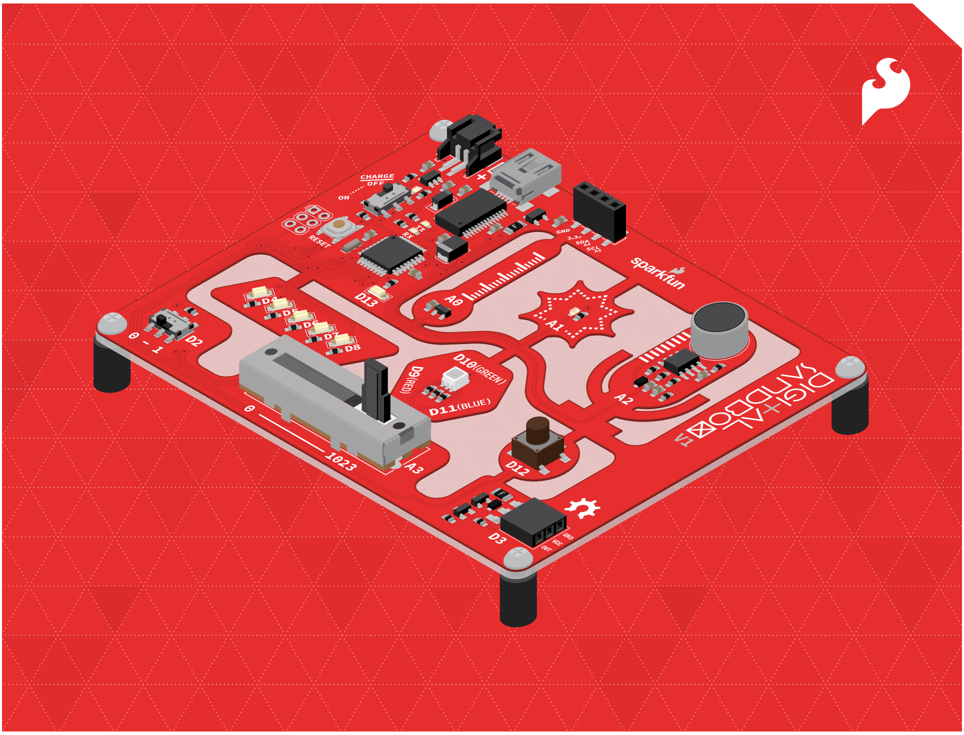
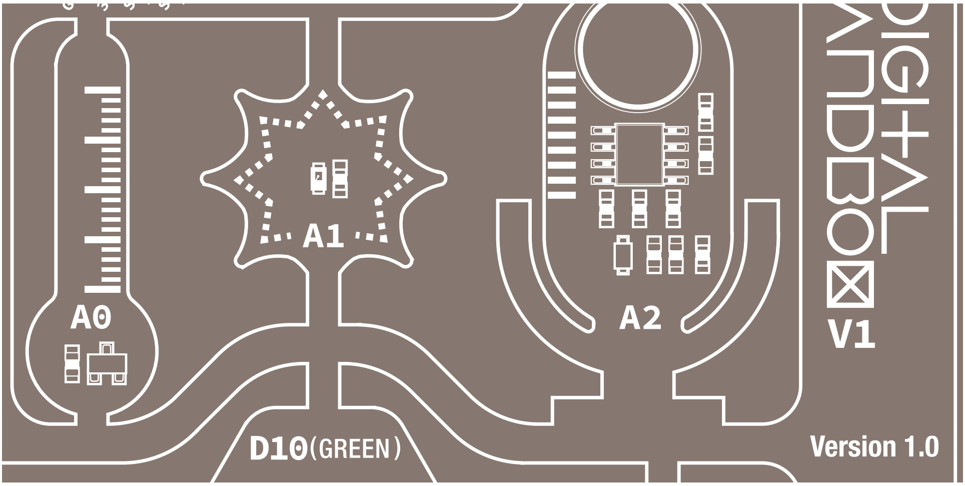
Chipsmall
深圳市津利帝科技有限公司
Geelead Technology Limited
Chipsmall
采购部



DS GUIDE

Your guide to the SparkFun Digital Sandbox



Version 1.0

Welcome to the Digital Sandbox!

When partnered with the Digital Sandbox, this guide helps to introduce the fundamental concepts of programming and electronics. Using ArduBlock – a simple, graphical version of the popular Arduino programming language – you will program 13 experiments that progressively explore subjects like digital inputs, analog outputs, serial communication and more. The experiments are project-based, and are built to inspire inventions such as reaction-testing games, automatic night lights, adjustable volume meters and more.

2 .... The Anatomy of the Digital Sandbox
4 .... Digital Sandbox Baseplate Setup
5 .... Setting up Arduino and ArduBlock
7 .... Experiment 0: Setup, Loop, and Blink
11 .... Experiment 1: Exploring Blink
15 .... Experiment 2: Multi-Blink
19 .... Experiment 3: Dimming (the Hard Way)
23 .... Experiment 4: Dimming (the Easy Way)
27 .... Experiment 5: Color Mixing
31 .... Experiment 6: Number Storage with Variables
35 .... Experiment 7: If This, Then That
39 .... Experiment 8: The Reaction Tester
43 .... Experiment 9: Serial Calculator
47 .... Experiment 10: Do the Analog Slide
51 .... Experiment 11: Automatic Night Light
55 .... Experiment 12: Thermal Alert!
59 .... Experiment 13: Sound Detecting

Bonus add-on experiments (parts not included):

63 .... Experiment 14: Opto-Theremin
69 .... Experiment 15: Serial Motoring
75 .... Experiment 16: Servo Sweeper

The Anatomy of the Digital Sandbox

1 USB Mini-B Connector

Used to connect to a computer

2 JST Right-Angle Connector

Used to supply power to the board

3 Slide Switch for Charging

Used to charge a lithium polymer battery that is plugged into the two-pin JST connector, while the Digital Sandbox is connected to a computer and the slide switch is in the "ON" position

4 Reset Button

This is a way to manually reset your Digital Sandbox, which will restart your code from the beginning.

