# Table of Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Overview</td>
<td>3</td>
</tr>
<tr>
<td>• Update:</td>
<td></td>
</tr>
<tr>
<td>Logic Probe</td>
<td>4</td>
</tr>
<tr>
<td>• Operation</td>
<td></td>
</tr>
<tr>
<td>• How it works</td>
<td></td>
</tr>
<tr>
<td>• Building one</td>
<td></td>
</tr>
<tr>
<td>Logic Level Inputs</td>
<td>9</td>
</tr>
<tr>
<td>Logic Level Outputs</td>
<td>11</td>
</tr>
<tr>
<td>Logic Pulse Inputs</td>
<td>12</td>
</tr>
<tr>
<td>• Bounce</td>
<td></td>
</tr>
<tr>
<td>Clock Generator</td>
<td>14</td>
</tr>
<tr>
<td>Parts</td>
<td>17</td>
</tr>
<tr>
<td>• UPDATE:</td>
<td></td>
</tr>
<tr>
<td>Series Index</td>
<td>20</td>
</tr>
</tbody>
</table>
When we work on electronic circuits, there are some tools that get frequent use. Things like multi-meters, oscilloscopes, and waveform generators. Working with digital circuit has some use for those, but brings with it different requirements that require different tools. We'll have a look at some of these. These are fairly simple and can be made quite easily.

Update:

When I designed, prototyped, and even when I laid out several revisions of the PCBs, my thought was always "these will be useful when people are playing around with the TTL chips I'll talk about in this guide series."

Then it hit me.

These are incredibly useful whenever you're hacking on any 5v microcontroller board. That includes

- ATmega328 boards such as Arduino UNO, Nano, Pro Mini
- ATmega32U4 boards, such as the Arduino Leonardo, Micro,
• ATmega2560 boards such as the Arduino Mega2560

If you want to use the I/O boards with a 3.3v microcontroller board, you generally can't do it directly. Instead, you have to use a level shifter. The majority of Adafruit's breakout boards have level shifters built-in, but these I/O boards don't. You are in luck, though: Adafruit sells a levelshifter breakout that can be used between your 3.3v board and the I/O boards.

---

Logic Probe

With analogue circuits, we need to measure voltages, currents, waveforms, and the like. With digital circuits we often only care whether a signal is logic high or low.

A logic probe allows you to peek in at logic signals inside an operating circuit on an as-desired basis; no need to wire up LEDs to monitor signals (although that's sometimes exactly what you want, and we'll cover that later in this guide.

You connect the probe's Vcc and ground to their counterparts in the circuit being tested and you use the probe connection to touch points in the circuit where you wish to know the logic level. Be careful not to short connections (e.g. adjacent pins of a chip) with the probe. With digital circuits nothing really bad should happen (usually), but you won't get an accurate measurement.

You can buy a logic probe for $25-$50 from Amazon, Mouser, or the like. But we're makers and there's not a lot to it. So let's make one.

After looking around the internet for a simple, cheap, and easy to build probe I found the article below. It describes (including schematic and build suggestions) a simple transistor based logic probe.

I replicated the schematic in Eagle so that I could make a PCB. While you should have a look at the original article, it assumes some knowledge of discrete transistor circuits. A simplified description of the circuit's operation is below.
Operation

You start by connecting the 5v and ground connections to the 5v and ground of the circuit being tested. If you are powering the probe with another supply, the will need to have a common ground as usual. Then you simply touch the probe connection to the signal you want to check. If it is a logic low (< ~1.2v) the L LED will light (I use red) and if it is a logic high (> ~2.4v) the H LED will light (for which I use green). If the probe is unconnected, neither LED should be lit.*

If the signal is switching between high and low the LEDs will flicker between the two, clearly toggling back and forth if the frequency is low, and both glowing at something less than full brightness if the frequency is higher. In this case the relative brightness can give you an idea of the duty cycle even though you can't observe it directly.

How it works

When the probe is unconnected, R1 and R2 set the voltage at the base of T1 such that it turns on just a bit which causes enough voltage drop across R5 to turn T3 turn on which keeps T4 off, meaning LED2 is dark. At the same time current running through T1 (and R3 and R5) is small and results in a small enough voltage across R3 to keep T2 from turning on, keeping LED1 dark.

When the probe is at a logic high (> ~2.4v) T1 is held strongly on, keeping T3 on which keeps T4 off and LED2 dark as before. In this situation the probe voltage will be higher than when unconnected so the current through T1, and hence R3 will be
higher. This means a bigger voltage drop across R3, which lowers the voltage at the base of T2 making it turn on (notice it is a PNP transistor, not an NPN like the others so it sort of works backwards), lighting LED1.

The final case is when the probe is at logic low (<~1.2v). This will cause T1 to turn off, turning off T3. This, in turn, lets the voltage at the base of T4 rise (though R7 and R9) enough to turn it on causing current to flow through LED2, lighting it. There's no current flowing through R3, thus no voltage drop and T2 stays off, keeping LED1 dark.

