# Guide Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Guide Contents</td>
<td>2</td>
</tr>
<tr>
<td>Overview</td>
<td>3</td>
</tr>
<tr>
<td>Required Parts</td>
<td>3</td>
</tr>
<tr>
<td>Before Starting</td>
<td>4</td>
</tr>
<tr>
<td>Digital Signals</td>
<td>5</td>
</tr>
<tr>
<td>3V Logic</td>
<td>5</td>
</tr>
<tr>
<td>Pocket Ref</td>
<td>6</td>
</tr>
<tr>
<td>Circuit Hookup</td>
<td>7</td>
</tr>
<tr>
<td>Hello Digital</td>
<td>10</td>
</tr>
<tr>
<td>Floating Inputs</td>
<td>13</td>
</tr>
<tr>
<td>Pull It Up</td>
<td>16</td>
</tr>
<tr>
<td>Pull It Down</td>
<td>18</td>
</tr>
<tr>
<td>Which Way?</td>
<td>20</td>
</tr>
<tr>
<td>Bouncing Inputs</td>
<td>21</td>
</tr>
<tr>
<td>Under The Hood</td>
<td>25</td>
</tr>
<tr>
<td>Questions and Code Challenges</td>
<td>28</td>
</tr>
<tr>
<td>Questions</td>
<td>28</td>
</tr>
<tr>
<td>Code Challenges</td>
<td>28</td>
</tr>
</tbody>
</table>
Overview

In this guide we will use our Circuit Playground along with some alligator clips and a resistor to explore how we can read digital inputs.

No soldering required!

Required Parts

You will need the following items.

Circuit Playground (http://adafruit.it/3000)
Before Starting

If you are new to the Circuit Playground, you may want to first read these overview guides.

- Overview (https://adafru.it/ncG)
- Lesson #0 (https://adafru.it/rb4)

This project will use the Arduino IDE. Make sure you have added the board support for the Circuit Playground as well as installed the Circuit Playground library. MUST DO BOTH. This is covered in the guides linked above.
Digital Signals

The term [digital signal](https://adafruit.it/uBi) can be used to describe a wide range of signal types. But in this guide we are going to use a very simple definition for a digital signal - a signal which has only two values. Such a signal would look something like this:

![Digital Signal Diagram](image)

The two values could be anything. However, since we are dealing with an electrical signal, we will be dealing with an electrical value - [voltage](https://adafruit.it/uBj). But what are the two values for the voltage?

Typically, 0V is used for one of them and is referred to as "LOW". The other one is referred to as "HIGH" and its voltage depends on the hardware ([https://adafruit.it/uBj](https://adafruit.it/uBj)), but some common values you'll see are 5V (ex: Arduino Uno) and 3.3V (ex: Circuit Playground).

### 3V Logic

Let's focus on the 3.3V logic used by the Circuit Playground. Quite often this is just referred to as "three volt logic" as we're all too lazy to mention that extra "point three".

So.....what about 3.2999V? It's not 3.3V, so is it HIGH? Is it LOW? Or what about 0.1V? It's not 0V, so is it LOW? Is it HIGH? Real world signals will have some noise and thus will behave this way. Therefore, to deal with this, a range of values is setup to represent HIGH and LOW. For 3.3V level logic, it looks like this:
So 3.2999V would be seen as HIGH and 0.1V would be seen as LOW. Everything that is designed to work with 3.3V logic is expected to work within these limits. If it doesn't, it is considered broken. Nothing should be generating values in the "gray zone".

Pocket Ref

For this guide, it's good enough to just keep this decoder ring in mind:

- 0V = LOW = FALSE = 0
- 3.3V = HIGH = TRUE = 1
Circuit Hookup

Alright, let's go hands on.

The hookup for this guide is very simple. While a button could be used, we will simply touch the alligator clips together to simulate this same function. We'll use a resistor a little later, but for now just connect the alligator clips as shown.

This is the overall view of the hookup.

- RED to 3.3V
- BLUE to #3
- BLACK to GND

Simply connect the alligator clips to the gold pads on the edge of the Circuit Playground as shown.

READ THE FOLLOWING TO PREVENT POTENTIAL DAMAGE.

Be careful to not let the BLACK and RED alligator clips touch each other directly. We will be touching other alligator clips together, but never these two.
Touching RED to BLUE is OK.

Touching BLACK to BLUE is OK.
Touching **BLACK** to **RED** is NOT OK.

This will short power to ground and the Circuit Playground will not be happy.
Hello Digital

Alright, let's start super simple. To read the digital value on the #3 pad, we will use the Arduino library function `digitalRead` (https://adafruit.it/uBk). You can use the following sketch to read and print the value.

```cpp
#include <Adafruit_CircuitPlayground.h>

void setup() {
  Serial.begin(9600);
  CircuitPlayground.begin();
  pinMode(3, INPUT);
}

void loop() {
  int reading = digitalRead(3);
  Serial.println(reading);
  delay(500);
}
```

With this code loaded and running on the Circuit Playground, open the Serial Monitor.