5 Slide Switch (Pin D2)

On or off slide switch.

6 LEDs (Pins D4-D8)

Use one or all of the LEDs (light-emitting diodes) to light up your project!

7 LED (Pin 13)

Incorporate this into your sketch to show whether your program is running properly.

8 Temperature Sensor (Pin A0)

Measures ambient temperature

9 Light Sensor (Pin A1)

Measures the amount of light hitting the sensor

10 RGB LED (Pins D9-D11)

RGB (red-green-blue) LEDs have three different color-emitting diodes that can be combined to create many colors.

11 Slide Potentiometer (Pin A3)

Change the values by sliding it back and forth.

12 Microphone (Pin A2)

Measures how loud something is

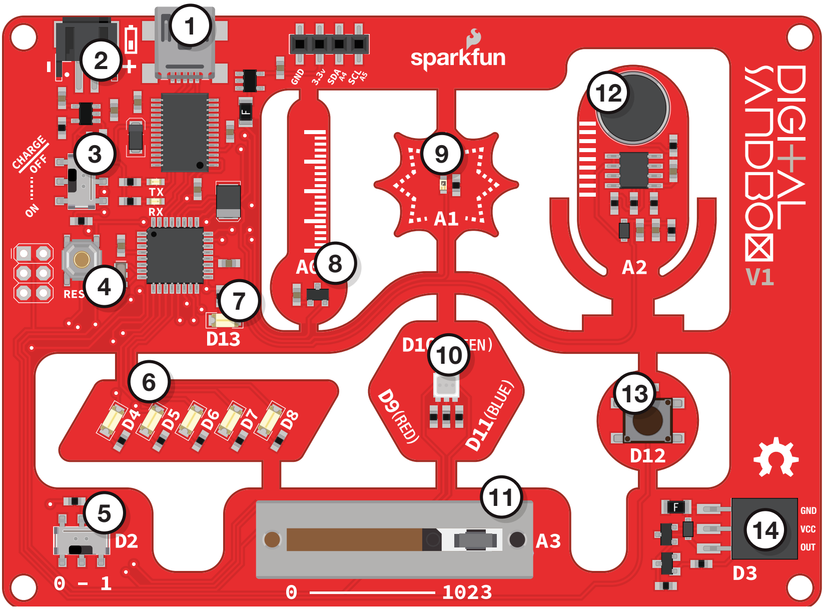
13 Push Button (Pin D12)

A button is a digital input. It can be either “on” or “off.”

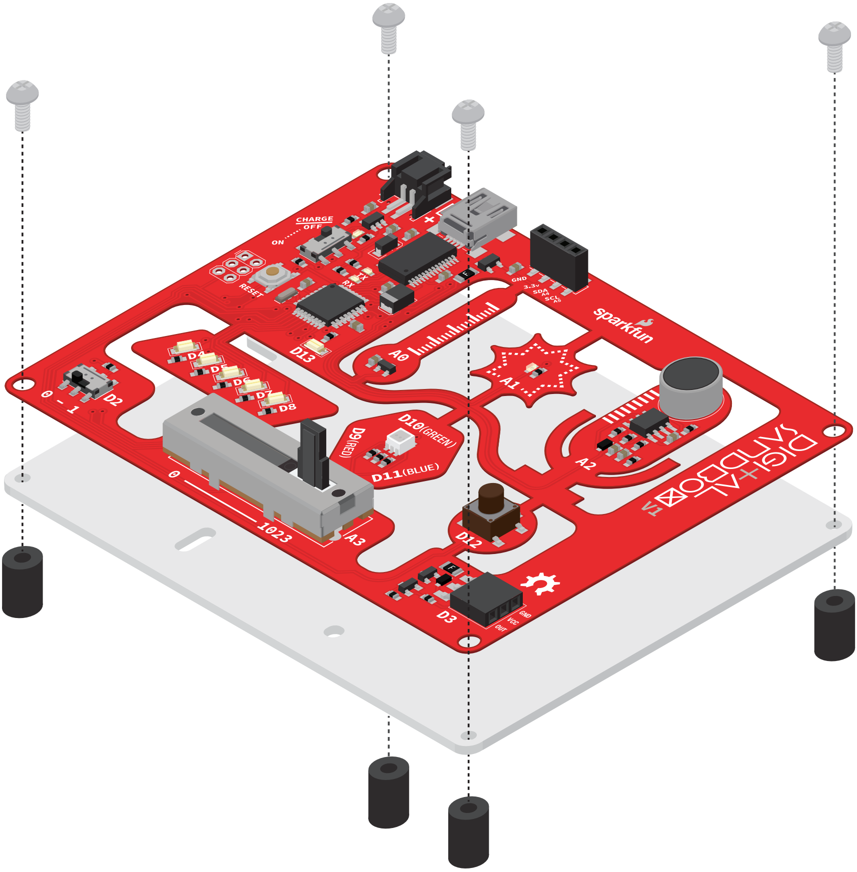
14 Add-on Header (Pin D3)

Three-pin header for add-ons. Example add-ons are servos, motors and buzzers.

The Anatomy of the Digital Sandbox



Digital Sandbox Baseplate Setup



Secure the Digital Sandbox board to the baseplate

The Digital Sandbox board can be attached with the included Phillips-head screws for easy removal later.

Download the Arduino/ArduBlock Combo

Arduino and ArduBlock are available for all popular operating systems. To download Arduino and ArduBlock, please go to:

www.sparkfun.com/digitalsandbox

Make sure you grab the version that matches your operating system.

