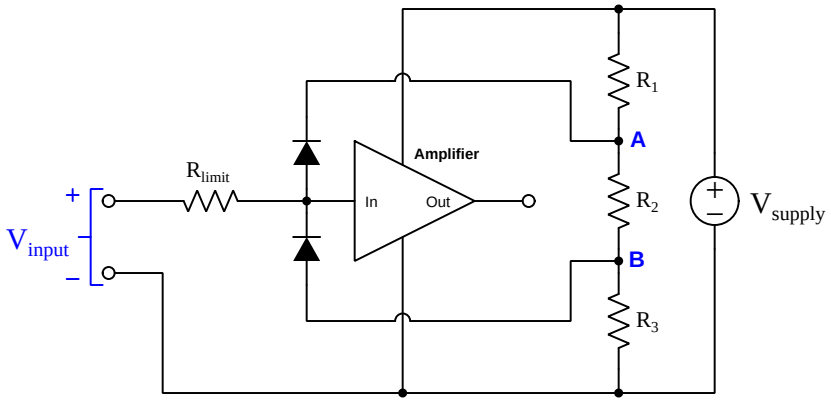
Similarly, if an excessive negative voltage is applied by an external source ("excessive" being any significant potential below that of the negative DC power supply rail):



Here the lower protection diode forward-biases and permits current from the external source to pass through the limiting resistor. While this occurs, the amplifier will never see more than that protection diode's forward voltage drop (V_F) between its input terminal and the negative DC rail bus, which again is unlikely to damage any transistor inside the amplifier.

Such protection networks are often found inside of integrated circuits, sometimes without a series protection resistor in their most basic forms. Protection diodes are standard for CMOS digital logic gates to help protect their constituent MOSFET transistor gates from damage from ESD.

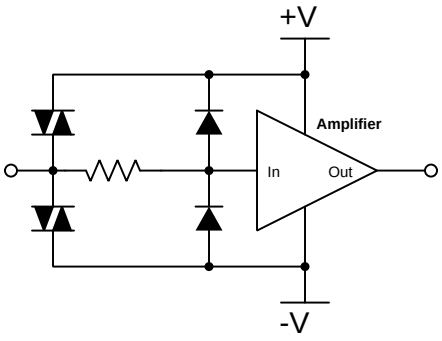
For some devices it is not sufficient for a diode network to clamp the input voltage to some value(s) slightly in excess of the device's power supply rail potential(s). For example, a device operating on a 5 Volt DC supply may tolerate input voltages no lower than zero and no greater than 5 Volts. A relatively simple solution to this problem is to provide the diode network with a set of "protection rails" just shy of the actual power supply rail potentials. Consider the following example as an illustration of this technique:



Here, resistors R_1, R_2, and R_3 form a voltage divider to produce two electrical potentials (labeled **A** and **B**), each one diode forward-voltage drop shy of the nearest rail potentials. For example, if the DC power supply output 5 Volts and the two protection diodes each dropped 0.7 Volts when forward-biased, the voltage divider would need to be designed such that point **A** was +4.3 Volts and point **B** was +0.7 Volts with respect to the DC supply's negative terminal. This would prompt the upper diode to turn on and clamp the amplifier's input potential to +5 Volts if ever the input potential exceeded +5 Volts, the limit resistor dropping the rest. Likewise, the lower diode would turn on if ever the input terminal's potential fell below the power supply's negative rail, clamping the amplifier's input terminal potential to exactly the same as that negative rail with R_{limit} dropping the rest.

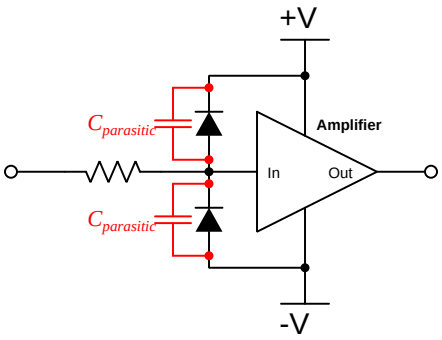
An important caveat to this strategy is that the voltage divider resistors R_1 and R_3 must be relatively small compared to the resistance of R_{limit} in order to ensure that those "protection rail" potentials **A** and **B** do not vary significantly when the protection diodes begin to conduct. A general engineering design principle here is to size R_{limit} at least ten times greater than either R_1 or R_3. If we size the resistors properly, this voltage-divider-based protection strategy may even be made *adjustable* by incorporating potentiometers into the voltage-divider network.

When protection must be provided against extraordinarily strong sources, additional protection may be added in their form of *DIACs* connected between each input terminal and power supply rail:



DIACs are classified as *thyristors*, because once triggered into an electrically conductive state by sufficiently high voltage (and/or sufficiently high *rate of change* of voltage, $\frac{dV}{dt}$) they will remain "on" so long as sufficient "holding" current passes through them even if voltage falls far below the initial triggering value. In summary, a thyristor acts as a very effective snubbing device to tame over-voltage conditions, essentially acting as a near-short to that offending source.

Protection networks, however, are not without their disadvantages. Chief among these is their tendency to adversely affect the signal being sensed by the amplifier being protected. Ideally a protection network should not corrupt the sensed signal at all, and only come into play if and when that signal becomes strong enough to pose threat of damage to the amplifier, but this would only be true if the protection diodes (and DIACs) had no parasitic properties when non-conducting. This is unfortunately untrue, as both diodes and DIACs exhibit *parasitic capacitance* which not only has the effect of storing electrical charge that the amplifier may interpret as a voltage that should not be present, but along with the protection resistor will form a *low-pass filter network* preventing the amplifier from being able to fully sense rapidly-changing input signals:



In other words, even with applied signals weak enough to pose absolutely no threat of harm to the amplifier, the capacitance inherent to the protection diodes will conspire with the series resistor

to "slow down" rates of rise and fall for any voltage arriving at the amplifier's input, thus making the amplifier "think" the signal isn't changing as rapidly as it really is.

In order to minimize the effects of parasitic capacitance within the protection diodes, we must choose diodes with as little of that capacitance as possible and also select a protection resistor with as low a value as possible that still limits maximum current to a value safe for the amplifier.

Chapter 6

Questions

This learning module, along with all others in the ModEL collection, is designed to be used in an inverted instructional environment where students independently read[1] the tutorials and attempt to answer questions on their own *prior* to the instructor's interaction with them. In place of lecture[2], the instructor engages with students in Socratic-style dialogue, probing and challenging their understanding of the subject matter through inquiry.

Answers are not provided for questions within this chapter, and this is by design. Solved problems may be found in the Tutorial and Derivation chapters, instead. The goal here is *independence*, and this requires students to be challenged in ways where others cannot think for them. Remember that you always have the tools of *experimentation* and *computer simulation* (e.g. SPICE) to explore concepts!

The following lists contain ideas for Socratic-style questions and challenges. Upon inspection, one will notice a strong theme of *metacognition* within these statements: they are designed to foster a regular habit of examining one's own thoughts as a means toward clearer thinking. As such these sample questions are useful both for instructor-led discussions as well as for self-study.

[1] Technical reading is an essential academic skill for any technical practitioner to possess for the simple reason that the most comprehensive, accurate, and useful information to be found for developing technical competence is in textual form. Technical careers in general are characterized by the need for continuous learning to remain current with standards and technology, and therefore any technical practitioner who cannot read well is handicapped in their professional development. An excellent resource for educators on improving students' reading prowess through intentional effort and strategy is the book textitReading For Understanding – How Reading Apprenticeship Improves Disciplinary Learning in Secondary and College Classrooms by Ruth Schoenbach, Cynthia Greenleaf, and Lynn Murphy.

[2] Lecture is popular as a teaching method because it is easy to implement: any reasonably articulate subject matter expert can talk to students, even with little preparation. However, it is also quite problematic. A good lecture always makes complicated concepts seem easier than they are, which is bad for students because it instills a false sense of confidence in their own understanding; reading and re-articulation requires more cognitive effort and serves to verify comprehension. A culture of teaching-by-lecture fosters a debilitating dependence upon direct personal instruction, whereas the challenges of modern life demand independent and critical thought made possible only by gathering information and perspectives from afar. Information presented in a lecture is ephemeral, easily lost to failures of memory and dictation; text is forever, and may be referenced at any time.

General challenges following tutorial reading

- Summarize as much of the text as you can in one paragraph of your own words. A helpful strategy is to explain ideas as you would for an intelligent child: as simple as you can without compromising too much accuracy.
- Simplify a particular section of the text, for example a paragraph or even a single sentence, so as to capture the same fundamental idea in fewer words.
- Where did the text make the most sense to you? What was it about the text's presentation that made it clear?
- Identify where it might be easy for someone to misunderstand the text, and explain why you think it could be confusing.
- Identify any new concept(s) presented in the text, and explain in your own words.
- Identify any familiar concept(s) such as physical laws or principles applied or referenced in the text.
- Devise a proof of concept experiment demonstrating an important principle, physical law, or technical innovation represented in the text.
- Devise an experiment to disprove a plausible misconception.
- Did the text reveal any misconceptions you might have harbored? If so, describe the misconception(s) and the reason(s) why you now know them to be incorrect.
- Describe any useful problem-solving strategies applied in the text.
- Devise a question of your own to challenge a reader's comprehension of the text.

General follow-up challenges for assigned problems

- Identify where any fundamental laws or principles apply to the solution of this problem, especially before applying any mathematical techniques.
- Devise a thought experiment to explore the characteristics of the problem scenario, applying known laws and principles to mentally model its behavior.
- Describe in detail your own strategy for solving this problem. How did you identify and organized the given information? Did you sketch any diagrams to help frame the problem?
- Is there more than one way to solve this problem? Which method seems best to you?
- Show the work you did in solving this problem, even if the solution is incomplete or incorrect.
- What would you say was the most challenging part of this problem, and why was it so?
- Was any important information missing from the problem which you had to research or recall?
- Was there any extraneous information presented within this problem? If so, what was it and why did it not matter?
- Examine someone else's solution to identify where they applied fundamental laws or principles.
- Simplify the problem from its given form and show how to solve this simpler version of it. Examples include eliminating certain variables or conditions, altering values to simpler (usually whole) numbers, applying a limiting case (i.e. altering a variable to some extreme or ultimate value).
- For quantitative problems, identify the real-world meaning of all intermediate calculations: their units of measurement, where they fit into the scenario at hand. Annotate any diagrams or illustrations with these calculated values.
- For quantitative problems, try approaching it qualitatively instead, thinking in terms of "increase" and "decrease" rather than definite values.
- For qualitative problems, try approaching it quantitatively instead, proposing simple numerical values for the variables.
- Were there any assumptions you made while solving this problem? Would your solution change if one of those assumptions were altered?
- Identify where it would be easy for someone to go astray in attempting to solve this problem.
- Formulate your own problem based on what you learned solving this one.

General follow-up challenges for experiments or projects

- In what way(s) was this experiment or project easy to complete?
- Identify some of the challenges you faced in completing this experiment or project.

- Show how thorough documentation assisted in the completion of this experiment or project.
- Which fundamental laws or principles are key to this system's function?
- Identify any way(s) in which one might obtain false or otherwise misleading measurements from test equipment in this system.
- What will happen if (component X) fails (open/shorted/etc.)?
- What would have to occur to make this system unsafe?

6.1 Conceptual reasoning

These questions are designed to stimulate your analytic and synthetic thinking[3]. In a Socratic discussion with your instructor, the goal is for these questions to prompt an extended dialogue where assumptions are revealed, conclusions are tested, and understanding is sharpened. Your instructor may also pose additional questions based on those assigned, in order to further probe and refine your conceptual understanding.

Questions that follow are presented to challenge and probe your understanding of various concepts presented in the tutorial. These questions are intended to serve as a guide for the Socratic dialogue between yourself and the instructor. Your instructor's task is to ensure you have a sound grasp of these concepts, and the questions contained in this document are merely a means to this end. Your instructor may, at his or her discretion, alter or substitute questions for the benefit of tailoring the discussion to each student's needs. The only absolute requirement is that each student is challenged and assessed at a level equal to or greater than that represented by the documented questions.

It is far more important that you convey your *reasoning* than it is to simply convey a correct answer. For this reason, you should refrain from researching other information sources to answer questions. What matters here is that *you* are doing the thinking. If the answer is incorrect, your instructor will work with you to correct it through proper reasoning. A correct answer without an adequate explanation of how you derived that answer is unacceptable, as it does not aid the learning or assessment process.

You will note a conspicuous lack of answers given for these conceptual questions. Unlike standard textbooks where answers to every other question are given somewhere toward the back of the book, here in these learning modules students must rely on other means to check their work. The best way by far is to debate the answers with fellow students and also with the instructor during the Socratic dialogue sessions intended to be used with these learning modules. Reasoning through challenging questions with other people is an excellent tool for developing strong reasoning skills.