Tools -> Serial Monitor

Now touch the BLACK clip to the BLUE clip.

This sets the voltage of the digital input to 0V (LOW). You should see 0s scroll by in the Serial Monitor.
Now touch the RED clip to the BLUE clip.

This sets the voltage of the digital input to 3.3V (HIGH). You should see 1s scroll by in the Serial Monitor.

Pretty straight forward. When the digital input pin is connected to 0V (GND) it is LOW and prints out 0s. When the digital input pin is connected to 3.3V it is HIGH and prints out 1s.

And there are our two values, 0 (LOW) and 1 (HIGH). So we're done, right? Of course not. Just getting started.
What about the case where neither the BLACK nor the RED are touching the BLUE clip? Is that LOW? Is that HIGH? Truth is, I don't know. And no one does, not even the Circuit Playground. The poor little BLUE alligator clip is floating in space not tied to any known value. This is called a "floating input" and we'll look at how to deal with this next.
Floating Inputs

Here's what our alligator clip setup looks like when nothing is connected to the **BLUE** clip.

You might think that the voltage would be 0V and the digital pin would read **LOW**. After all, it's not attached to anything. It's attached to nothing. And 0 is like nothing. However, voltages don't exist at a point - they exist between two points. And in the case of the floating input, one of those points is not fixed to any known value. Therefore, the voltage is also unknown.

The end result is that the actual value is unpredictable, which isn't very useful. You can try using the sketch from the previous section and watch for the value to change. However, the following sketch will monitor the pin constantly (very fast) and detect when it changes.
With this code loaded and running on the Circuit Playground, open the Serial Monitor.

**Tools -> Serial Monitor**

Now pick up the **BLUE** alligator clip and try touching it with your finger. You may need to try touching it several times, but hopefully at some point it will see a change and report it as shown below. Note that your initial value might be 1 - it totally depends on what ever randomness is happening in your test environment.
The fix for this is pretty straightforward. You just set things up so that the digital input, the **BLUE** alligator clip, is always attached to a known value, in this case either the **RED** (3.3V) alligator clip or the **BLACK** (0V) alligator clip. This is called "pulling" the input, as you are forcing it (pulling it) to a specific value. When the input is pulled to 3.3V it is called "pulling up". When the input is pulled to 0V it is called "pulling down". You can go either way.

But wait! Remember we can't touch the **RED** and **BLACK** wires directly to each other. So if one of those is already attached to the **BLUE** clip, how do we ever touch the other one? The answer is to use a resistor - and guess what it's called? Well, a pull up resistor or a pull down resistor, as the case may be.

Let's play with both.
Pull It Up

Let's start with a pull up resistor. To do so, we will take the 10k ohm resistor and attach it as shown below.

We can just use the alligator clips to hook this up.

Attach the 10k ohm resistor between the BLUE and RED alligator clips.

Now run the sketch from the previous section again. Try touching the BLUE alligator clip with your hand as you did before. Now it should never see any changes. It should be nicely held at 3.3V.
Now try touching the **BLACK** clip to the **BLUE** clip.

Now it should see the change and report it.

So we've fixed the floating input problem. The resistor "pulls" the input "up" to 3.3V and the pin reads **HIGH** (1). When we touch the **BLACK** clip, the pin sees 0V and reads **LOW** (0).

Additionally, the resistor prevents a short between power and ground. That's why it's important to touch the **BLACK** only to the **BLUE**. Even with this resistor in the circuit, it is still bad to touch **BLACK** to **RED**. **DON'T DO IT!!**

We can do the same thing in the other direction. That is, we can "pull down". Let's look at that next.
Pull It Down

This is the same idea as the pull up resistor, except now it is attached to GND. The hook up is shown below.

Again, we use the alligator clips to set this up.

Attach the 10k ohm resistor between the BLACK and BLUE alligator clips.

Run the sketch again and touch the BLUE clip with your finger. As before, it should not detect any changes. It is being held LOW and is happy there.
Now try touching the RED clip to the BLUE clip.

It should see the change and report it.

Note that the only thing different between this and the pull up configuration is the order of the values.

- A pull up is normally **HIGH** and goes **LOW** when the alligator clips are touched together.
- A pull down is normally **LOW** and goes **HIGH** when the alligator clips are touched together.

And remember that our touching of the alligator clips is like pressing a button. So the same logic applies for button presses as well.
Which Way?

So you've got two options for dealing with floating inputs. You can either pull it up or pull it down. But which one should you use? It's a good question but beyond the scope of the guide. It's one of those "depends" kind of things.