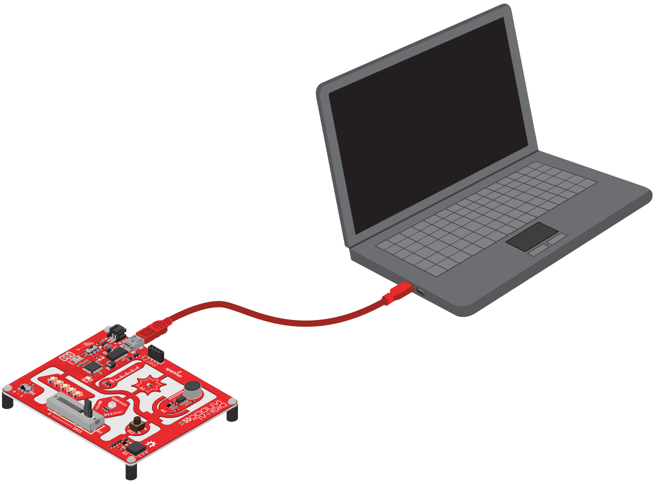
The Arduino software comes packaged in an archived **.ZIP** format. Once you've downloaded the **ZIP** file, you'll need to **extract it**.

Mac users: Move the **Arduino** application into the **Applications** folder. Move the **Digital Sandbox Examples** folder to your preferred location.

Windows users: Move the *Arduino* folder to your preferred location.

Install Drivers

Once you have downloaded and extracted the Arduino software, connect the **Digital Sandbox to your computer**.



Once the board is connected, you will need to install drivers. For instructions specific to your operating system, please go to:

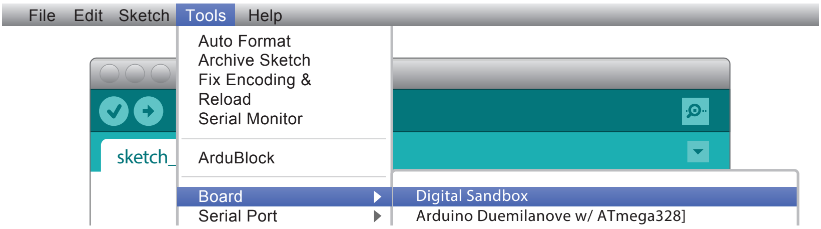
www.sparkfun.com/ftdi

Open Arduino and ArduBlock

ArduBlock is an add-on that exists inside the Arduino software. To open it, first open the Arduino IDE. Windows users should run *Arduino.exe*; Mac users can run the *Arduino* application.

Setting up Arduino and ArduBlock

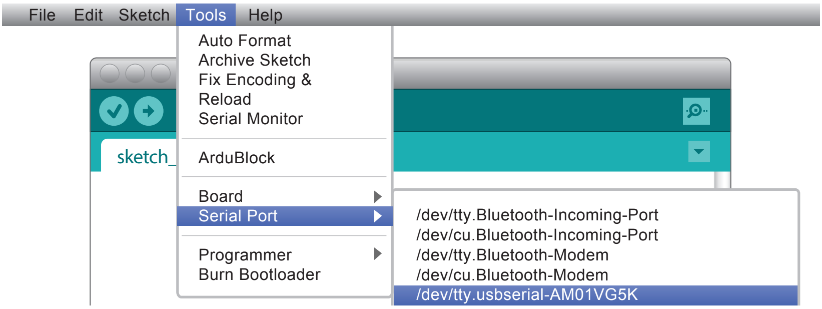
Let’s do some preparation before opening ArduBlock. First, go to the **Tools** menu, hover over **Board** and select **Digital Sandbox**.



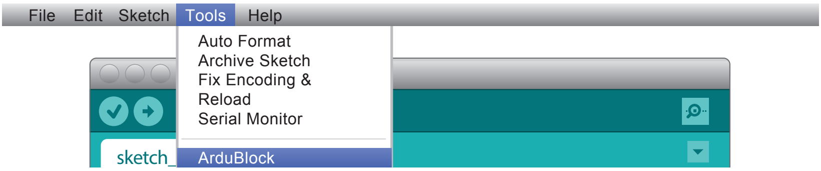
Next, go back to the **Tools** menu, hover over **Serial Port** and select the serial port number that matches your Sandbox board.

Window users: This is likely to be COM3 or higher (COM1 and COM2 are usually reserved for hardware serial ports). To find out, you can disconnect your Sandbox and re-open the menu; the entry that disappears should be the Sandbox. Reconnect the board and select that serial port.

Mac users: On the Mac, this should be something with /dev/tty.usbmodem or /dev/tty.usbserial in it.



Finally, to open ArduBlock, go to **Tools** and select **ArduBlock**.



What opens next is the ArduBlock interface. Make sure the Arduino window remains running in the background. If you close that, ArduBlock will close as well.

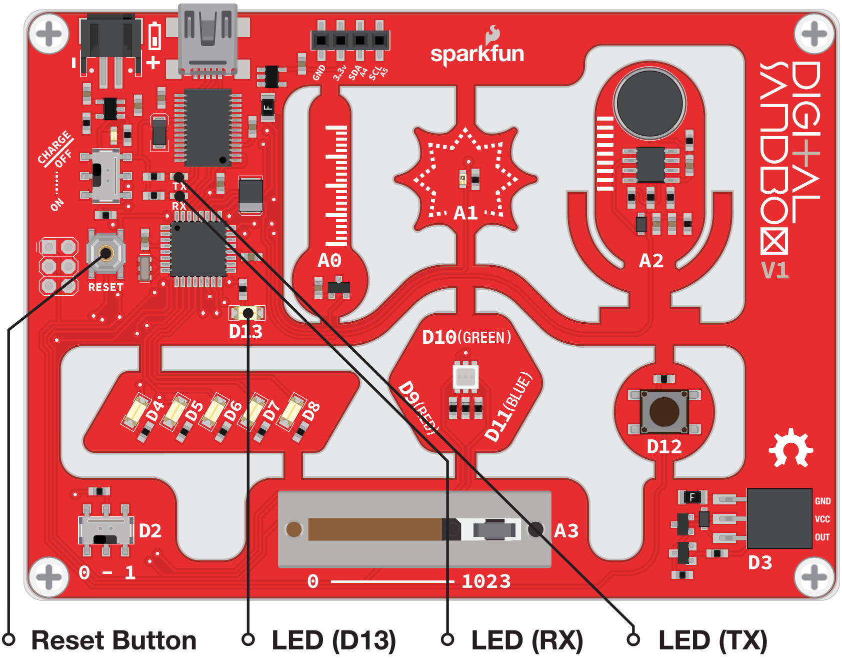
Note: If you don’t see *ArduBlock* under the *Tools* menu, you may need to manually install it. For help adding ArduBlock to a previous Arduino installation, please visit:

www.sparkfun.com/ardublock

0: SETUP, LOOP, AND BLINK

When faced with a new platform, a coder’s first task is to write a “Hello, world” program. Usually a “Hello, world” program actually displays those comforting words on a screen. The Digital Sandbox doesn’t give us a screen to display words on, but we do have LEDs: wonderful, blinky, bright and shiny LEDs. Instead of words, let’s blink an LED to say, “Hello, world.”