Another means of checking your conceptual answers, where applicable, is to use circuit simulation software to explore the effects of changes made to circuits. For example, if one of these conceptual questions challenges you to predict the effects of altering some component parameter in a circuit, you may check the validity of your work by simulating that same parameter change within software and seeing if the results agree.

[3] *Analytical* thinking involves the "disassembly" of an idea into its constituent parts, analogous to dissection. *Synthetic* thinking involves the "assembly" of a new idea comprised of multiple concepts, analogous to construction. Both activities are high-level cognitive skills, extremely important for effective problem-solving, necessitating frequent challenge and regular practice to fully develop.

6.1.1 Reading outline and reflections

"Reading maketh a full man; conference a ready man; and writing an exact man" – Francis Bacon

Francis Bacon's advice is a blueprint for effective education: reading provides the learner with knowledge, writing focuses the learner's thoughts, and critical dialogue equips the learner to confidently communicate and apply their learning. Independent acquisition and application of knowledge is a powerful skill, well worth the effort to cultivate. To this end, students should read these educational resources closely, journal their own reflections on the reading, and discuss in detail their findings with classmates and instructor(s). You should be able to do all of the following after reading any instructional text:

☑ Briefly SUMMARIZE THE TEXT in the form of a journal entry documenting your learning as you progress through the course of study. Share this summary in dialogue with your classmates and instructor. Journaling is an excellent self-test of thorough reading because you cannot clearly express what you have not read or did not comprehend.

☑ Demonstrate ACTIVE READING STRATEGIES, including verbalizing your impressions as you read, simplifying long passages to convey the same ideas using fewer words, annotating text and illustrations with your own interpretations, working through mathematical examples shown in the text, cross-referencing passages with relevant illustrations and/or other passages, identifying problem-solving strategies applied by the author, etc. Technical reading is a special case of problem-solving, and so these strategies work precisely because they help solve any problem: paying attention to your own thoughts (metacognition), eliminating unnecessary complexities, identifying what makes sense, paying close attention to details, drawing connections between separated facts, and noting the successful strategies of others.

☑ Identify IMPORTANT THEMES, especially GENERAL LAWS and PRINCIPLES, expounded in the text and express them in the simplest of terms as though you were teaching an intelligent child. This emphasizes connections between related topics and develops your ability to communicate complex ideas to anyone.

☑ Form YOUR OWN QUESTIONS based on the reading, and then pose them to your instructor and classmates for their consideration. Anticipate both correct and incorrect answers, the incorrect answer(s) assuming one or more plausible misconceptions. This helps you view the subject from different perspectives to grasp it more fully.

☑ Devise EXPERIMENTS to test claims presented in the reading, or to disprove misconceptions. Predict possible outcomes of these experiments, and evaluate their meanings: what result(s) would confirm, and what would constitute disproof? Running mental simulations and evaluating results is essential to scientific and diagnostic reasoning.

☑ Specifically identify any points you found CONFUSING. The reason for doing this is to help diagnose misconceptions and overcome barriers to learning.

6.1.2 Foundational concepts

Correct analysis and diagnosis of electric circuits begins with a proper understanding of some basic concepts. The following is a list of some important concepts referenced in this module's full tutorial. Define each of them in your own words, and be prepared to illustrate each of these concepts with a description of a practical example and/or a live demonstration.

Thought experiments as a problem-solving strategy

Truth table

Boolean algebra

Logic function

Logic state

AND function

NOT function

NAND function

Logic gate

Pullup/pulldown resistor

Interposing

Sourcing

Sinking

Bipolar logic

CMOS logic

BiCMOS logic

TTL

V_{CC}

V_{EE}

V_{DD}

V_{SS}

Totem-pole output

Open-collector output

Open-drain output

Tri-state logic

Electrically common points

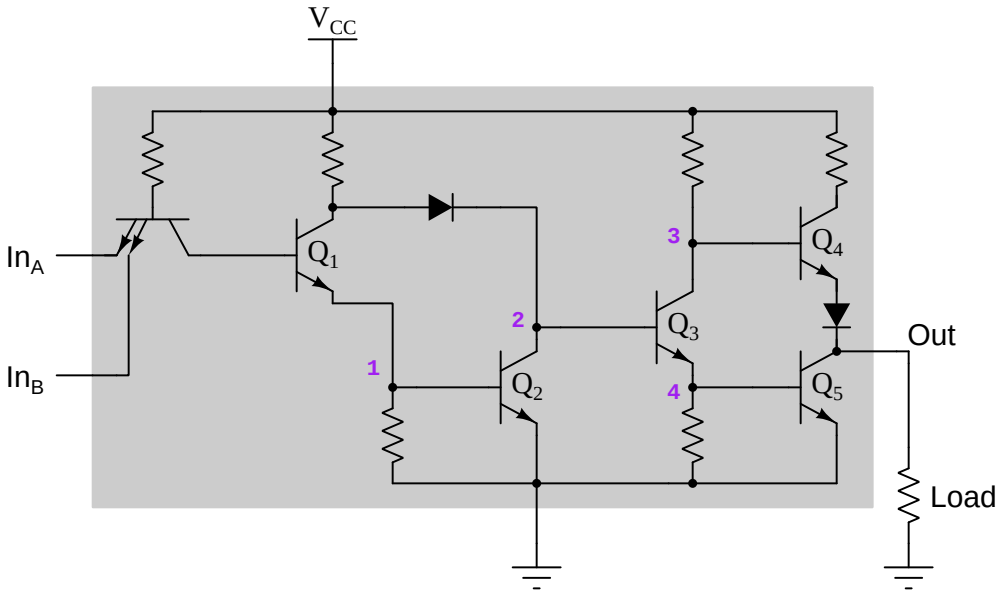
Wired-AND function

Hysteresis

Propagation delay

6.1.3 Discrete analysis of a bipolar AND gate

Identify the on/off states of the following transistors within this two-input Bipolar (TTL) AND gate (either *on* or *off*) for all combinations of input states. Note that one row of the table has already been completed for you:



$\mathbf{In}_A$	$\mathbf{In}_B$	$\mathbf{Q}_1$	$\mathbf{Q}_2$	$\mathbf{Q}_3$	$\mathbf{Q}_4$	$\mathbf{Q}_5$	**Out**
0	0						
0	1						
1	0						
1	1	On	On	Off	On	Off	1

Note: remember that the double-emitter "transistor" is not really an amplifier in this circuit, but rather is merely a "steering diode" network: two PN diode junctions from the "base" terminal pointing outward to the two "emitter" terminals, and one PN diode junction from the "base" terminal pointing outward to the "collector" terminal and to the base of Q_1. IC manufacturers do this because it is easier for them to "copy and paste" bipolar transistor structures in their IC layouts than to build in sets of stand-alone diode junctions.

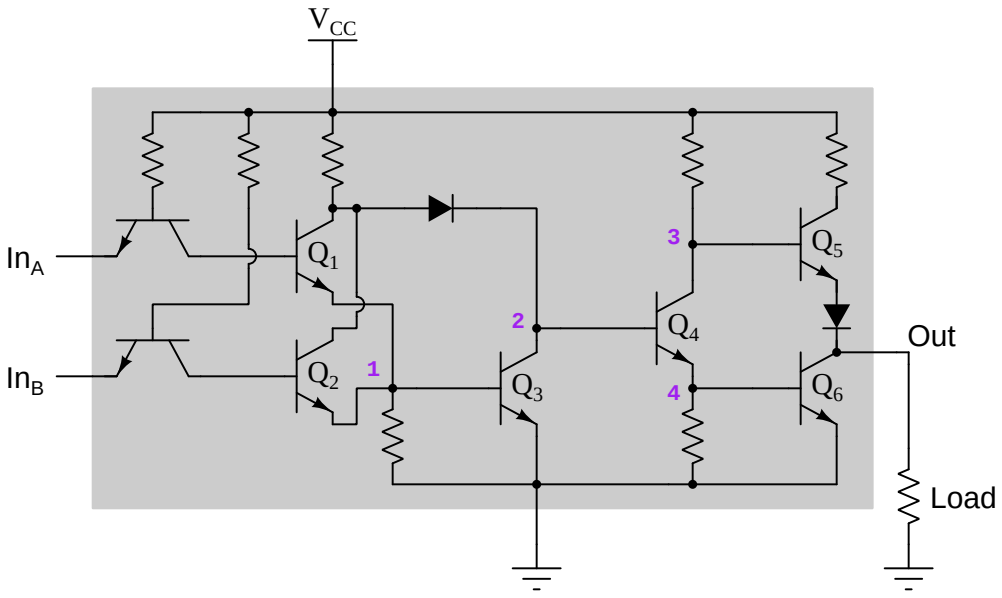
Challenges

- The unlabeled transistor is not actually functioning as a transistor, but rather represents the *steering diode* network for the gate's input. Explain how this "transistor" is able to function the same as a set of three diodes.

- How does the fact that knowing this is an AND gate assist you in your analysis of its internal states?

6.1.4 Discrete analysis of a bipolar OR gate

Identify the on/off states of the following transistors within this two-input Bipolar (TTL) OR gate (either *on* or *off*) for all combinations of input states. Note that one row of the table has already been completed for you:



$\mathbf{In}_A$	$\mathbf{In}_B$	$\mathbf{Q}_1$	$\mathbf{Q}_2$	$\mathbf{Q}_3$	$\mathbf{Q}_4$	$\mathbf{Q}_5$	$\mathbf{Q}_6$	**Out**
0	0							
0	1							
1	0							
1	1	On	On	On	Off	On	Off	1

Note: remember that the two "transistors" connecting to the gate's two input terminals are not really amplifiers in this circuit, but rather serve as "steering diode" networks: each "transistor" forming a PN diode junction from the "base" terminal pointing outward to the "emitter" terminal, and a PN diode junction from the "base" terminal pointing outward to the "collector" terminal and to the bases of the two real transistors Q_1 and Q_2. IC manufacturers do this because it is easier for them to "copy and paste" bipolar transistor structures in their IC layouts than to build in sets of stand-alone diode junctions.

Challenges

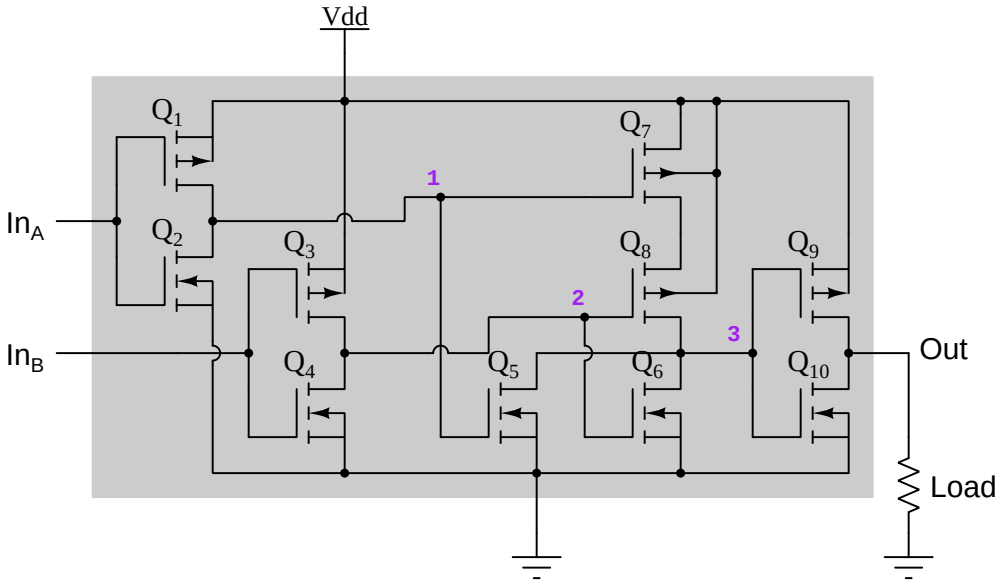
- The unlabeled transistors are not actually functioning as transistors, but rather represent the

steering diode networks for the gate's inputs. Explain how each of these "transistors" is able to function the same as a pair of diodes.

- How does the fact that knowing this is an OR gate assist you in your analysis of its internal states?