Note that the BC177 (or BC557) can not be replaced with a 2N3906. I tried since I had 3906s in stock.

One unanticipated feature of this probe is, since it is transistor based and not TTL based, it works fine with 3.3v circuits. You do have to power it with 5v, though. And make sure you have connected the grounds together.

* The high LED might light dimly when the probe is unconnected. That will depend on the LEDs you use. I have some very bright LEDs which are especially nice for this project and the H led glows quite brightly (about half it's "on" brightness) when the probe is unconnected. Changing R3 from the 15K that is specified in the original circuit to 1K (meaning the voltage drop is lower for a given current) makes it work as expected. So depending on your LED, you might need to play around with R3.

Building one

The circuit is quite simple to build on a piece of perfboard. That said, I've made a PCB shaped to work well as a probe (i.e. could be placed in a thin, transparent plastic tube or 3D printed case) that will be available to order on OshPark. You have to order 3 at a time there, so I plan to sell it on Tindie as a bare PCB, a kit, or assembled.

The power connection is easy: just solder on a couple wires (colour coding is a good idea) and terminate them as desired. I put a pair of female header pins on one using the make-your-own ribbon cable wires. One end gets soldered to the board, and the other can plug into my breadboard power supplies. I can easily use a couple long
male header pins to convert it to plug into the power rails of a breadboard. You can use longer wires with alligator clips on one end if that's more useful. You could use a little heavier connector and make replaceable ends. It really depends on how you want to connect it.

The probe is a bit more problematic. This probe is designed to hold in your hand and poke at signals points in your circuit. That means you want something rigid, that extends from the end of the PCB far enough to reach into the circuit. For the same reason, it needs to be fairly thin; you want to touch the signal you are interested in, but nothing nearby. My solution was to use the same kind of wire as above, one with a male end in this case. To stiffen it I used pieces of ziptie like a splint you'd use for a broken leg.

I cut the wire to the length I wanted and soldered the cut end to the probe connection on the board. I cut one of the zipties into pieces of a length that would leave the probe tip exposed and extend over the board slightly.
I then used a short piece of shrink-wrap to secure the pieces of ziptie to the probe wire.

Another piece of shrink-wrap was used to cover the entire thing (leaving the tip exposed) which secured the bits of ziptie to the wire.

As a final touch, I used another ziptie to secure the splint to the board.
Logic Level Inputs

We often need a way to provide inputs to a digital circuit. We can always do some ad-hoc wiring with jumpers, especially if the circuit is on a breadboard, but it is nicer to have a more stable, reusable system.

My solution uses a bank of 8 slide switches that switch between 0v and 5v. I've used moderately sized individual switches to make them easier to work with. A block of 8 DIP switches could be used instead. That would make the circuit smaller, but it would be more bothersome to change values.

To isolate the circuit being worked on from the noisy* slide/toggle switches a 74373 octal latch** is used. Once you set the switches, press the button to grab (aka latch) the now stable values. I didn't worry about the noise from the pushbutton since latching repeatedly isn't a concern. The output enable of the '373 is tied to ground so it's always enabled. Not only does the '373 work to clean up the logic levels, it also lets the switches appear to the circuit as a collection of pure logic levels from a TTL output. This makes everything nice and clean for the circuit being fed the signals.
Logic Level Outputs

If we have a way to put logic signals into a circuit, it makes sense that we will want a way to get them out. I've gone with the tried and true method of using the logic signals to drive a collection of LEDs. A 74244 is used to buffer the signals*. LEDs are power hungry and driving them from arbitrary signals in our circuit could draw too much current and cause incorrect behaviour. Using the buffer presents a single TTL load to the circuit while at the same time providing plenty of current to make the LEDs happy.

* See the section on pulse inputs for a discussion of switch noise/bounce.

** The 74373 will be discussed in detail in a later guide in this series.
* See a later guide in this series for more information on this handy chip.

Logic Pulse Inputs

So far we've talked about logic levels. These are relatively static; we are dealing with circuits that have response times measured in double digit numbers of nanoseconds, so the time it takes you to flip a switch is a really long time. We've mitigated the issue of bounce in the logic level input circuit by using a latch to capture switch states once they've settled (which happens very quickly in the human timescale).

Sometimes, though we want a pulse: a signal briefly changing to the opposite logic value and back. We could slide one of our switches back and forth, but that takes some random amount of time, and is messy. We could just use a momentary contact push-button switch. Better. But it's still messy.

Bounce

When I say messy, I'm referring to the change from one state to the other bouncing back and forth before settling into the new state.
This is just something that mechanical switches do. Logic circuits are so fast that they'll likely see each bounce as a separate pulse, which is pretty much what it is. That can be all kinds of not good. In order to use this as a single, clean change in logic state, we need to get rid of the bounce. That is to say, we need to debounce the switch.