Active Parts



Background Information

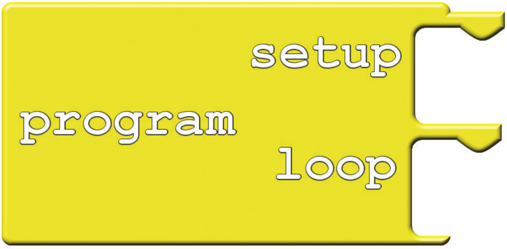
This experiment introduces the general concept of **physical** programming. Changes you make in your program will actually affect what's happening **on** the Digital Sandbox board.

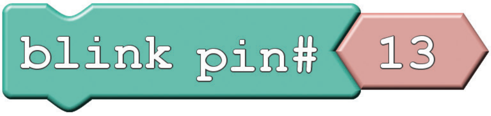
This drawing also serves to introduce a couple of the most fundamental Arduino programming concepts: *setup* and *loop*.

Please note: You can continue with the **Code Components** section to see what blocks you will need for this experiment. Each experiment also has its own saved file with all the experiment's blocks already set up, so you can get started faster or troubleshoot if needed. To open this experiment, click the **Open button** in ArduBlock. Then, go to the **Digital Sandbox Examples** folder in the same location where you moved the folder during the initial setup. From there, you will open the corresponding experiment file. For example, **00_setup_loop_blink.abp** is for Experiment 00.

Code Components

This code drawing requires **two blocks** (well, sort of three):



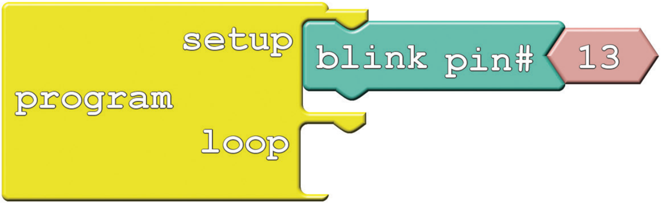


- **Program**: This block is **required** for every single ArduBlock drawing! You can only have one per drawing. *Program* always has two slots for blocks – one named "setup" and another named "loop." Find this block under the **Control** bin.

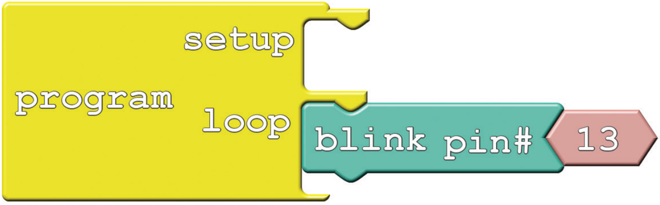
- **Blink**: Find this block under the **Pins** bin. This block “blinks” a pin on the Digital Sandbox. The *Blink* block actually gives you *two* blocks for the price of one! It also includes a pink block with the number 13 inside it. Leave that block alone for now; we’ll discover its use in later experiments.

Do This

With this pair of blocks, there are only two functional drawings that we can create. You can either stick the *Blink* block under the *setup* section of *Program*, or under the *loop* section.

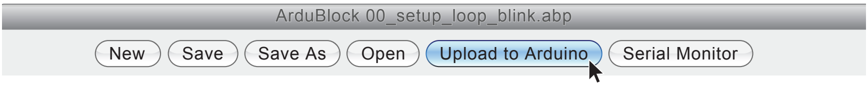


OR



Experiment 0: Setup, Loop, and Blink

Try sticking the *Blink* block **under** *setup*, then **click the Upload to Arduino button**.



Keep your eyes glued to the Digital Sandbox as the code uploads. You'll see the red and green RX and TX LEDs blink like crazy as code is sent from your computer to the Digital Sandbox. Pay extra close attention to what happens after the red and green LEDs do their dance. Do you notice anything?

Now **move** the *Blink* block from *setup* to *loop*, and do the **Upload to Arduino** jig again. Notice anything different?

Every Arduino program requires that two functions **always** be present: *setup* and *loop*. From the names of these functions, it should be pretty clear what their job is.

Setup runs once, at the very beginning of the program. Its purpose is usually to set the platform up for the rest of its lifecycle (until the Sandbox is reset, or loses power). As we continue on with these experiments, you'll have a greater understanding of what kinds of things need to be set up in advance.

If *setup* sets the Sandbox up, *loop* must…**loop**. Code in this block will execute sequentially and endlessly. Once we get to the bottom of *loop*, we jump right back up to the top and do it all over again. This looping will continue until you either reset or remove power.