6.1.5 Discrete analysis of a CMOS NAND gate

Identify the logic states of the numbered points (either *1* or *0*) and the on/off states of the following transistors within this two-input CMOS NAND gate (either *on* or *off*), for all combinations of input states. Note that one row of the table has already been completed for you:



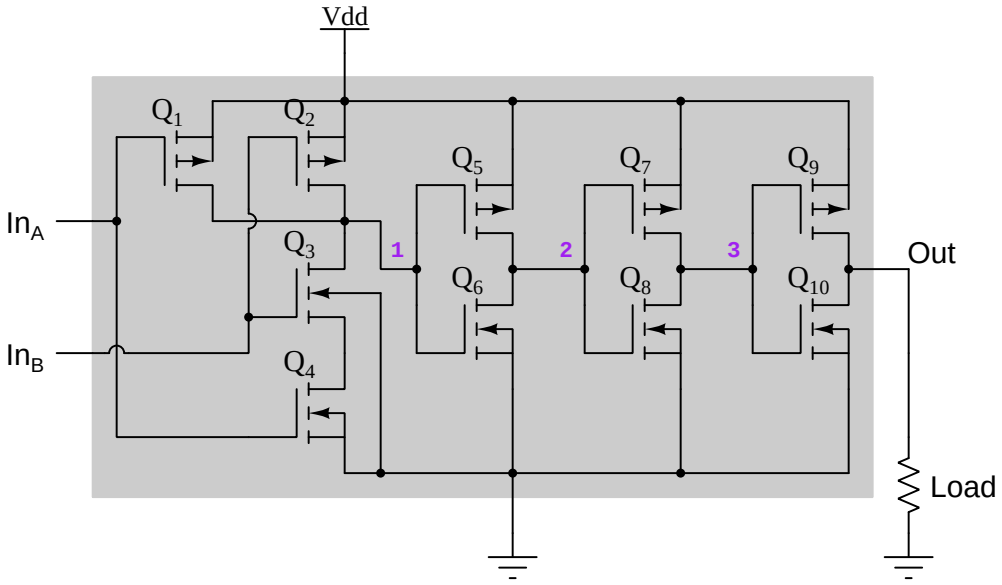
$\mathbf{In}_A$	$\mathbf{In}_B$	$\mathbf{Q}_1$	$\mathbf{Q}_2$	$\mathbf{Q}_3$	$\mathbf{Q}_4$	$\mathbf{Q}_5$	$\mathbf{Q}_6$	$\mathbf{Q}_7$	$\mathbf{Q}_8$	$\mathbf{Q}_9$	$\mathbf{Q}_{10}$	1	2	3	**Out**
0	0														
0	1														
1	0														
1	1	Off	On	Off	On	Off	Off	On	On	Off	On	0	0	1	0

Challenges

- How does the fact that knowing this is a NAND gate assist you in your analysis of its internal states?

6.1.6 Discrete analysis of a CMOS AND gate

Identify the logic states of the numbered points (either *1* or *0*) and the on/off states of the following transistors within this two-input CMOS AND gate (either *on* or *off*), for all combinations of input states. Note that one row of the table has already been completed for you:



$\mathbf{In}_A$	$\mathbf{In}_B$	$\mathbf{Q}_1$	$\mathbf{Q}_2$	$\mathbf{Q}_3$	$\mathbf{Q}_4$	$\mathbf{Q}_5$	$\mathbf{Q}_6$	$\mathbf{Q}_7$	$\mathbf{Q}_8$	$\mathbf{Q}_9$	$\mathbf{Q}_{10}$	1	2	3	**Out**
0	0														
0	1														
1	0														
1	1	Off	Off	On	On	On	Off	Off	On	On	Off	0	1	0	1

Challenges

- How does the fact that knowing this is an AND gate assist you in your analysis of its internal states?

6.1.7 Clashing gate outputs

It has been said that multiple logic gate outputs (with totem-pole output stages) should never be electrically connected, as this may result in the gates "fighting" each other. Sketch a diagram of such a "fight", tracing current through the warring output terminals and through the DC power supply.

Challenges

- How much current would you expect to flow between the "clashing" output terminals?
- Reference internal schematic diagrams for common bipolar or CMOS logic gates, and then trace the paths of current through the output transistors for two "warring" gates.
- It is unsafe to parallel gates with totem-pole outputs, and it is safe to parallel gates with open-collector or open-drain outputs. How about mixing these two? Is it safe to parallel the outputs of two logic gates if one of them has totem-pole output stage and the other is open-collector or open-drain?

6.1.8 Bipolar versus CMOS

Bipolar and CMOS transistor technologies are both used to construct semiconductor logic gates, and these two technologies differ in many performance aspects. Determine which of these technologies is generally superior in each of the following measures:

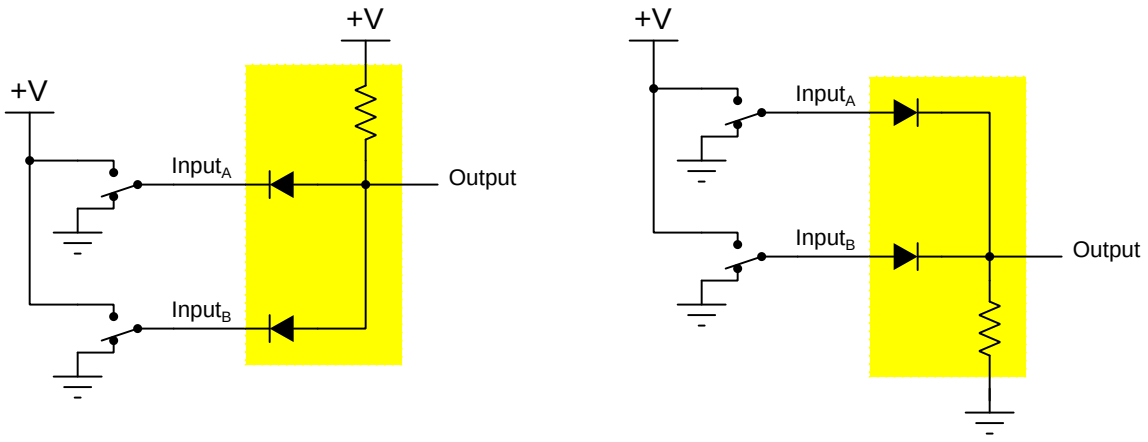
- Switching speed
- Power consumption
- Wide range of power supply voltage
- Resistance to ESD

Challenges

- For each of these choices, explain *why* they are true. Specifically, which principles of electricity informs your selections?

6.1.9 Diode-resistor logic gates

Crude logic gates circuits may be constructed out of nothing but diodes and resistors. Take for example the following diode-resistor logic circuits:



Identify the function represented by each of these gate circuits (XOR, NAND, NOR, AND, OR, NOT, etc.), and identify the sinking/sourcing nature of all input and output terminals.

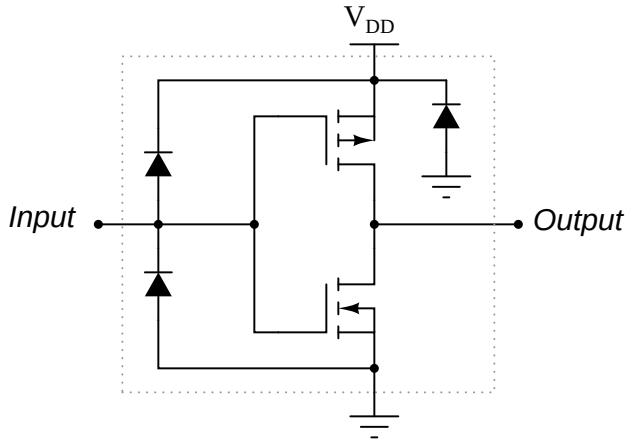
Also, identify how some of the foundational concepts you've studied apply to this circuit: **behavior of PN junctions**, **effects of opens vs. shorts**, **Kirchhoff's Voltage Law**, **sourcing vs. sinking currents**, **logic voltage levels**. Feel free to include any other foundational concepts not listed here.

Challenges

- Identify parameters that would define "high" and "low" logic levels for input and output signals, for both of these gates.
- Are these gates compatible with each other? For example, could we connect the output of one to the input of the other and create a functional "combinational" logic circuit?

6.1.10 CMOS protection diodes

Practical CMOS logic gates contain more than just MOSFETs. Here is a schematic diagram for a typical CMOS gate, complete with *protection diodes*:



First, identify what type of logic function this gate implements.

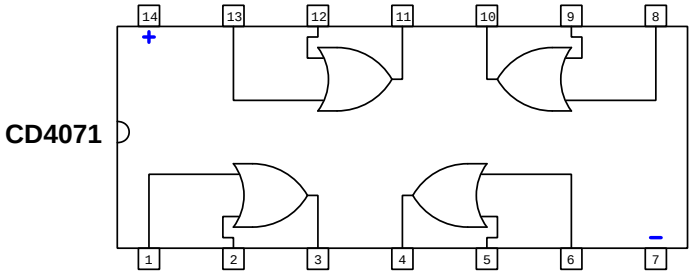
Explain what specific conditions each protection diode protects against.

Challenges

- It is important to realize these "protection" diodes do not allow circuit designers and builders to disregard good design practices with impunity. Identify an external condition that could damage this logic gate despite (or because of!) the protection diodes.
- Suppose the upper input protection diode failed open. How would this affect the operation of the logic gate, if at all?
- Suppose the upper input protection diode failed shorted. How would this affect the operation of the logic gate, if at all?
- Suppose the lower input protection diode failed open. How would this affect the operation of the logic gate, if at all?
- Suppose the lower input protection diode failed shorted. How would this affect the operation of the logic gate, if at all?
- Suppose the V_{DD} protection diode failed open. How would this affect the operation of the logic gate, if at all?
- Suppose the V_{DD} protection diode failed shorted. How would this affect the operation of the logic gate, if at all?

6.1.11 Sketching an OR gate circuit

Add necessary components and sketch wire connections needed so that an LED illuminates if either or both of two *normally-closed* pushbutton switches is pressed, using a CD4071 quad two-input OR gate IC:



Challenges

- What should be done with the pins on the unused gates within this IC?

6.2 Quantitative reasoning

These questions are designed to stimulate your computational thinking. In a Socratic discussion with your instructor, the goal is for these questions to reveal your mathematical approach(es) to problem-solving so that good technique and sound reasoning may be reinforced. Your instructor may also pose additional questions based on those assigned, in order to observe your problem-solving firsthand.

Mental arithmetic and estimations are strongly encouraged for all calculations, because without these abilities you will be unable to readily detect errors caused by calculator misuse (e.g. keystroke errors).

You will note a conspicuous lack of answers given for these quantitative questions. Unlike standard textbooks where answers to every other question are given somewhere toward the back of the book, here in these learning modules students must rely on other means to check their work. My advice is to use circuit simulation software such as SPICE to check the correctness of quantitative answers. Refer to those learning modules within this collection focusing on SPICE to see worked examples which you may use directly as practice problems for your own study, and/or as templates you may modify to run your own analyses and generate your own practice problems.

Completely worked example problems found in the Tutorial may also serve as "test cases[4]" for gaining proficiency in the use of circuit simulation software, and then once that proficiency is gained you will never need to rely[5] on an answer key!

[4] In other words, set up the circuit simulation software to analyze the same circuit examples found in the Tutorial. If the simulated results match the answers shown in the Tutorial, it confirms the simulation has properly run. If the simulated results disagree with the Tutorial's answers, something has been set up incorrectly in the simulation software. Using every Tutorial as practice in this way will quickly develop proficiency in the use of circuit simulation software.

[5] This approach is perfectly in keeping with the instructional philosophy of these learning modules: *teaching students to be self-sufficient thinkers*. Answer keys can be useful, but it is even more useful to your long-term success to have a set of tools on hand for checking your own work, because once you have left school and are on your own, there will no longer be "answer keys" available for the problems you will have to solve.

6.2.1 Miscellaneous physical constants

Note: constants shown in **bold** type are *exact*, not approximations. Values inside of parentheses show one standard deviation (σ) of uncertainty in the final digits: for example, the magnetic permeability of free space value given as $1.25663706212(19) \times 10^{-6}$ H/m represents a center value (i.e. the location parameter) of $1.25663706212 \times 10^{-6}$ Henrys per meter with one standard deviation of uncertainty equal to $0.0000000000019 \times 10^{-6}$ Henrys per meter.

Avogadro's number (N_A) = **6.02214076** $\times 10^{23}$ **per mole** (mol^{-1})

Boltzmann's constant (k) = **1.380649** $\times 10^{-23}$ **Joules per Kelvin** (J/K)

Electronic charge (e) = **1.602176634** $\times 10^{-19}$ **Coulomb** (C)

Faraday constant (F) = **96,485.33212...** $\times 10^{4}$ **Coulombs per mole** (C/mol)

Magnetic permeability of free space (μ_0) = $1.25663706212(19) \times 10^{-6}$ Henrys per meter (H/m)

Electric permittivity of free space (ϵ_0) = $8.8541878128(13) \times 10^{-12}$ Farads per meter (F/m)

Characteristic impedance of free space (Z_0) = 376.730313668(57) Ohms (Ω)

Gravitational constant (G) = $6.67430(15) \times 10^{-11}$ cubic meters per kilogram-seconds squared ($m^3/kg\text{-}s^2$)

Molar gas constant (R) = **8.314462618... Joules per mole-Kelvin** (J/mol-K) = 0.08205746(14) liters-atmospheres per mole-Kelvin

Planck constant (h) = **6.62607015** $\times 10^{-34}$ **joule-seconds** (J-s)

Stefan-Boltzmann constant (σ) = **5.670374419...** $\times 10^{-8}$ **Watts per square meter-Kelvin**^{4} ($W/m^2 \cdot K^4$)

Speed of light in a vacuum (c) = **299,792,458 meters per second** (m/s) = 186282.4 miles per second (mi/s)

Note: All constants taken from NIST data "Fundamental Physical Constants – Complete Listing", from `http://physics.nist.gov/constants`, National Institute of Standards and Technology (NIST), 2018 CODATA Adjustment.