For now, it is enough to simply understand what a floating input is, why it is an issue, and how they can be dealt with.
Bouncing Inputs

Bumbles (https://adafruit.it/uBl) aren't the only thing that bounce. So do push buttons. When you press your basic, normally open, momentary style button, it closes its little internal switch connection. In an ideal world we could hook that up to our Circuit Playground, like we have the alligator clips, and get an input signal that would look like this:

![Ideal button signal diagram](image)

That's the way a button should work and how most people think they work. However, in reality it looks more like this:

![Realistic button signal diagram](image)

It goes up and down several times before finally stabilizing. Take a button apart and check out its guts and see how it works. Inside you'll find some form of mechanism that brings two contacts together - much like we are doing with the alligator clips. However, the contact doesn't happen instantly. It bounces around a bit, going closed, open, closed, open, closed, etc. This is purely a mechanical phenomenon, kind of like dropping a ball - it bounces before finally sitting still on the ground.

It does this over the period of time = t. While this period of time is very fast, and seems almost instantaneous to you and me, to the Circuit Playground, it's a huge period of time. The Circuit Playground can execute many many instructions in that period of time.

We can use our alligator clip setup to gain an appreciation for this. We will use the "pull down" setup from the previous section, so make sure you've got the alligator clips hooked up that way.

Now run the following sketch.
With this sketch loaded and running on the Circuit Playground, open the Serial Monitor.

Tools -> Serial Monitor

Now play around with touching the alligator clips together.
Touch the **RED** to the **BLUE** alligator clip.

Did you get more than one print out when you made the connection? Try touching the clips several times and watch the behavior of the messages in the Serial Monitor. If you do it just right, you can get only one message to print. However, more often you get multiple lines. Same when the connection is removed - the bounce happens in both directions.

Touching the alligator clips together like this is a very bouncy operation. The Circuit Playground is fast enough to see all those bounces and prints out a message each time.

Here’s what I get trying this after a fair amount of coffee to add some extra bounce.

So how do we fix this? There are many ways, both hardware based and software based. So many in fact that a discussion of debouncing techniques is worthy of its own guide (or more).

However, we will demonstrate one very simple software approach. The trick is to just add a delay after the initial
detection. This forces the Circuit Playground to go to sleep during the bouncing and not pay any attention to it.

To test this, simply uncomment (remove the //) the following line in the previous sketch. (it's near the bottom)

```
delay(100);
```

Run this new code and play around again with touching the alligator clips together. Now you should get a single print out when touched and separated as shown below.

![Printout example](image)

The main drawback to this technique is that the Circuit Playground can not do anything else during the delay() period. For more complicated programs the processor is always doing something and you try to avoid forcing is to go to sleep using something like delay().

However, for simple programs, this technique works just fine. In fact, you'll see it used quite often. It's simple and works. Ever seen anything like this at the top of a sketch?

```
#define DEBOUNCE    100
```

Look elsewhere in the code and you'll likely find:

```
delay(DEBOUNCE)
```

It's using the same technique and simply defining the delay period globally.
Under The Hood

The Circuit Playground has two push buttons (https://adafru.it/uBm). They are simple momentary buttons that effectively do the same thing as pressing the alligator clips together. But how are they integrated into the Circuit Playground? Do they deal with any of the issues we've talked about? Let's take a look.

Take a look at the schematic (https://adafru.it/uBn) for the Circuit Playground. The circuit diagram for the buttons can be found in the upper left.

Let's just look at one button since they are both used in the same way. Here's a blow up of the part of the circuit diagram that matters. The thing above the label EVQQ is the actual button.
We don’t have a button in our alligator clip setup, so let’s remove it. Now we are left with this.

It may not look like it, but this is identical to the "pull down" version of our alligator clip setup. Here’s the schematic showing where the various alligator clips are.
So the Circuit Playground deals with the floating input issue by using a pull down resistor. The same thing is done for both buttons. Righty-o.

Now, what about dealing with switch bounce? Any magic going on there? Well, let's take a look at the code for `CircuitPlayground.leftButton()` and see what's going on. Here it is (direct link to repo [https://adafru.it/uet](https://adafru.it/uet)):

```c
boolean Adafruit_CircuitPlayground::leftButton(void) {
  return digitalRead(CPLAY_LEFTBUTTON);
}
```

Hmmmm. It just returns the value from `digitalRead()` for the input the button is attached to. Same as we've been doing with our alligator clips. That's it. Nothing else.

So, there is no debouncing being done by the Circuit Playground library. It is left up to the user (you) to incorporate this in their program somehow.
Questions and Code Challenges
The following are some questions related to this project along with some suggested code challenges. The idea is to provoke thought, test your understanding, and get you coding!

While the sketches provided in this guide work, there is room for improvement and additional features. Have fun playing with the provided code to see what you can do with it. Or do something new and exciting!

Questions

- The microcontroller on the Circuit Playground has internal pull up resistors. Why were external pull downs used instead? (hint: it's kind of subjective)
- Why was a value of 10k ohms used for the pull up/down resistor?

Code Challenges

- Reduce the amount of delay in the bounce sketch and see how it affects the debounce behavior. Try making it as low as 1, i.e. delay(1).
- Modify the bounce sketch to read the input from one of the Circuit Playground push buttons to examine its bounce behavior. (remove the delay)