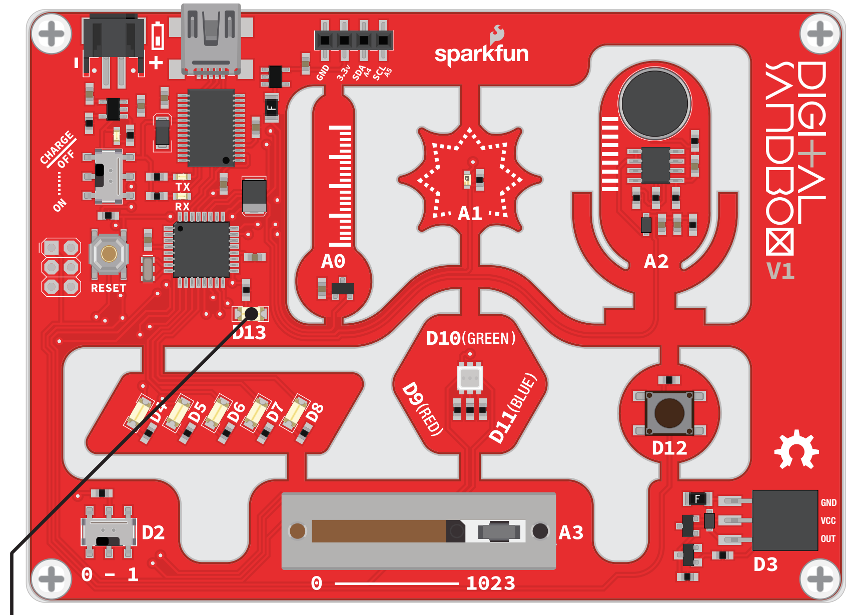
Further Explorations

- What happens when you press the **reset button**?
- What happens if there's nothing in either the *setup* or *loop* (move the *Blink* block out)?
- What happens if you add a **second** *Blink* block to the drawing? Regardless of where you put it, can you discern which of your *Blink* blocks are being executed?
- What do you think the **13** inside the *Blink* block is for?

1: EXPLORING BLINK

Now, we didn't exactly cheat in experiment 0, but the Blink block was a bit of a shortcut. What if you wanted to speed up the blinking? Or change how long it's on, versus how long it's off? Or even blink something other than that wimpy, little red LED?

Active Part



LED (D13)

Background Information

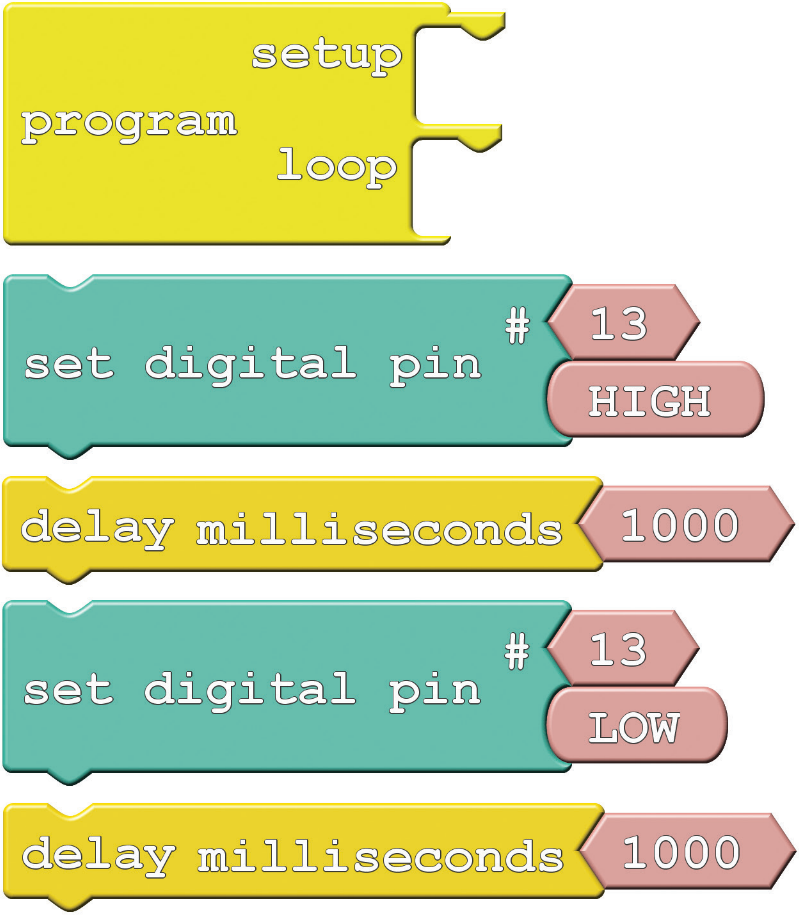
This experiment digs into the anatomy of the *Blink* block. We can customize an LED blink with a combination of two blocks – *Set Digital Pin* and *Delay Milliseconds*.

This experiment introduces the concept of **digital output**. We call the Sandbox's LEDs "outputs" because it's an effect that the board produces.

The term "digital" means that the output can only take **one of two states**: ON or OFF. We may also refer to those two opposing states as HIGH/LOW or 1/0. When an output connected to an LED is HIGH, the LED turns on. When the output is LOW, the LED turns off.

Code Components

Here is the set of blocks we'll use to create this drawing:



```
program  setup
         loop
set digital pin # 13
                  HIGH
delay milliseconds 1000
set digital pin # 13
                  LOW
delay milliseconds 1000
```

Aside from *Program* block, which you should include in every drawing, there are two new blocks to be added:

- **Set Digital Pin**: This block **sets an output** to either HIGH or LOW, so it can be used to turn the LED on or off. Find this block under the **Pins** bin. When you drag this block over, it includes a pair of pink blocks containing "13" and "HIGH." Let's only concern ourselves with the bottom pink block for now.

- **HIGH/LOW block**: If you mouse over this block, a drop-down arrow will appear. Click the arrow and you can change the value of the block to either "HIGH" or "LOW." This determines which of the two states you're setting the digital output to.

- **Delay Milliseconds**: The Digital Sandbox runs code so fast that sometimes we need to slow it down with a delay. This block will halt the Sandbox from doing anything for a specified number of milliseconds. There are 1000 milliseconds (ms) in a second, so delaying for 1000ms will stop for 1 second. Find this block under the **Control** bin.

Do This

Organize and snap together the *Set Digital Pin* and *Delay Milliseconds* blocks so they alternate – teal, yellow, teal, yellow. Then put the group of four blocks in the *loop* section of the *Program* block. Then, **upload** the code.

You should see a very familiar sight. But this time you have control over the blink rate! Adjust the value in the *Delay Milliseconds* block(s). What happens if you make the delays shorter? What happens if the two delays are not for the same amount of time?