6.2.2 Introduction to spreadsheets

A powerful computational tool you are encouraged to use in your work is a *spreadsheet*. Available on most personal computers (e.g. Microsoft Excel), *spreadsheet* software performs numerical calculations based on number values and formulae entered into cells of a grid. This grid is typically arranged as lettered columns and numbered rows, with each cell of the grid identified by its column/row coordinates (e.g. cell `B3`, cell `A8`). Each cell may contain a string of text, a number value, or a mathematical formula. The spreadsheet automatically updates the results of all mathematical formulae whenever the entered number values are changed. This means it is possible to set up a spreadsheet to perform a series of calculations on entered data, and those calculations will be re-done by the computer any time the data points are edited in any way.

For example, the following spreadsheet calculates average speed based on entered values of distance traveled and time elapsed:

	A	B	C	D
1	Distance traveled	46.9	Kilometers	
2	Time elapsed	1.18	Hours	
3	Average speed	= B1 / B2	km/h	
4				
5				

Text labels contained in cells `A1` through `A3` and cells `C1` through `C3` exist solely for readability and are not involved in any calculations. Cell `B1` contains a sample distance value while cell `B2` contains a sample time value. The formula for computing speed is contained in cell `B3`. Note how this formula begins with an "equals" symbol (`=`), references the values for distance and speed by lettered column and numbered row coordinates (`B1` and `B2`), and uses a forward slash symbol for division (`/`). The coordinates `B1` and `B2` function as *variables*[6] would in an algebraic formula.

When this spreadsheet is executed, the numerical value `39.74576` will appear in cell `B3` rather than the formula `= B1 / B2`, because 39.74576 is the computed speed value given 46.9 kilometers traveled over a period of 1.18 hours. If a different numerical value for distance is entered into cell `B1` or a different value for time is entered into cell `B2`, cell `B3`'s value will automatically update. All you need to do is set up the given values and any formulae into the spreadsheet, and the computer will do all the calculations for you.

Cell `B3` may be referenced by other formulae in the spreadsheet if desired, since it is a variable just like the given values contained in `B1` and `B2`. This means it is possible to set up an entire chain of calculations, one dependent on the result of another, in order to arrive at a final value. The arrangement of the given data and formulae need not follow any pattern on the grid, which means you may place them anywhere.

[6] Spreadsheets may also provide means to attach text labels to cells for use as variable names (Microsoft Excel simply calls these labels "names"), but for simple spreadsheets such as those shown here it's usually easier just to use the standard coordinate naming for each cell.

Common[7] arithmetic operations available for your use in a spreadsheet include the following:

- Addition (`+`)
- Subtraction (`-`)
- Multiplication (`*`)
- Division (`/`)
- Powers (`^`)
- Square roots (`sqrt()`)
- Logarithms (`ln()` , `log10()`)

Parentheses may be used to ensure[8] proper order of operations within a complex formula. Consider this example of a spreadsheet implementing the *quadratic formula*, used to solve for roots of a polynomial expression in the form of $ax^2 + bx + c$:

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

	A	B
1	x_1	= (-B4 + sqrt((B4^2) - (4*B3*B5))) / (2*B3)
2	x_2	= (-B4 - sqrt((B4^2) - (4*B3*B5))) / (2*B3)
3	a =	9
4	b =	5
5	c =	-2

This example is configured to compute roots[9] of the polynomial $9x^2 + 5x - 2$ because the values of 9, 5, and -2 have been inserted into cells `B3`, `B4`, and `B5`, respectively. Once this spreadsheet has been built, though, it may be used to calculate the roots of *any* second-degree polynomial expression simply by entering the new a, b, and c coefficients into cells `B3` through `B5`. The numerical values appearing in cells `B1` and `B2` will be automatically updated by the computer immediately following any changes made to the coefficients.

[7] Modern spreadsheet software offers a bewildering array of mathematical functions you may use in your computations. I recommend you consult the documentation for your particular spreadsheet for information on operations other than those listed here.

[8] Spreadsheet programs, like text-based programming languages, are designed to follow standard order of operations by default. However, my personal preference is to use parentheses even where strictly unnecessary just to make it clear to any other person viewing the formula what the intended order of operations is.

[9] Reviewing some algebra here, a *root* is a value for x that yields an overall value of zero for the polynomial. For this polynomial ($9x^2 + 5x - 2$) the two roots happen to be $x = 0.269381$ and $x = -0.82494$, with these values displayed in cells `B1` and `B2`, respectively upon execution of the spreadsheet.

Alternatively, one could break up the long quadratic formula into smaller pieces like this:

$$y = \sqrt{b^2 - 4ac} \qquad z = 2a$$

$$x = \frac{-b \pm y}{z}$$

	A	B	C
1	x_1	= (-B4 + C1) / C2	= sqrt((B4^2) - (4*B3*B5))
2	x_2	= (-B4 - C1) / C2	= 2*B3
3	a =	9	
4	b =	5	
5	c =	-2	

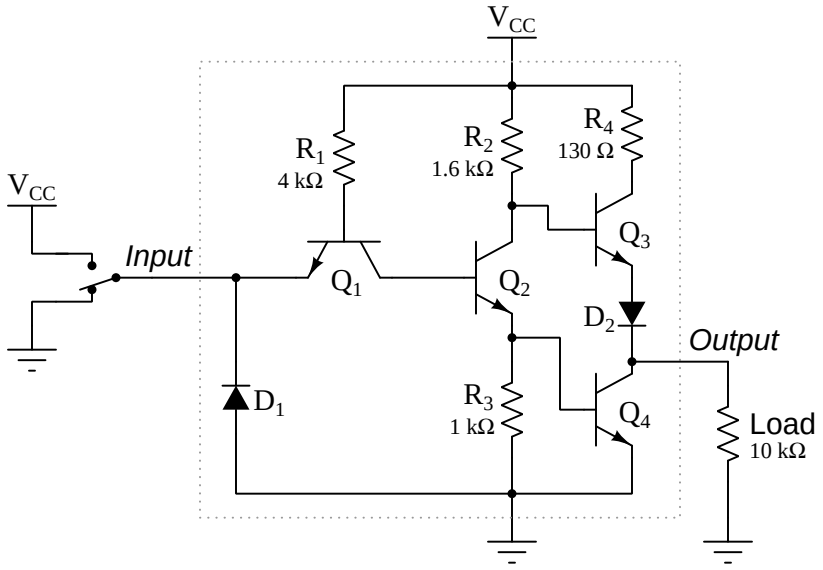
Note how the square-root term (y) is calculated in cell `C1`, and the denominator term (z) in cell `C2`. This makes the two final formulae (in cells `B1` and `B2`) simpler to interpret. The positioning of all these cells on the grid is completely arbitrary[10] – all that matters is that they properly reference each other in the formulae.

Spreadsheets are particularly useful for situations where the same set of calculations representing a circuit or other system must be repeated for different initial conditions. The power of a spreadsheet is that it automates what would otherwise be a tedious set of calculations. One specific application of this is to simulate the effects of various components within a circuit failing with abnormal values (e.g. a shorted resistor simulated by making its value nearly zero; an open resistor simulated by making its value extremely large). Another application is analyzing the behavior of a circuit design given new components that are out of specification, and/or aging components experiencing drift over time.

[10] My personal preference is to locate all the "given" data in the upper-left cells of the spreadsheet grid (each data point flanked by a sensible name in the cell to the left and units of measurement in the cell to the right as illustrated in the first distance/time spreadsheet example), sometimes coloring them in order to clearly distinguish which cells contain entered data versus which cells contain computed results from formulae. I like to place all formulae in cells below the given data, and try to arrange them in logical order so that anyone examining my spreadsheet will be able to figure out *how* I constructed a solution. This is a general principle I believe all computer programmers should follow: *document and arrange your code to make it easy for other people to learn from it.*

6.2.3 Voltages in a TTL gate

Identify the following voltages across each of the listed components within this logic circuit, assuming a power supply (V_{CC}) value of 5.0 Volts and the input switch in the position shown:



Note: remember that "transistor" Q_1 is not really an amplifier in this circuit, but rather is merely a "steering diode" network forming two PN diode junctions: one pointing from "base" to "emitter" and the other pointing from "base" to "collector". IC manufacturers do this because it is easier for them to "copy and paste" bipolar transistor structures in their IC layouts than to build in sets of individual diode junctions.

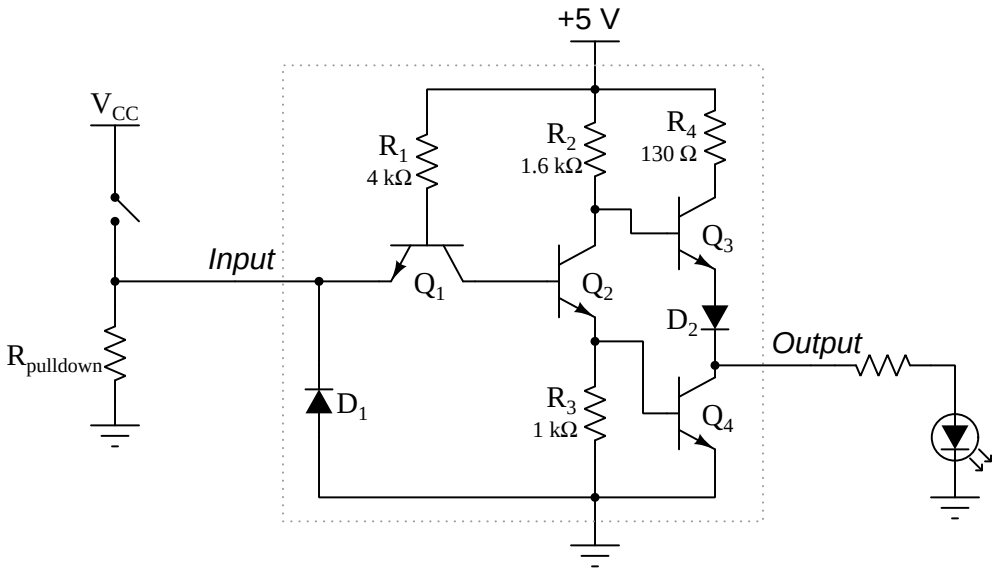
- V_{D1} =
- V_{D2} =
- V_{R1} =
- V_{R3} =

Challenges

- What logic state is naturally assumed by the gate if the input terminal is left "floating", unconnected to anything else?

6.2.4 Pulldown resistor sizing for a TTL gate input

Calculate an appropriate pulldown resistor size for this simple gate circuit:



Challenges

- All logic gate manufacturers specify acceptable voltage levels for logical "high" and "low" states. For classic TTL gates operating on a 5 Volt DC power supply, $V_{IH} = 2.0$ Volts and $V_{IL} = 0.8$ Volts. For classic CMOS gates operating on a 5 Volt DC power supply, $V_{IH} = 3.5$ Volts and $V_{IL} = 1.5$ Volts. Explain the significance of these ratings for the purpose of calculating necessary pullup and pulldown resistor sizes.

6.3 Diagnostic reasoning

These questions are designed to stimulate your deductive and inductive thinking, where you must apply general principles to specific scenarios (deductive) and also derive conclusions about the failed circuit from specific details (inductive). In a Socratic discussion with your instructor, the goal is for these questions to reinforce your recall and use of general circuit principles and also challenge your ability to integrate multiple symptoms into a sensible explanation of what's wrong in a circuit. Your instructor may also pose additional questions based on those assigned, in order to further challenge and sharpen your diagnostic abilities.