There are a variety of approaches to this, many which try to remove the bounce using an RC circuit or flipflops, but for a momentary pushbutton I like the approach of using a 555. This also means that we can use a small, cheap, SPST pushbutton switch. The idea is to trigger on the initial bounce of the switch to produce a clean output pulse that ends (cleanly) shortly after the switch is released. Specifically, after it finishes bouncing back to its stable state. It's a simple, clean circuit... and it shows a great use of the 555 about which I wrote a previous guide.
The trigger input is held high by R1 until S1 is pressed. That bounces as expected, but the first time it goes below 1/3 Vcc the 555 triggers and its Q output goes high. Since the bipolar 555 is triggered by the input being below 1/3 Vcc and not (as in the CMOS version) the act of going below 1/3 Vcc, it stays in a triggered state with Q high until the switch is released. Then the RC timing (R2 & C2) takes over and holds Q high until the voltage on C2 reaches 2/3 Vcc at which time Q goes low. R2 and C2 are chosen to hold Q high until after the switch stops bouncing and settles into its released state. Notice that there’s an inverted version of the output made available for convenience.

Four copies of this circuit fit nicely onto my standard 40mm x 60mm piece of perfboard so that’s what I’ve used for the PCB as well. The 7404 has 6 NOT gates so we just need one of them (and even have a couple spare gates). Four pulse generators are probably plenty.

Clock Generator

The final thing we’ll make is a something to generate a regular series of pulses. There are many cases in digital circuits where we need a continuous sequence of pulses: generally called a clock.

A clock has two parameters we will want to be able to control:

1. frequency: how often the pulse occurs, measured in pulses per second, aka Hertz.
2. duty cycle: the relative amount of time the signal is high, versus the amount of time it is low. In other words: the width of the pulses, generally expressed as a percentage.
You can buy proper clock generators and give you fine, visible control over these and let you dial in exactly what you want. That is not this clock generator. We just need something basic that we can adjust somewhat. Back to our friend the 555. To make a debounced pulse generator we used it in monostable mode. For this we need to use it in astable mode.

The purpose of this circuit is not to generate high speed clock signals. In fact that wouldn't be overly useful. What we need to generate is a series of pulses, regular and continuous but slow enough (i.e. with a low enough frequency) that we can watch our circuit go through its various states.

The two 100K adjustable resistors that are used to adjust the frequency and duty cycle of the output. It's not so simple as one for adjusting frequency and the other for duty cycle. Recall how this circuit works*: C1 charges through R1 and R2; the output of the 555 is high during this time. When the voltage on C1 reaches 2/3 Vcc, it begins discharging through R2 (via the DIScharge pin of the 555). While it is discharging the output is low. Once the voltage on C2 reaches 1/3 Vcc, it starts charging again and the process repeats indefinitely. The frequency is the reciprocal of total time of a cycle (the time to charge plus the time to discharge): 1.44/((R1+2R2)C1). The duty cycle will be the ratio of the time to charge to the total time. Since the capacitor value and timing constant is in both time calculations it can be factored out leaving it just dependant on the two resistors: (R1 + R2)/(R1 + 2R2).

For example, if we set R1 to 30K and R2 to 15K the frequency will be:

\[
\frac{1.44}{((30000 + 2*15000) * 0.000047)}
\]

\[
1.44 / (60000 * 0.000047)
\]

\[
1.44 / 2.82
\]

\[
0.51 \text{ Hz}
\]

That's about 1 full cycle every 2 seconds. 
The duty cycle is:

\[
\frac{(30000 + 15000)}{(30000 + 2 \times 15000)} \quad \frac{(30000 + 15000)}{(30000 + 30000)} \quad \frac{45000}{60000} \quad .75
\]

which is 75%. That means the LED should be lit for 3/4 of the time, and dark for 1/4.

The LED was added so that there is visible feedback on the frequency and duty cycle.

The first two invertors (IC2A and IC2B) are used to avoid the LED impacting the circuit you are connecting the signal to. The first buffers the 555 output to provide plenty of current to drive the LED and the second isolates the circuit connected to the output.

The third invertor (IC2C) provides an inverted output. The way this circuit works, you get a duty cycle greater than 50% (i.e. a series of logic low pulses). If you want a duty cycle lower than 50%, the inverted output gives that to you (i.e. a series of logic high pulses).

Given the simplicity, usefulness, and low cost of the circuit and the fact that it uses 3 of the 6 NOT gates on the 7404, I've put 2 on the PCB I made.

* See my 555 tutorial for details.
Parts

On the side is a list of parts that would be handy to build the perfboard version of the tools like the ones I've included photos of. Unfortunately not all the parts are available at Adafruit and you will have to go to Digikey(), Mouser(), Arrow(), or the like. I am making PCBs for all the tools discussed in this guide which will be available at OshPark() and I am considering making kits available on Tindie. Check back here for details, or find me on Twitter, Facebook, Instagram, or the Adafruit Discord.

UPDATE:

I've shared all 5 boards on OshPark. I will also be offering kits (with all parts as well as an option for 3d printed cases) on Tindie.
Series Index

1. Binary, Boolean, and Logic
2. Some Tools
3. Combinational Circuits
4. Sequential Circuits
5. Memories
6. An EPROM Emulator
7. MCUs... how do they work?