The Digital Sandbox operates at 8MHz – there's a clock in there that ticks eight million times per second. That means it can run millions of lines of code per second. Without any delays in the program, the digital output would flick on and off so fast you wouldn't be able to tell if it's actually on or off.

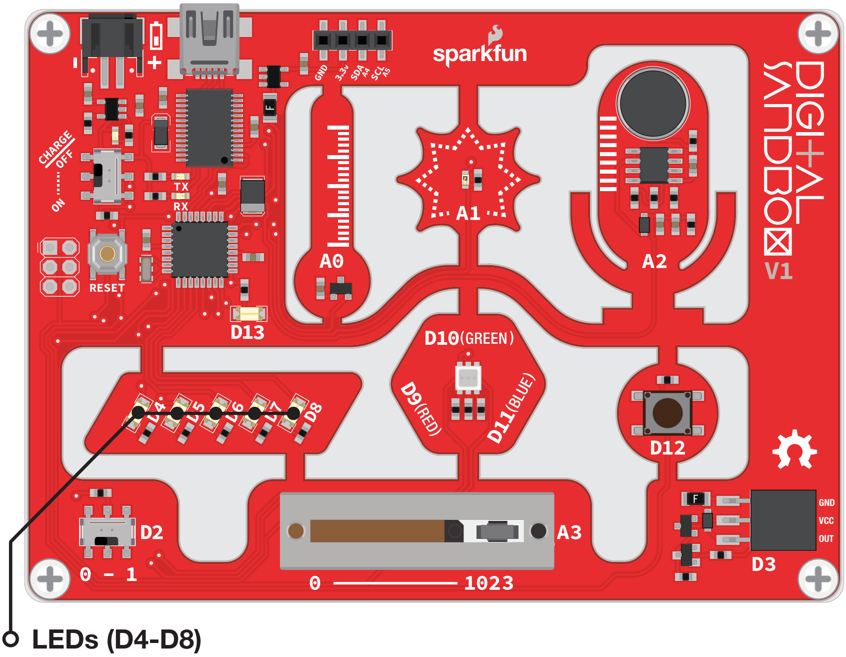
Further Explorations

- How short can you make the delays and still notice a blink (10ms? 1ms?)?
- What happens if you take the *Delay Milliseconds* block out of the program?
- While digging around for the *Delay Milliseconds* block, you may have discovered a *Delay Microseconds* block as well. What happens if you swap that in?

2: MULTI-BLINK

Large arrays of LEDs are often used to create massive outdoor signs and animations, because they're both bright and efficient. While we don't have the millions of LED pixels that a display in Times Square might have, we can still create some fun patterns with the Digital Sandbox.

Active Parts



LEDs (D4-D8)

Background Information

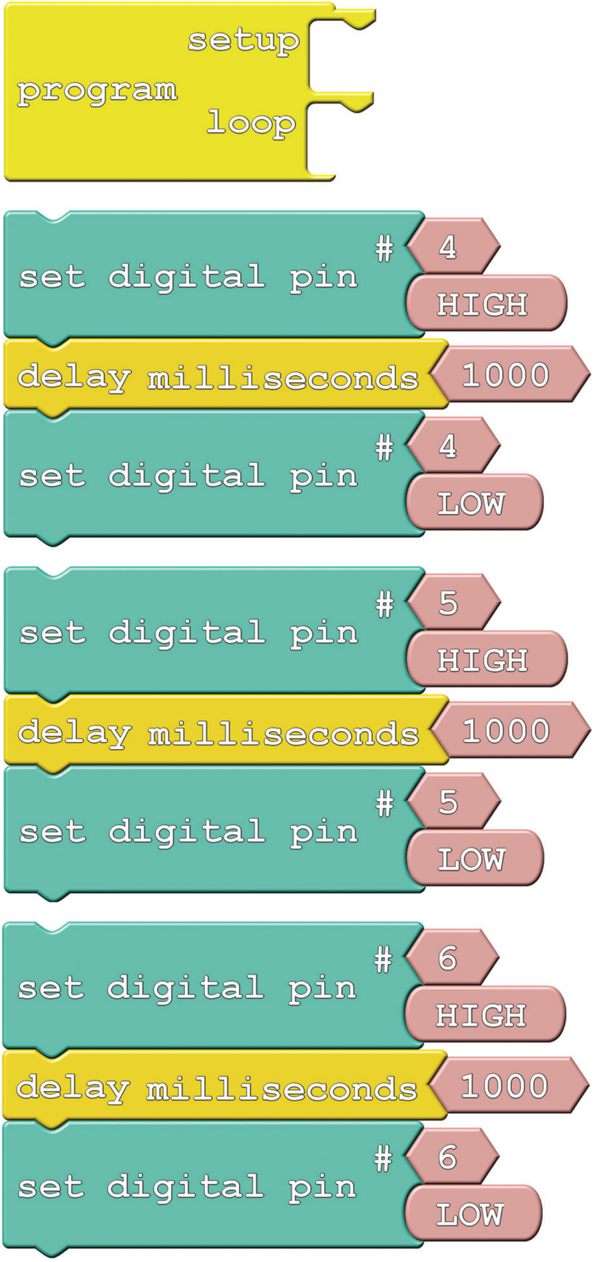
In this experiment we explore the subject of **pins** - the manipulators of the Sandbox. Each LED (as well as the other inputs and outputs on the Digital Sandbox) is connected to a specific pin on the Sandbox's microcontroller.

Pins are all **uniquely numbered**, and each input or output component on the Sandbox is labeled with the pin number it's connected to - that's the *D2, D4, D11, A1,* etc. lettering next to each LED, switch and sensor.

Every pin can be separately controlled; for instance pin 4 can be set HIGH at the same time pin 5 is set LOW. Some pins (as we'll later discover) have special powers, but every pin is at least able to accomplish digital input and output.