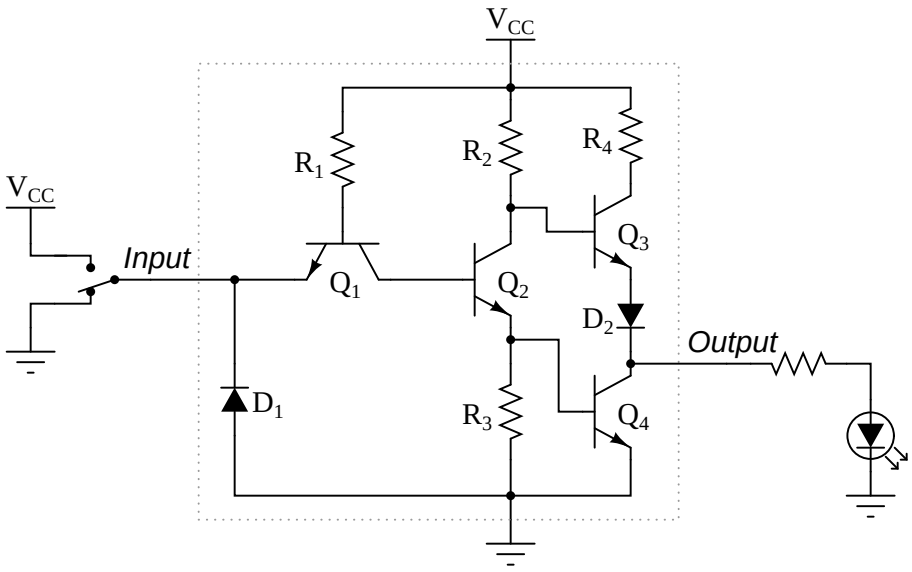
As always, your goal is to fully *explain* your analysis of each problem. Simply obtaining a correct answer is not good enough – you must also demonstrate sound reasoning in order to successfully complete the assignment. Your instructor's responsibility is to probe and challenge your understanding of the relevant principles and analytical processes in order to ensure you have a strong foundation upon which to build further understanding.

You will note a conspicuous lack of answers given for these diagnostic questions. Unlike standard textbooks where answers to every other question are given somewhere toward the back of the book, here in these learning modules students must rely on other means to check their work. The best way by far is to debate the answers with fellow students and also with the instructor during the Socratic dialogue sessions intended to be used with these learning modules. Reasoning through challenging questions with other people is an excellent tool for developing strong reasoning skills.

Another means of checking your diagnostic answers, where applicable, is to use circuit simulation software to explore the effects of faults placed in circuits. For example, if one of these diagnostic questions requires that you predict the effect of an open or a short in a circuit, you may check the validity of your work by simulating that same fault (substituting a very high resistance in place of that component for an open, and substituting a very low resistance for a short) within software and seeing if the results agree.

6.3.1 Proposed faults in a TTL gate

Identify this logic circuit, and predict the effects of the following faults. Consider each fault independently (i.e. one at a time, no coincidental faults):



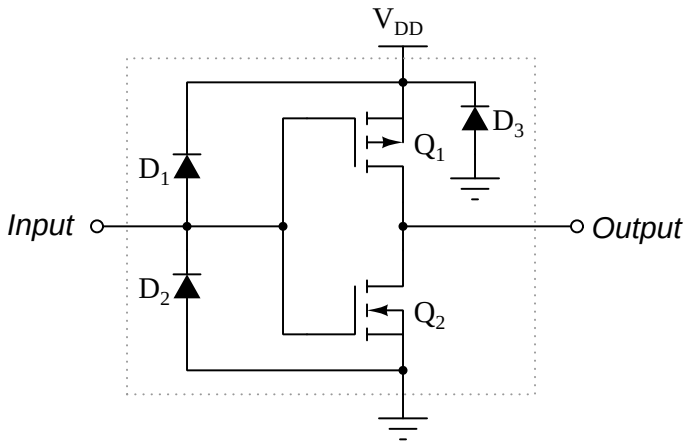
- Diode D_1 fails open:
- Diode D_1 fails shorted:
- Diode D_2 fails open:
- Resistor R_1 fails open:
- Resistor R_2 fails open:
- Resistor R_4 fails open:

Challenges

- Transistor Q_1 is not actually functioning as a transistor, but rather represents the *steering diode* network for the gate's input. Explain how this transistor is able to function the same as two diodes.

6.3.2 Proposed faults in CMOS gate

Predict how the operation of this logic gate circuit will be affected as a result of the following faults. Consider each fault independently (i.e. one at a time, no multiple faults):



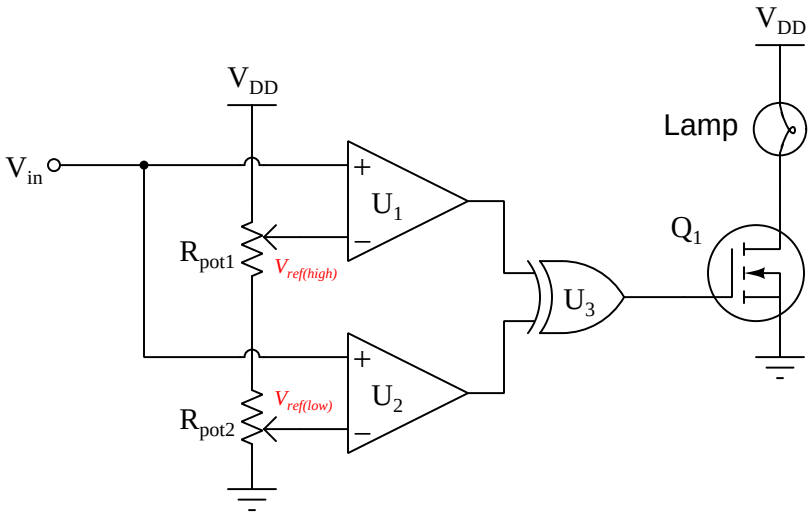
- Diode D_1 fails open
- Diode D_1 fails shorted
- Diode D_2 fails open
- Diode D_2 fails shorted
- Transistor Q_1 fails open (drain to source)
- Transistor Q_2 fails open (drain to source)

Challenges

- Diodes D_1 and D_2 are typically referred to as *input protection diodes*. Why is this, and what exactly are they protecting the IC against?

6.3.3 Effect of faulted IC outputs

This circuit is supposed to energize a lamp when the input voltage (V_{in}) falls between the two reference voltages set by R_{pot1} and R_{pot2}. Predict how the operation of this circuit will be affected as a result of the following faults. Consider each fault independently (i.e. one at a time, no multiple faults):



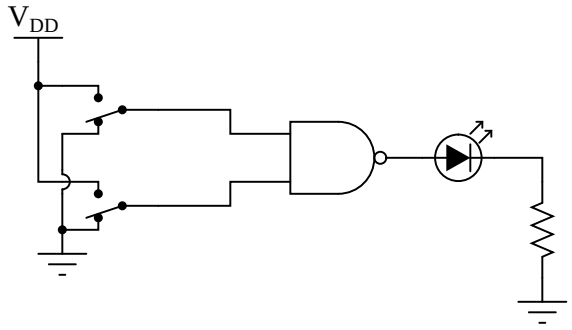
- Comparator U_1 output fails low
- Comparator U_1 output fails high
- Comparator U_2 output fails low
- Comparator U_2 output fails high
- Wire connecting V_{DD} to R_{pot1} fails open

Challenges

- Why is the logic gate of the XOR type? Why not use AND or NOR or some other type of gate?

6.3.4 Improper NAND gate function

A student builds the following circuit to demonstrate the behavior of a NAND gate:



When the student tests the circuit, though, something is wrong:

- Both switches LOW, no light.
- One switch HIGH, the other switch LOW; LED lights up.
- One switch LOW, the other switch HIGH; LED lights up.
- Both switches HIGH, no light.

Instead of acting as a NAND gate should, it seems to behave as if it were an Exclusive-OR gate! Examining the circuit for mistakes, the student discovers missing power connections to the chip – in other words, neither V_{DD} nor V_{SS} are connected to the power source.

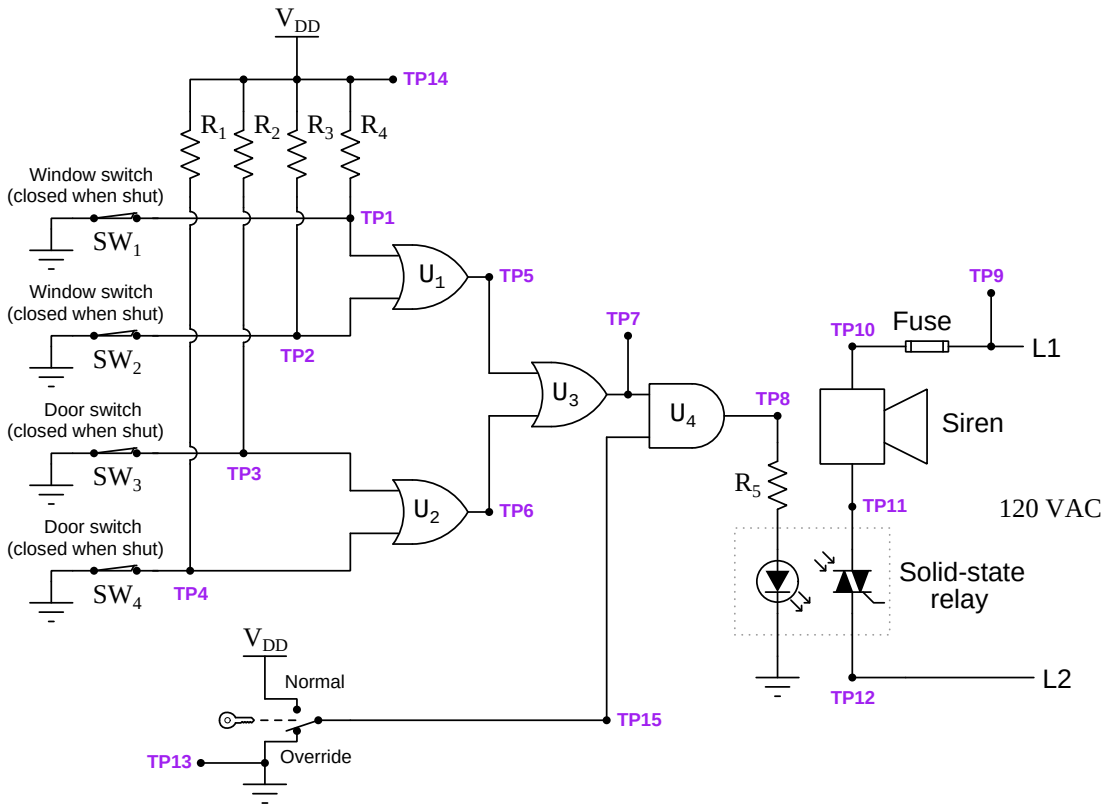
While this certainly is a problem, the student is left to wonder, *How did the circuit ever function at all?* With no power connected to the chip, how is it possible that the LED ever lit in *any* condition?

Challenges

- Explain how you would go about properly sizing the LED's resistor.

6.3.5 Malfunctioning security alarm system

Something is wrong with this security alarm system circuit. The alarm siren refuses to energize even when all windows and doors are opened:



Using your logic probe, you measure a high signal at TP1, a high signal at TP15, and a low signal at TP8 with all windows and doors propped open, and with the key switch in the "normal" position. From this information, identify two possible faults (either one of which could account for the problem and all measured values in this circuit). Then, choose one of those possible faults and explain why you think it could be to blame. The circuit elements you identify as possibly faulted can be wires, traces, and connections as well as components. Be as specific as you can in your answers, identifying both the circuit element and the type of fault.

Circuit elements that are possibly faulted

1.

2.

Next, explain *why* you think the above-listed faults are possible.

Challenges

- A good strategy to begin with when troubleshooting any malfunctioning system is to identify the expected signal states for a properly-functioning system. Do so for this system, and explain why this is a good diagnostic step.
- Should gate U_4 have a totem-pole output stage or an open-drain output stage?
- Explain how the circuit designer should properly size resistor R_5 in this circuit.
- Explain how the circuit designer should properly size resistors R_1 through R_4 in this circuit.

Appendix A

Problem-Solving Strategies

The ability to solve complex problems is arguably one of the most valuable skills one can possess, and this skill is particularly important in any science-based discipline.

- Study principles, not procedures. Don't be satisfied with merely knowing how to compute solutions – learn *why* those solutions work.
- Identify what it is you need to solve, identify all relevant data, identify all units of measurement, identify any general principles or formulae linking the given information to the solution, and then identify any "missing pieces" to a solution. Annotate all diagrams with this data.
- Sketch a diagram to help visualize the problem. When building a real system, always devise a plan for that system and analyze its function *before* constructing it.
- Follow the units of measurement and meaning of every calculation. If you are ever performing mathematical calculations as part of a problem-solving procedure, and you find yourself unable to apply each and every intermediate result to some aspect of the problem, it means you don't understand what you are doing. Properly done, every mathematical result should have practical meaning for the problem, and not just be an abstract number. You should be able to identify the proper units of measurement for each and every calculated result, and show where that result fits into the problem.
- Perform "thought experiments" to explore the effects of different conditions for theoretical problems. When troubleshooting real systems, perform *diagnostic tests* rather than visually inspecting for faults, the best diagnostic test being the one giving you the most information about the nature and/or location of the fault with the fewest steps.
- Simplify the problem until the solution becomes obvious, and then use that obvious case as a model to follow in solving the more complex version of the problem.
- Check for exceptions to see if your solution is incorrect or incomplete. A good solution will work for *all* known conditions and criteria. A good example of this is the process of testing scientific hypotheses: the task of a scientist is not to find support for a new idea, but rather to *challenge* that new idea to see if it holds up under a battery of tests. The philosophical

principle of *reductio ad absurdum* (i.e. disproving a general idea by finding a specific case where it fails) is useful here.

- Work "backward" from a hypothetical solution to a new set of given conditions.
- Add quantities to problems that are qualitative in nature, because sometimes a little math helps illuminate the scenario.
- Sketch graphs illustrating how variables relate to each other. These may be quantitative (i.e. with realistic number values) or qualitative (i.e. simply showing increases and decreases).
- Treat quantitative problems as qualitative in order to discern the relative magnitudes and/or directions of change of the relevant variables. For example, try determining what happens if a certain variable were to increase or decrease before attempting to precisely calculate quantities: how will each of the dependent variables respond, by increasing, decreasing, or remaining the same as before?
- Consider limiting cases. This works especially well for qualitative problems where you need to determine which direction a variable will change. Take the given condition and magnify that condition to an extreme degree as a way of simplifying the direction of the system's response.
- Check your work. This means regularly testing your conclusions to see if they make sense. This does *not* mean repeating the same steps originally used to obtain the conclusion(s), but rather to use some other means to check validity. Simply repeating procedures often leads to *repeating the same errors* if any were made, which is why alternative paths are better.

Appendix B

Instructional philosophy

"The unexamined circuit is not worth energizing" – Socrates (if he had taught electricity)

These learning modules, although useful for self-study, were designed to be used in a formal learning environment where a subject-matter expert challenges students to digest the content and exercise their critical thinking abilities in the answering of questions and in the construction and testing of working circuits.

The following principles inform the instructional and assessment philosophies embodied in these learning modules:

- The first goal of education is to enhance clear and independent thought, in order that every student reach their fullest potential in a highly complex and inter-dependent world. Robust reasoning is *always* more important than particulars of any subject matter, because its application is universal.
- Literacy is fundamental to independent learning and thought because text continues to be the most efficient way to communicate complex ideas over space and time. Those who cannot read with ease are limited in their ability to acquire knowledge and perspective.
- Articulate communication is fundamental to work that is complex and interdisciplinary.
- Faulty assumptions and poor reasoning are best corrected through challenge, not presentation. The rhetorical technique of *reductio ad absurdum* (disproving an assertion by exposing an absurdity) works well to discipline student's minds, not only to correct the problem at hand but also to learn how to detect and correct future errors.
- Important principles should be repeatedly explored and widely applied throughout a course of study, not only to reinforce their importance and help ensure their mastery, but also to showcase the interconnectedness and utility of knowledge.

These learning modules were expressly designed to be used in an "inverted" teaching environment[1] where students first read the introductory and tutorial chapters on their own, then individually attempt to answer the questions and construct working circuits according to the experiment and project guidelines. The instructor never lectures, but instead meets regularly with each individual student to review their progress, answer questions, identify misconceptions, and challenge the student to new depths of understanding through further questioning. Regular meetings between instructor and student should resemble a Socratic[2] dialogue, where questions serve as scalpels to dissect topics and expose assumptions. The student passes each module only after consistently demonstrating their ability to logically analyze and correctly apply all major concepts in each question or project/experiment. The instructor must be vigilant in probing each student's understanding to ensure they are truly *reasoning* and not just *memorizing*. This is why "Challenge" points appear throughout, as prompts for students to think deeper about topics and as starting points for instructor queries. Sometimes these challenge points require additional knowledge that hasn't been covered in the series to answer in full. This is okay, as the major purpose of the Challenges is to stimulate analysis and synthesis on the part of each student.

The instructor must possess enough mastery of the subject matter and awareness of students' reasoning to generate their own follow-up questions to practically any student response. Even completely correct answers given by the student should be challenged by the instructor for the purpose of having students practice articulating their thoughts and defending their reasoning. Conceptual errors committed by the student should be exposed and corrected not by direct instruction, but rather by reducing the errors to an absurdity[3] through well-chosen questions and thought experiments posed by the instructor. Becoming proficient at this style of instruction requires time and dedication, but the positive effects on critical thinking for both student and instructor are spectacular.

An inspection of these learning modules reveals certain unique characteristics. One of these is a bias toward thorough explanations in the tutorial chapters. Without a live instructor to explain concepts and applications to students, the text itself must fulfill this role. This philosophy results in lengthier explanations than what you might typically find in a textbook, each step of the reasoning process fully explained, including footnotes addressing common questions and concerns students raise while learning these concepts. Each tutorial seeks to not only explain each major concept in sufficient detail, but also to explain the logic of each concept and how each may be developed

[1] In a traditional teaching environment, students first encounter new information via *lecture* from an expert, and then independently apply that information via *homework*. In an "inverted" course of study, students first encounter new information via *homework*, and then independently apply that information under the scrutiny of an expert. The expert's role in lecture is to simply *explain*, but the expert's role in an inverted session is to *challenge*, *critique*, and if necessary *explain* where gaps in understanding still exist.

[2] Socrates is a figure in ancient Greek philosophy famous for his unflinching style of questioning. Although he authored no texts, he appears as a character in Plato's many writings. The essence of Socratic philosophy is to leave no question unexamined and no point of view unchallenged. While purists may argue a topic such as electric circuits is too narrow for a true Socratic-style dialogue, I would argue that the essential thought processes involved with scientific reasoning on *any* topic are not far removed from the Socratic ideal, and that students of electricity and electronics would do very well to challenge assumptions, pose thought experiments, identify fallacies, and otherwise employ the arsenal of critical thinking skills modeled by Socrates.

[3] This rhetorical technique is known by the Latin phrase *reductio ad absurdum*. The concept is to expose errors by counter-example, since only one solid counter-example is necessary to disprove a universal claim. As an example of this, consider the common misconception among beginning students of electricity that voltage cannot exist without current. One way to apply *reductio ad absurdum* to this statement is to ask how much current passes through a fully-charged battery connected to nothing (i.e. a clear example of voltage existing without current).

from "first principles". Again, this reflects the goal of developing clear and independent thought in students' minds, by showing how clear and logical thought was used to forge each concept. Students benefit from witnessing a model of clear thinking in action, and these tutorials strive to be just that.

Another characteristic of these learning modules is a lack of step-by-step instructions in the Project and Experiment chapters. Unlike many modern workbooks and laboratory guides where step-by-step instructions are prescribed for each experiment, these modules take the approach that students must learn to closely read the tutorials and apply their own reasoning to identify the appropriate experimental steps. Sometimes these steps are plainly declared in the text, just not as a set of enumerated points. At other times certain steps are implied, an example being assumed competence in test equipment use where the student should not need to be told *again* how to use their multimeter because that was thoroughly explained in previous lessons. In some circumstances no steps are given at all, leaving the entire procedure up to the student.

This lack of prescription is not a flaw, but rather a feature. Close reading and clear thinking are foundational principles of this learning series, and in keeping with this philosophy all activities are designed to *require* those behaviors. Some students may find the lack of prescription frustrating, because it demands more from them than what their previous educational experiences required. This frustration should be interpreted as an unfamiliarity with autonomous thinking, a problem which must be corrected if the student is ever to become a self-directed learner and effective problem-solver. Ultimately, the need for students to read closely and think clearly is more important both in the near-term and far-term than any specific facet of the subject matter at hand. If a student takes longer than expected to complete a module because they are forced to outline, digest, and reason on their own, so be it. The future gains enjoyed by developing this mental discipline will be well worth the additional effort and delay.

Another feature of these learning modules is that they do not treat topics in isolation. Rather, important concepts are introduced early in the series, and appear repeatedly as stepping-stones toward other concepts in subsequent modules. This helps to avoid the "compartmentalization" of knowledge, demonstrating the inter-connectedness of concepts and simultaneously reinforcing them. Each module is fairly complete in itself, reserving the beginning of its tutorial to a review of foundational concepts.

This methodology of assigning text-based modules to students for digestion and then using Socratic dialogue to assess progress and hone students' thinking was developed over a period of several years by the author with his Electronics and Instrumentation students at the two-year college level. While decidedly unconventional and sometimes even unsettling for students accustomed to a more passive lecture environment, this instructional philosophy has proven its ability to convey conceptual mastery, foster careful analysis, and enhance employability so much better than lecture that the author refuses to ever teach by lecture again.

Problems which often go undiagnosed in a lecture environment are laid bare in this "inverted" format where students must articulate and logically defend their reasoning. This, too, may be unsettling for students accustomed to lecture sessions where the instructor cannot tell for sure who comprehends and who does not, and this vulnerability necessitates sensitivity on the part of the "inverted" session instructor in order that students never feel discouraged by having their errors exposed. *Everyone* makes mistakes from time to time, and learning is a lifelong process! Part of the instructor's job is to build a culture of learning among the students where errors are not seen as shameful, but rather as opportunities for progress.

To this end, instructors managing courses based on these modules should adhere to the following principles:

- Student questions are always welcome and demand thorough, honest answers. The only type of question an instructor should refuse to answer is one the student should be able to easily answer on their own. Remember, *the fundamental goal of education is for each student to learn to think clearly and independently.* This requires hard work on the part of the student, which no instructor should ever circumvent. Anything done to bypass the student's responsibility to do that hard work ultimately limits that student's potential and thereby does real harm.

- It is not only permissible, but encouraged, to answer a student's question by asking questions in return, these follow-up questions designed to guide the student to reach a correct answer through their own reasoning.

- All student answers demand to be challenged by the instructor and/or by other students. This includes both correct and incorrect answers – the goal is to practice the articulation and defense of one's own reasoning.

- No reading assignment is deemed complete unless and until the student demonstrates their ability to accurately summarize the major points in their own terms. Recitation of the original text is unacceptable. This is why every module contains an "Outline and reflections" question as well as a "Foundational concepts" question in the Conceptual reasoning section, to prompt reflective reading.

- No assigned question is deemed answered unless and until the student demonstrates their ability to consistently and correctly apply the concepts to *variations* of that question. This is why module questions typically contain multiple "Challenges" suggesting different applications of the concept(s) as well as variations on the same theme(s). Instructors are encouraged to devise as many of their own "Challenges" as they are able, in order to have a multitude of ways ready to probe students' understanding.

- No assigned experiment or project is deemed complete unless and until the student demonstrates the task in action. If this cannot be done "live" before the instructor, video-recordings showing the demonstration are acceptable. All relevant safety precautions must be followed, all test equipment must be used correctly, and the student must be able to properly explain all results. The student must also successfully answer all Challenges presented by the instructor for that experiment or project.

Students learning from these modules would do well to abide by the following principles:

- No text should be considered fully and adequately read unless and until you can express every idea *in your own words, using your own examples.*
- You should always articulate your thoughts as you read the text, noting points of agreement, confusion, and epiphanies. Feel free to print the text on paper and then write your notes in the margins. Alternatively, keep a journal for your own reflections as you read. This is truly a helpful tool when digesting complicated concepts.
- Never take the easy path of highlighting or underlining important text. Instead, *summarize* and/or *comment* on the text using your own words. This actively engages your mind, allowing you to more clearly perceive points of confusion or misunderstanding on your own.
- A very helpful strategy when learning new concepts is to place yourself in the role of a teacher, if only as a mental exercise. Either explain what you have recently learned to someone else, or at least *imagine* yourself explaining what you have learned to someone else. The simple act of having to articulate new knowledge and skill forces you to take on a different perspective, and will help reveal weaknesses in your understanding.
- Perform each and every mathematical calculation and thought experiment shown in the text on your own, referring back to the text to see that your results agree. This may seem trivial and unnecessary, but it is critically important to ensuring you actually understand what is presented, especially when the concepts at hand are complicated and easy to misunderstand. Apply this same strategy to become proficient in the use of *circuit simulation software*, checking to see if your simulated results agree with the results shown in the text.
- Above all, recognize that learning is hard work, and that a certain level of frustration is unavoidable. There are times when you will struggle to grasp some of these concepts, and that struggle is a natural thing. Take heart that it will yield with persistent and varied[4] effort, and never give up!

Students interested in using these modules for self-study will also find them beneficial, although the onus of responsibility for thoroughly reading and answering questions will of course lie with that individual alone. If a qualified instructor is not available to challenge students, a workable alternative is for students to form study groups where they challenge[5] one another.

To high standards of education,

Tony R. Kuphaldt

[4] As the old saying goes, "Insanity is trying the same thing over and over again, expecting different results." If you find yourself stumped by something in the text, you should attempt a different approach. Alter the thought experiment, change the mathematical parameters, do whatever you can to see the problem in a slightly different light, and then the solution will often present itself more readily.

[5] Avoid the temptation to simply share answers with study partners, as this is really counter-productive to learning. Always bear in mind that the answer to any question is far less important in the long run than the method(s) used to obtain that answer. The goal of education is to empower one's life through the improvement of clear and independent thought, literacy, expression, and various practical skills.

Appendix C

Tools used

I am indebted to the developers of many open-source software applications in the creation of these learning modules. The following is a list of these applications with some commentary on each.