Code Components

Whoa! Block explosion! This experiment calls for sixteen total blocks:

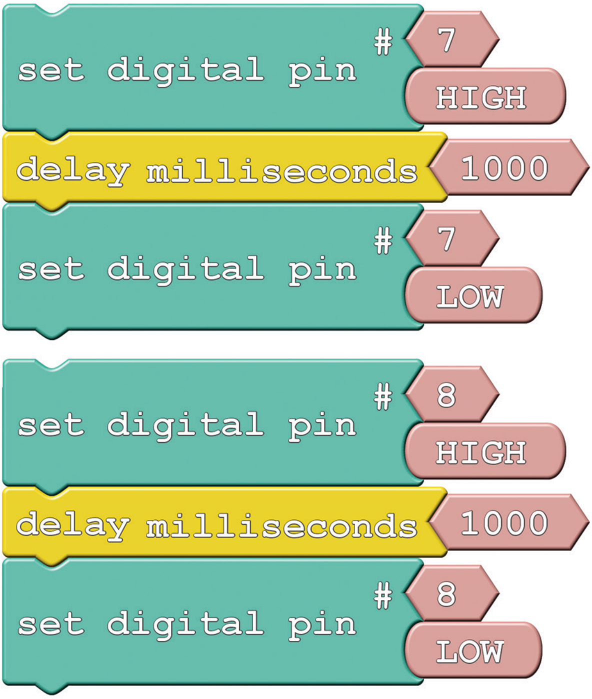


```
program  setup
         loop

set digital pin # 4  HIGH
delay milliseconds 1000
set digital pin # 4  LOW

set digital pin # 5  HIGH
delay milliseconds 1000
set digital pin # 5  LOW

set digital pin # 6  HIGH
delay milliseconds 1000
set digital pin # 6  LOW
```



```
set digital pin # 7 HIGH
delay milliseconds 1000
set digital pin # 7 LOW

set digital pin # 8 HIGH
delay milliseconds 1000
set digital pin # 8 LOW
```

Instead of introducing a new block, we'll be adjusting the value of *Set Digital Pin*'s top pink *Pin Number* block. This value specifies which of the Sandbox's pins we'll be toggling.

Do This

In our unfinished example, the blocks are all arranged in groups of three. Each group begins by setting a pin HIGH, then delays for a second and sets it back to LOW. Notice that each group of three toggles a different pin, ranging from pin 4 to pin 8. **Stack the groups of three** on top of each other in the *loop*, then **upload** and enjoy the exciting animation.

If the LED slide is too slow for you, try adjusting the delays to make it faster. Or perhaps you want to change the pins to adjust the order of the blinks?

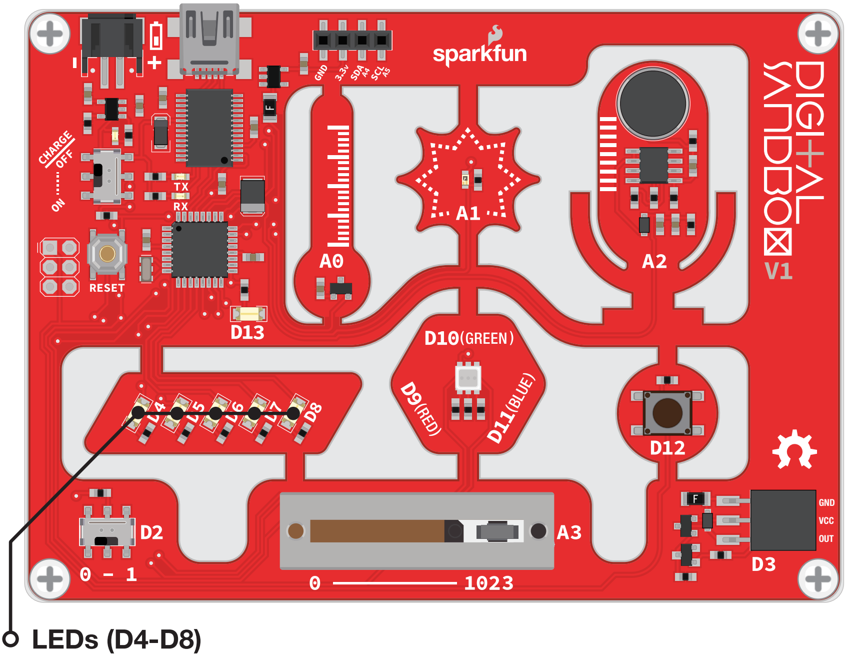
Further Explorations

- Try adding more blocks to create slicker patterns. Can you make a Larson scanner (ask an old person about Cylons or Knight Rider)? A chaser? Flip from odds to evens?
- Try turning on more than one LED at a time. Turn them all on (and shield your eyes)!

3: DIMMING (THE HARD WAY)

Yikes! Those white LEDs are blindingly bright! Is there any way to dim them? Unless one of your hobbies is staring into the sun, we recommend putting a piece of paper over the LEDs in this experiment…or wear sunglasses.

Active Parts



LEDs (D4-D8)