You will notice a theme common to many of these applications: a bias toward *code*. Although I am by no means an expert programmer in any computer language, I understand and appreciate the flexibility offered by code-based applications where the user (you) enters commands into a plain ASCII text file, which the software then reads and processes to create the final output. Code-based computer applications are by their very nature *extensible*, while WYSIWYG (What You See Is What You Get) applications are generally limited to whatever user interface the developer makes for you.

The `GNU/Linux` computer operating system

> There is so much to be said about Linus Torvalds' `Linux` and Richard Stallman's `GNU` project. First, to credit just these two individuals is to fail to do justice to the *mob* of passionate volunteers who contributed to make this amazing software a reality. I first learned of `Linux` back in 1996, and have been using this operating system on my personal computers almost exclusively since then. It is *free*, it is completely *configurable*, and it permits the continued use of highly efficient `Unix` applications and scripting languages (e.g. shell scripts, Makefiles, `sed`, `awk`) developed over many decades. `Linux` not only provided me with a powerful computing platform, but its open design served to inspire my life's work of creating open-source educational resources.

Bram Moolenaar's `Vim` text editor

> Writing code for any code-based computer application requires a *text editor*, which may be thought of as a word processor strictly limited to outputting plain-ASCII text files. Many good text editors exist, and one's choice of text editor seems to be a deeply personal matter within the programming world. I prefer `Vim` because it operates very similarly to `vi` which is ubiquitous on `Unix`/`Linux` operating systems, and because it may be entirely operated via keyboard (i.e. no mouse required) which makes it fast to use.

Donald Knuth's TEX typesetting system

Developed in the late 1970's and early 1980's by computer scientist extraordinaire Donald Knuth to typeset his multi-volume magnum opus *The Art of Computer Programming*, this software allows the production of formatted text for screen-viewing or paper printing, all by writing plain-text code to describe how the formatted text is supposed to appear. TEX is not just a markup language for documents, but it is also a Turing-complete programming language in and of itself, allowing useful algorithms to be created to control the production of documents. Simply put, TEX *is a programmer's approach to word processing*. Since TEX is controlled by code written in a plain-text file, this means anyone may read that plain-text file to see exactly how the document was created. This openness afforded by the code-based nature of TEX makes it relatively easy to learn how other people have created their own TEX documents. By contrast, examining a beautiful document created in a conventional WYSIWYG word processor such as Microsoft `Word` suggests nothing to the reader about *how* that document was created, or what the user might do to create something similar. As Mr. Knuth himself once quipped, conventional word processing applications should be called WYSIAYG (What You See Is *All* You Get).

Leslie Lamport's LATEX extensions to TEX

Like all true programming languages, TEX is inherently extensible. So, years after the release of TEX to the public, Leslie Lamport decided to create a massive extension allowing easier compilation of book-length documents. The result was LATEX, which is the markup language used to create all ModEL module documents. You could say that TEX is to LATEX as `C` is to `C++`. This means it is permissible to use any and all TEX commands within LATEX source code, and it all still works. Some of the features offered by LATEX that would be challenging to implement in TEX include automatic index and table-of-content creation.

Tim Edwards' `Xcircuit` drafting program

This wonderful program is what I use to create all the schematic diagrams and illustrations (but not photographic images or mathematical plots) throughout the ModEL project. It natively outputs PostScript format which is a true vector graphic format (this is why the images do not pixellate when you zoom in for a closer view), and it is so simple to use that I have never had to read the manual! Object libraries are easy to create for `Xcircuit`, being plain-text files using PostScript programming conventions. Over the years I have collected a large set of object libraries useful for drawing electrical and electronic schematics, pictorial diagrams, and other technical illustrations.

`Gimp` graphic image manipulation program

Essentially an open-source clone of Adobe's `PhotoShop`, I use `Gimp` to resize, crop, and convert file formats for all of the photographic images appearing in the ModEL modules. Although `Gimp` does offer its own scripting language (called `Script-Fu`), I have never had occasion to use it. Thus, my utilization of `Gimp` to merely crop, resize, and convert graphic images is akin to using a sword to slice bread.

`SPICE` circuit simulation program

`SPICE` is to circuit analysis as TEX is to document creation: it is a form of markup language designed to describe a certain object to be processed in plain-ASCII text. When the plain-text "source file" is compiled by the software, it outputs the final result. More modern circuit analysis tools certainly exist, but I prefer `SPICE` for the following reasons: it is *free*, it is *fast*, it is *reliable*, and it is a fantastic tool for *teaching* students of electricity and electronics how to write simple code. I happen to use rather old versions of `SPICE`, version 2g6 being my "go to" application when I only require text-based output. `NGSPICE` (version 26), which is based on Berkeley `SPICE` version 3f5, is used when I require graphical output for such things as time-domain waveforms and Bode plots. In all `SPICE` example netlists I strive to use coding conventions compatible with all `SPICE` versions.

Andrew D. Hwang's `ePiX` mathematical visualization programming library

This amazing project is a `C++` library you may link to any `C/C++` code for the purpose of generating PostScript graphic images of mathematical functions. As a completely free and open-source project, it does all the plotting I would otherwise use a Computer Algebra System (CAS) such as `Mathematica` or `Maple` to do. It should be said that `ePiX` is *not* a Computer Algebra System like `Mathematica` or `Maple`, but merely a mathematical *visualization* tool. In other words, it won't determine integrals for you (you'll have to implement that in your own `C/C++` code!), but it can graph the results, and it does so beautifully. What I really admire about `ePiX` is that it is a `C++` programming library, which means it builds on the existing power and toolset available with that programming language. Mr. Hwang could have probably developed his own stand-alone application for mathematical plotting, but by creating a `C++` library to do the same thing he accomplished something much greater.

`gnuplot` mathematical visualization software

Another open-source tool for mathematical visualization is `gnuplot`. Interestingly, this tool is *not* part of Richard Stallman's GNU project, its name being a coincidence. For this reason the authors prefer "gnu" *not* be capitalized at all to avoid confusion. This is a much "lighter-weight" alternative to a spreadsheet for plotting tabular data, and the fact that it easily outputs directly to an X11 console or a file in a number of different graphical formats (including PostScript) is very helpful. I typically set my `gnuplot` output format to default (X11 on my Linux PC) for quick viewing while I'm developing a visualization, then switch to PostScript file export once the visual is ready to include in the document(s) I'm writing. As with my use of `Gimp` to do rudimentary image editing, my use of `gnuplot` only scratches the surface of its capabilities, but the important points are that it's *free* and that it *works well*.

`Python` programming language

Both Python and C++ find extensive use in these modules as instructional aids and exercises, but I'm listing Python here as a *tool* for myself because I use it almost daily as a *calculator*. If you open a Python interpreter console and type `from math import *` you can type mathematical expressions and have it return results just as you would on a hand calculator. Complex-number (i.e. *phasor*) arithmetic is similarly supported if you include the complex-math library (`from cmath import *`). Examples of this are shown in the Programming References chapter (if included) in each module. Of course, being a fully-featured programming language, Python also supports conditionals, loops, and other structures useful for calculation of quantities. Also, running in a console environment where all entries and returned values show as text in a chronologically-ordered list makes it easy to copy-and-paste those calculations to document exactly how they were performed.

Appendix D

Creative Commons License

Creative Commons Attribution 4.0 International Public License

By exercising the Licensed Rights (defined below), You accept and agree to be bound by the terms and conditions of this Creative Commons Attribution 4.0 International Public License ("Public License"). To the extent this Public License may be interpreted as a contract, You are granted the Licensed Rights in consideration of Your acceptance of these terms and conditions, and the Licensor grants You such rights in consideration of benefits the Licensor receives from making the Licensed Material available under these terms and conditions.

Section 1 – Definitions.

a. **Adapted Material** means material subject to Copyright and Similar Rights that is derived from or based upon the Licensed Material and in which the Licensed Material is translated, altered, arranged, transformed, or otherwise modified in a manner requiring permission under the Copyright and Similar Rights held by the Licensor. For purposes of this Public License, where the Licensed Material is a musical work, performance, or sound recording, Adapted Material is always produced where the Licensed Material is synched in timed relation with a moving image.

b. **Adapter's License** means the license You apply to Your Copyright and Similar Rights in Your contributions to Adapted Material in accordance with the terms and conditions of this Public License.

c. **Copyright and Similar Rights** means copyright and/or similar rights closely related to copyright including, without limitation, performance, broadcast, sound recording, and Sui Generis Database Rights, without regard to how the rights are labeled or categorized. For purposes of this Public License, the rights specified in Section 2(b)(1)-(2) are not Copyright and Similar Rights.

d. **Effective Technological Measures** means those measures that, in the absence of proper authority, may not be circumvented under laws fulfilling obligations under Article 11 of the WIPO Copyright Treaty adopted on December 20, 1996, and/or similar international agreements.

e. **Exceptions and Limitations** means fair use, fair dealing, and/or any other exception or

limitation to Copyright and Similar Rights that applies to Your use of the Licensed Material.

f. **Licensed Material** means the artistic or literary work, database, or other material to which the Licensor applied this Public License.

g. **Licensed Rights** means the rights granted to You subject to the terms and conditions of this Public License, which are limited to all Copyright and Similar Rights that apply to Your use of the Licensed Material and that the Licensor has authority to license.

h. **Licensor** means the individual(s) or entity(ies) granting rights under this Public License.

i. **Share** means to provide material to the public by any means or process that requires permission under the Licensed Rights, such as reproduction, public display, public performance, distribution, dissemination, communication, or importation, and to make material available to the public including in ways that members of the public may access the material from a place and at a time individually chosen by them.

j. **Sui Generis Database Rights** means rights other than copyright resulting from Directive 96/9/EC of the European Parliament and of the Council of 11 March 1996 on the legal protection of databases, as amended and/or succeeded, as well as other essentially equivalent rights anywhere in the world.

k. **You** means the individual or entity exercising the Licensed Rights under this Public License. **Your** has a corresponding meaning.

Section 2 – Scope.

a. License grant.

1. Subject to the terms and conditions of this Public License, the Licensor hereby grants You a worldwide, royalty-free, non-sublicensable, non-exclusive, irrevocable license to exercise the Licensed Rights in the Licensed Material to:

A. reproduce and Share the Licensed Material, in whole or in part; and

B. produce, reproduce, and Share Adapted Material.

2. Exceptions and Limitations. For the avoidance of doubt, where Exceptions and Limitations apply to Your use, this Public License does not apply, and You do not need to comply with its terms and conditions.

3. Term. The term of this Public License is specified in Section 6(a).

4. Media and formats; technical modifications allowed. The Licensor authorizes You to exercise the Licensed Rights in all media and formats whether now known or hereafter created, and to make technical modifications necessary to do so. The Licensor waives and/or agrees not to assert any right or authority to forbid You from making technical modifications necessary to exercise the Licensed Rights, including technical modifications necessary to circumvent Effective Technological Measures.

For purposes of this Public License, simply making modifications authorized by this Section 2(a)(4) never produces Adapted Material.

5. Downstream recipients.

A. Offer from the Licensor – Licensed Material. Every recipient of the Licensed Material automatically receives an offer from the Licensor to exercise the Licensed Rights under the terms and conditions of this Public License.

B. No downstream restrictions. You may not offer or impose any additional or different terms or conditions on, or apply any Effective Technological Measures to, the Licensed Material if doing so restricts exercise of the Licensed Rights by any recipient of the Licensed Material.

6. No endorsement. Nothing in this Public License constitutes or may be construed as permission to assert or imply that You are, or that Your use of the Licensed Material is, connected with, or sponsored, endorsed, or granted official status by, the Licensor or others designated to receive attribution as provided in Section 3(a)(1)(A)(i).

b. Other rights.

1. Moral rights, such as the right of integrity, are not licensed under this Public License, nor are publicity, privacy, and/or other similar personality rights; however, to the extent possible, the Licensor waives and/or agrees not to assert any such rights held by the Licensor to the limited extent necessary to allow You to exercise the Licensed Rights, but not otherwise.

2. Patent and trademark rights are not licensed under this Public License.

3. To the extent possible, the Licensor waives any right to collect royalties from You for the exercise of the Licensed Rights, whether directly or through a collecting society under any voluntary or waivable statutory or compulsory licensing scheme. In all other cases the Licensor expressly reserves any right to collect such royalties.

Section 3 – License Conditions.

Your exercise of the Licensed Rights is expressly made subject to the following conditions.

a. Attribution.

1. If You Share the Licensed Material (including in modified form), You must:

A. retain the following if it is supplied by the Licensor with the Licensed Material:

i. identification of the creator(s) of the Licensed Material and any others designated to receive attribution, in any reasonable manner requested by the Licensor (including by pseudonym if designated);

ii. a copyright notice;

iii. a notice that refers to this Public License;

iv. a notice that refers to the disclaimer of warranties;

v. a URI or hyperlink to the Licensed Material to the extent reasonably practicable;

B. indicate if You modified the Licensed Material and retain an indication of any previous modifications; and

C. indicate the Licensed Material is licensed under this Public License, and include the text of, or the URI or hyperlink to, this Public License.

2. You may satisfy the conditions in Section 3(a)(1) in any reasonable manner based on the medium, means, and context in which You Share the Licensed Material. For example, it may be reasonable to satisfy the conditions by providing a URI or hyperlink to a resource that includes the required information.

3. If requested by the Licensor, You must remove any of the information required by Section 3(a)(1)(A) to the extent reasonably practicable.

4. If You Share Adapted Material You produce, the Adapter's License You apply must not prevent recipients of the Adapted Material from complying with this Public License.

Section 4 – Sui Generis Database Rights.

Where the Licensed Rights include Sui Generis Database Rights that apply to Your use of the Licensed Material:

a. for the avoidance of doubt, Section 2(a)(1) grants You the right to extract, reuse, reproduce, and Share all or a substantial portion of the contents of the database;

b. if You include all or a substantial portion of the database contents in a database in which You have Sui Generis Database Rights, then the database in which You have Sui Generis Database Rights (but not its individual contents) is Adapted Material; and

c. You must comply with the conditions in Section 3(a) if You Share all or a substantial portion of the contents of the database.

For the avoidance of doubt, this Section 4 supplements and does not replace Your obligations under this Public License where the Licensed Rights include other Copyright and Similar Rights.

Section 5 – Disclaimer of Warranties and Limitation of Liability.

a. Unless otherwise separately undertaken by the Licensor, to the extent possible, the Licensor offers the Licensed Material as-is and as-available, and makes no representations or warranties of any kind concerning the Licensed Material, whether express, implied, statutory, or other. This includes, without limitation, warranties of title, merchantability, fitness for a particular purpose, non-infringement, absence of latent or other defects, accuracy, or the presence or absence of errors,

whether or not known or discoverable. Where disclaimers of warranties are not allowed in full or in part, this disclaimer may not apply to You.

b. To the extent possible, in no event will the Licensor be liable to You on any legal theory (including, without limitation, negligence) or otherwise for any direct, special, indirect, incidental, consequential, punitive, exemplary, or other losses, costs, expenses, or damages arising out of this Public License or use of the Licensed Material, even if the Licensor has been advised of the possibility of such losses, costs, expenses, or damages. Where a limitation of liability is not allowed in full or in part, this limitation may not apply to You.

c. The disclaimer of warranties and limitation of liability provided above shall be interpreted in a manner that, to the extent possible, most closely approximates an absolute disclaimer and waiver of all liability.

Section 6 – Term and Termination.

a. This Public License applies for the term of the Copyright and Similar Rights licensed here. However, if You fail to comply with this Public License, then Your rights under this Public License terminate automatically.

b. Where Your right to use the Licensed Material has terminated under Section 6(a), it reinstates:

1. automatically as of the date the violation is cured, provided it is cured within 30 days of Your discovery of the violation; or

2. upon express reinstatement by the Licensor.

For the avoidance of doubt, this Section 6(b) does not affect any right the Licensor may have to seek remedies for Your violations of this Public License.

c. For the avoidance of doubt, the Licensor may also offer the Licensed Material under separate terms or conditions or stop distributing the Licensed Material at any time; however, doing so will not terminate this Public License.

d. Sections 1, 5, 6, 7, and 8 survive termination of this Public License.

Section 7 – Other Terms and Conditions.

a. The Licensor shall not be bound by any additional or different terms or conditions communicated by You unless expressly agreed.

b. Any arrangements, understandings, or agreements regarding the Licensed Material not stated herein are separate from and independent of the terms and conditions of this Public License.

Section 8 – Interpretation.

a. For the avoidance of doubt, this Public License does not, and shall not be interpreted to, reduce, limit, restrict, or impose conditions on any use of the Licensed Material that could lawfully

be made without permission under this Public License.

b. To the extent possible, if any provision of this Public License is deemed unenforceable, it shall be automatically reformed to the minimum extent necessary to make it enforceable. If the provision cannot be reformed, it shall be severed from this Public License without affecting the enforceability of the remaining terms and conditions.

c. No term or condition of this Public License will be waived and no failure to comply consented to unless expressly agreed to by the Licensor.

d. Nothing in this Public License constitutes or may be interpreted as a limitation upon, or waiver of, any privileges and immunities that apply to the Licensor or You, including from the legal processes of any jurisdiction or authority.

Creative Commons is not a party to its public licenses. Notwithstanding, Creative Commons may elect to apply one of its public licenses to material it publishes and in those instances will be considered the "Licensor." Except for the limited purpose of indicating that material is shared under a Creative Commons public license or as otherwise permitted by the Creative Commons policies published at `creativecommons.org/policies`, Creative Commons does not authorize the use of the trademark "Creative Commons" or any other trademark or logo of Creative Commons without its prior written consent including, without limitation, in connection with any unauthorized modifications to any of its public licenses or any other arrangements, understandings, or agreements concerning use of licensed material. For the avoidance of doubt, this paragraph does not form part of the public licenses.

Creative Commons may be contacted at `creativecommons.org`.

Appendix E

References

Bogart, Theodore F. Jr., *Introduction to Digital Circuits*, Glencoe division of Macmillan/McGraw-Hill, 1992.

"CD4069UB CMOS hex inverter", document SCHS054E, Texas Instruments Incorporated, Dallas, TX, January 2019.

"CD40106B CMOS Hex Schmitt-Trigger Inverters", document SCHS097F, Texas Instruments Incorporated, Dallas, TX, March 2017.

"CMOS NAND Gates", document SCHS021B, Texas Instruments, Dallas, TX, 2002.

Cockrill, Chris, "Understanding Schmitt Triggers", Application Report SCEA046, Texas Instruments Incorporated, Dallas, TX, September 2011.

Hall, Eldon C., "A Case History Of The AGC Integrated Logic Circuits", document E-1880, National Aeronautics and Space Administration, December 1965.

"Logic Guide – Logic Products", Texas Instruments, Dallas, TX, 2017.

"Logic Reference Guide – Bipolar, BiCMOS, and CMOS Logic Technology", document SCYB004B, Texas Instruments, Dallas, TX, 2004.

"Quadruple 2-Input Positive-NAND Gates", document SDLS025, Texas Instruments, Dallas, TX, 1999.

"Understanding and Interpreting Standard-Logic Data Sheets", Application Report SZZA036C, Texas Instruments, Dallas, TX, June 2016.

Appendix F

Version history

This is a list showing all significant additions, corrections, and other edits made to this learning module. Each entry is referenced by calendar date in reverse chronological order (newest version first), which appears on the front cover of every learning module for easy reference. Any contributors to this open-source document are listed here as well.

19 November 2024 – added more explanation about the "transistor" steering-diode network inside a TTL logic gate to those Conceptual Reasoning questions requiring analysis of transistors inside of TTL gates.

6-9 November 2024 – added more Challenging concepts to the Introduction chapter.

23-24 July 2024 – divided the Introduction chapter into two sections, one for students and one for instructors, and added content to the instructor section recommending learning outcomes and measures.

21 July 2024 – moved all content from the Case Tutorial section on the NAND gate demonstration circuit to a common file to be shared among multiple modules, and also added a TTL version to that demonstration to complement the CMOS.

8 March 2024 – added another page of text and illustrations to the "Voltage level translation" section of the Tutorial showing how a diode and pullup resistor may be used to translate incompatible "high" logic levels between gates.

22 January 2024 – added two new sections to divide up the existing Tutorial, one on open drain/collector gates and another on tri-state gates.

20 January 2024 – added a new Technical Reference section on protecting amplifier inputs against overvoltage from electro-static discharge (ESD) as well as electrical over-stress (EOS) from improperly connected signal sources.

5 January 2024 – added content to the Tutorial chapter regarding the outputs of logic gates, and how they are analogous to a SPDT switch connecting the output terminal to either the +V or the

Ground power supply terminals.

6 December 2023 – added content to the Tutorial chapter about Schmitt-trigger gates being used to make slow-rising and slow-falling pulse signals rise faster and fall faster.

14-16 September 2023 – added a Case Tutorial section showcasing a gallery of practical logic gate application circuits.

14-16 August 2023 – added some more explanatory text to the 'Example: comparator demonstration circuit" Case Tutorial section, as well as more explanatory text and illustrations on comparators in the Tutorial chapter. Added more questions to the Questions chapter. Also corrected some assorted typographical (spelling) errors.

24 April 2023 – added "Out" column in tables for qualitative logic gate analysis questions.

15 January 2023 – added comments about logic families, as well as a new Technical Reference section listing common logic families and their characteristics.

14 December 2022 – added photographs of 14-pin DIP logic gate ICs to the Tutorial.

7 December 2022 – added a new Conceptual Reasoning question on sketching a logic circuit based on an IC pinout. Also added reference in the Tutorial to passive sign convention.

28 November 2022 – placed questions at the top of the itemized list in the Introduction chapter prompting students to devise experiments related to the tutorial content.

20 July 2022 – corrected an error in image_2841 by swapping polarities of the 2 Volt, 5 Volt, and 7 Volt sources in the lowest-left example.

8 May 2022 – added Digital Pulse Criteria to the Technical References chapter.

3 May 2022 – edited image_2478 to add time markers showing the initiating events for the start-up pulse generator and the switch debouncing.

25 April 2022 – added commentary in the Tutorial about the default "high" state of an unconnected TTL input and how that contrasts against unconnected CMOS inputs. Also summarized the differences between TTL and CMOS in a bullet-point list.

30 November 2021 – minor edits to the Tutorial, clarifying that TTL means bipolar transistor logic. Also added instructor notes to the "Voltages in a TTL gate" Quantitative Reasoning question.

7 November 2021 – added a new introductory section to the Tutorial on basic logic functions.

30 October 2021 – simplified the presentation on comparators in the Tutorial.

1 August 2021 – added Case Tutorial on the use of decoupling capacitors to stabilize IC power supply voltage.

30 July 2021 – minor edit to inverter symbol in Case Tutorial. Also, edited all references to the bicycle "seat" to now read "saddle".

26-27 July 2021 – added some Case Tutorial sections showing some basic logic gate demonstration circuits.

22 July 2021 – added some Foundational concept application to the "Diode-resistor logic gates" Conceptual Reasoning question.

16 July 2021 – added Case Tutorial section showing a basic comparator demonstration circuit.

2 July 2021 – added an analogy to the Tutorial helping to explain noise margin for logic level standards.

31 May 2021 – added a new Tutorial section on comparators, and a Case Tutorial chapter with a section on supervised switch circuit signal interpretation.

11 May 2021 – added a new Tutorial section on level-shifting.

9 May 2021 – commented out or deleted empty chapters.

6 May 2021 – divided the Tutorial into sections and added content on logic levels.

7-8 October 2020 – minor additions to the Introduction and the Tutorial.

29 September 2020 – significantly edited the Introduction chapter to make it more suitable as a pre-study guide and to provide cues useful to instructors leading "inverted" teaching sessions. Also fixed a broken cross-reference to the Derivations and Technical References chapter.

5 March 2020 – added notes about the steering-diode "transistor" in TTL logic gate input stages in some of the schematic diagrams. Also eliminated some voltage calculations in the "Voltages in a TTL gate" Quantitative Reasoning problem.

27 February 2020 – added more Foundational Concepts to the list in the Conceptual Reasoning section.

28 January 2020 – added more Foundational Concepts to the list in the Conceptual Reasoning section.

21 November 2019 – included references to propagation delay.

19 November 2019 – included references to bus contention and tri-state outputs.

18 November 2019 – included references to Schmitt trigger inputs.

16 June 2019 – minor edits to diagnostic questions, replacing "no multiple faults" with "no coincidental faults".

17 April 2019 – Added multiple Conceptual Reasoning questions determining logic states within

both TTL and CMOS gates, and a Quantitative Reasoning question on voltage levels within a bipolar logic gate.

16 April 2019 – corrected a silly typographical error: "push-pole" should have been "push-pull", and also added a Quantitative Reasoning question on calculating an appropriate pulldown resistor size for a TTL gate circuit.

27 March 2019 – added questions to Conceptual, Quantitative, and Diagnostic Reasoning sections.

13 March 2019 – added an experiment, demonstrating some fundamental logic function using IC gates.

3 March 2019 – Added more content to the Tutorial, and corrected a major error in the universal-NOR logic gate diagram shown in the "NASA's Apollo Guidance Computer" section.

1 March 2019 – Added more content to the Tutorial, specifically open-collector and open-drain gate design.

27 February 2019 – Added more content to the Tutorial, and renamed it "Tutorial" instead of "Simplified Tutorial". Simplified and Full Tutorial development will occur at some later date.

19 February 2019 – Added "Semiconductor" to the title.

5 February 2019 – Simplified Tutorial completed.

30 January 2019 – document first created.

Index