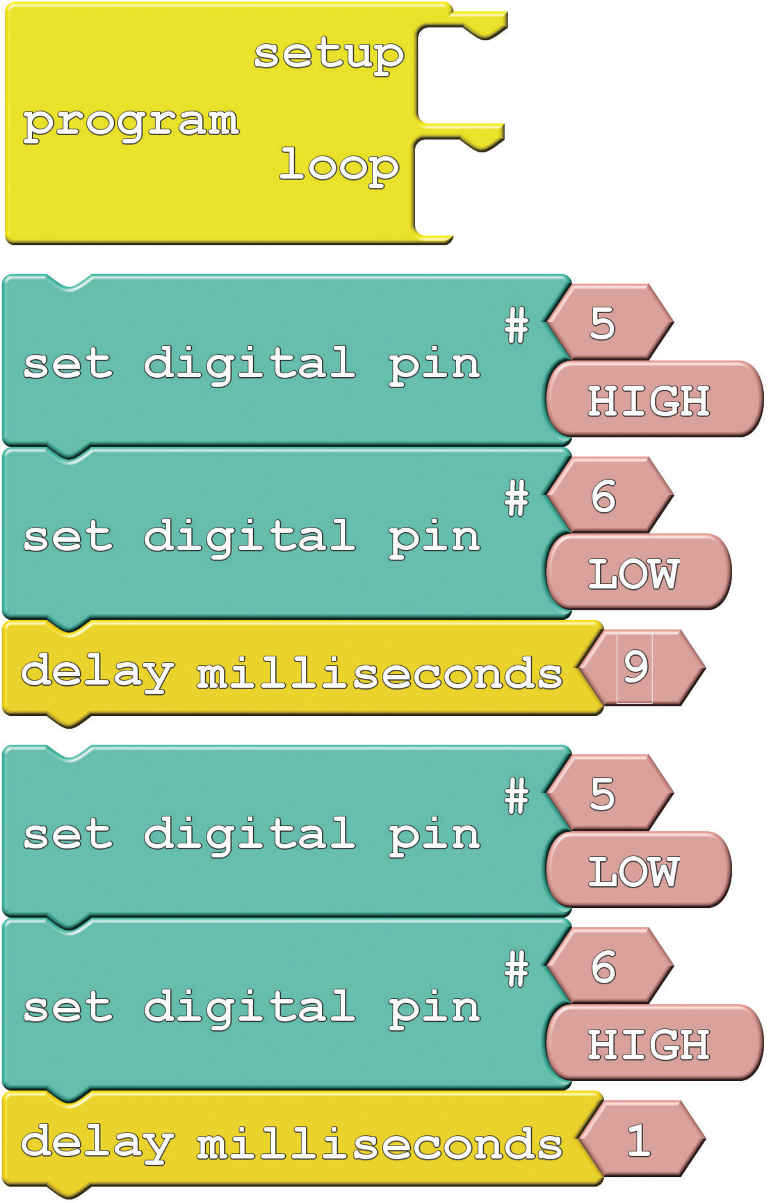
Background Information

Remember that the Digital Sandbox is **fast**. It can flick an LED on and off millions of times per second. What if we blinked the LED super fast, but also made it so the time it's off was more than the time it was on? This is called **pulse-width modulation (PWM)**, a tool with a variety of applications, including the **dimming** of LEDs.

In this experiment we'll explore PWM the hard way, by coding it in manually.

Code Components

We’ll use a similar set of blocks:



```
program  setup
         loop

set digital pin # 5  HIGH
set digital pin # 6  LOW
delay milliseconds 9

set digital pin # 5  LOW
set digital pin # 6  HIGH
delay milliseconds 1
```

Take note of how long each delay is, and which pins are on/off in each group.

Do This

Stack the two groups of three on top of each other in the *loop* section, and **upload**.

After uploading, take a close look at the LEDs connected to pins 5 and 6. Can you spot a difference between the two? The D6 LED should look dimmer in comparison to D5. That's because D6 is set to be low 90% of the time, and on only 10%. It's blinking on and off so fast that you can't notice, and the blinking is creating a dimming effect.

What happens if you swap the two *Delay Milliseconds* blocks? What if you change the values in each of the delay blocks (try to keep the sum of delay times to around 10ms)?

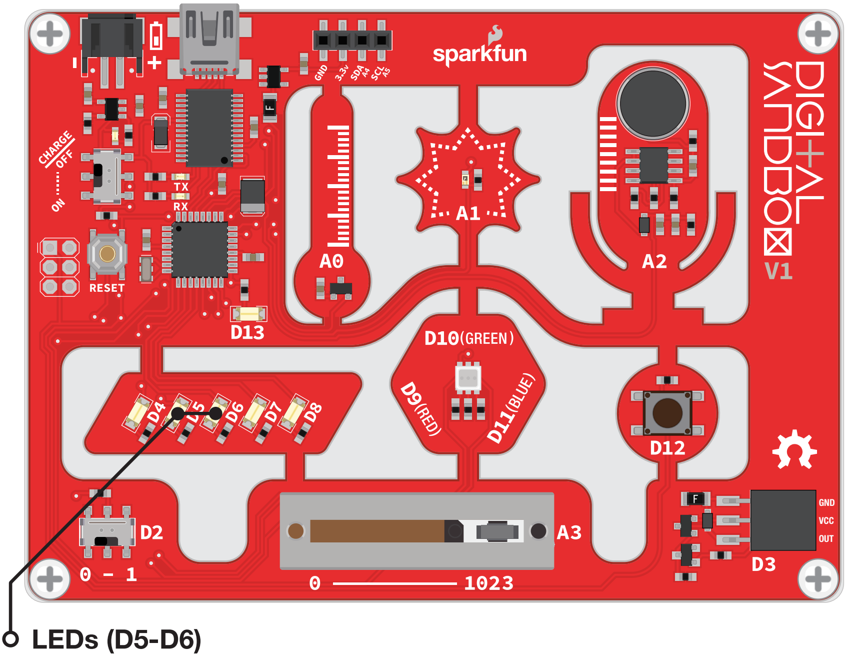
Further Explorations

- How long can you make the delays before you start noticing a blink?
- Try comparing both LEDs to a fully-on LED. Add a *Set Digital Pin* block to the *setup*, and have it turn the D4 LED HIGH. Can you tell a difference between D4, D5 and D6?
- What happens if you add something else to the *loop* section, like your animation from Experiment 2?

4: DIMMING (THE EASY WAY)

Manual PWM is hard, and it doesn’t leave room for anything else in the program. Why can’t we offload that chore to the Digital Sandbox’s microcontroller? It’s smart enough for that…right?

Active Parts



LEDs (D5-D6)

Background Information

PWM is such a popular tool many microcontrollers implement special hardware so they can mindlessly toggle the pin while doing something else. We call this PWM-based output "**analog output**."

Unlike *digital* outputs, which only have two possible values, analog outputs have a huge range of possible values. On the Sandbox, we can analog-ly output **256 different values**. If we set an analog output to 0, that's like setting a pin LOW, and 255 is like setting a pin HIGH, but all of the values in between produce an output that's neither HIGH or LOW – it's somewhere in between.

Analog output seems great – why wouldn't you use it all the time? Unfortunately, not all pins have special PWM powers. Only pins 3, 5, 6, 9, 10 and 11 are able to produce analog outputs.