

Basics of Hardware with CircuitPython

Hi, and welcome to my talk on the basics of hardware with CircuitPython.

Hello!



I am Kira Hartlage

- ▣ Mechanical engineering background
- ▣ Consumer appliance design
- ▣ Self-taught software engineer
- ▣ Embedded software development

2

I am Kira Hartlage. I studied mechanical engineering in college and worked for about 8 years in consumer appliance design, designing physical parts. I learned how to code on my own as a hobby, and then got to the point where I felt comfortable enough to transition from my current role to a software engineer role at my company. Now, I write embedded firmware for our microcontrollers in refrigeration products at GE Appliances.

Desired Outcomes

- Learn a bit about embedded software development
- Learn a bit about hardware
- Learn how you can get started using CircuitPython

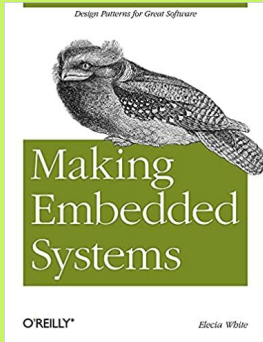
For this session, my goal is to teach you a bit about what embedded software development is, some details on hardware, and how you can get started creating your own hardware projects using CircuitPython.

1.

Embedded System

4

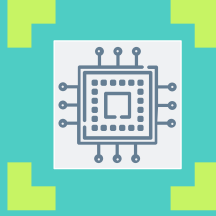
First up, what is an embedded system?



An embedded system is a computerized system that is purpose-built for its application.

- Elecia White

An embedded system is basically a system that is purpose-built for its application. It usually does what it's designed for and nothing else. It's not as multi-purpose as an operating system where you can run new programs on it without updating the operating system itself. It's usually firmware that is run on the bare metal of the microcontroller itself, and not a program that's run by an operating system.



Microcontroller

An integrated circuit with a processor, memory, and various input/output peripherals

6

So what is a microcontroller? It's simpler than a computer. It's an integrated circuit with a processor, some memory and various input and output peripherals that let it interact with the outside world and basically do things. Microcontrollers are designed for embedded systems, whereas microprocessors are used in personal computers and other general purpose applications.

What is a microcontroller?

1. Instructions - what the microcontroller is capable of doing
2. Registers - fast storage locations in memory that the microcontroller has access to that the instructions can use to do things
3. Memory - store code or data (usually slower than accessing registers)



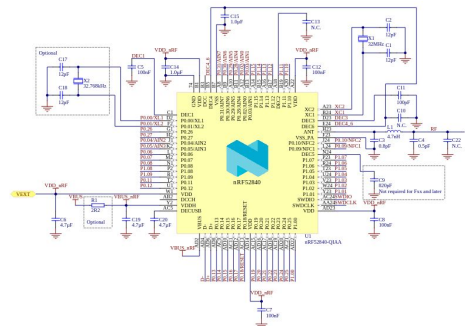
7

What are microcontrollers? They have 3 main pieces:

1. Instructions - what the microcontroller is capable of doing
2. Registers - which are fast storage locations in memory that the microcontroller has access to that the instructions can use
3. Other memory - which can be used to store your code and data, but they are usually slower for the microcontroller to access than it is for it to access the registers

What is a peripheral?

- GPIO (General purpose input/output)
 - Turn on/off LED
- ADC (Analog to digital control)
 - Read thermistor for room temperature
- DAC (Digital to analog control)
 - Convert audio from digital to analog signal
- PWM (Pulse-width modulation)
 - Controlling a variable speed motor or fan
- Hardware timers
- Serial communication
- Interrupts



https://infocenter.nordicsemi.com/index.jsp?topic=%2Fps_nrf52840%2Fref_circuitry.html

Some examples of peripherals are:

1. GPIOs (General Purpose Input/Output) - examples for this would be using it as an output to turn an LED on or off or reading a binary sensor input such as if a magnet passed over a hall effect sensor or not

1. ADC (Analog to Digital Control) - an example would be reading a thermistor for room temperature where the input is an analog count and you would translate that to a meaningful temperature

1. DAC (Digital to Analog Control) - an example would be video or audio converters that translate from digital to analog signals

1. PWM (Pulse-width modulation) - an example would be controlling a variable speed motor with a pulse-width signal, which is basically a square wave with varying frequency and/or duty-cycle

1. Hardware timers to keep time or count operations or events

1. Serial communication - such as UART, I2C, SPI - these are used to communicate between two devices such as microcontroller to microcontroller or sensor to microcontroller

1. Interrupts - an interrupt is a signal to the microcontroller that's emitted by hardware or software to indicate that an event

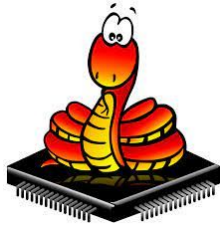
has happened and needs immediate attention; an example is pressing a key on a keyboard or clicking a mouse - this causes a hardware interrupt that will be serviced by the processor to read the key or mouse click and act accordingly

2. CircuitPython



<https://learn.adafruit.com/assets/49441>

CircuitPython



MicroPython - full Python compiler and runtime environment that runs on bare-metal (i.e. no operating system)



CircuitPython - easy to learn derivative of MicroPython created to get users up and running as quickly as possible

Commonly, C is the language of choice for writing code for microcontrollers. C is a compiled, low-level language that plays nicely with low-level hardware needs as it can be very small in size and efficient. It's small in terms of memory because it doesn't rely on as many libraries as higher-level languages do, and microcontrollers are very memory-constrained. C is a bit more difficult to learn than other languages and it can be difficult to get a microcontroller started up from scratch. But if you're looking to get into hardware, thankfully Adafruit has created a derivative of MicroPython (which is a full Python compiler and runtime environment that runs on bare-metal) known as CircuitPython. They created CircuitPython specifically to educate people on how to program microcontrollers and get up and going as quickly as possible.

There are many low-cost microcontroller boards, and with CircuitPython there are no special desktop software requirements. You can use any text editor to start editing code. You won't need to install a fancy compiler or need any special equipment to

upload the code either to the board. Since CircuitPython is based on Python, it's also easy to transition and get started with little prior programming experience.

Writing code for hardware

Challenges

- Limited memory (requiring C, assembly, or other low-level languages)
- No operating system (running on bare-metal)
- Startup code (lots of setup required just to get going and get an LED to blink)
- Really large (1000+ pages) manuals that are not user friendly
- Compilers, linkers, special equipment to flash the micro etc.

CircuitPython

- Designed for beginners and people already familiar or new to Python
- Can purchase boards with microcontrollers already running CircuitPython (no setup required!)
 - Plug and play with USB
- No special desktop software needed
 - Can edit in your text editor of choice
 - Save file and it uploads the code for you!

11

Some challenges with writing code for hardware that CircuitPython is able to address and make CircuitPython easier are:

- You usually have limited memory on a microcontroller. This requires a low-level and small in size language like C.
- There's no operating system. So you have to do everything from scratch.
- To do everything from scratch, you need to write the startup code to first be able to do anything at all. Again lots of difficult set up before you're ever able to just blink an LED.
- Microcontrollers usually have really long manuals with confusing acronyms and very few instructions on what to do.
- You'll need to deal with compilers, linkers, special equipment to flash the micro itself etc.

CircuitPython was designed for beginners and those familiar with Python, so you don't need to learn a new language. You can also purchase boards where the microcontroller already has CircuitPython on there, no startup code required. They've done that all behind the scenes for you, and usually give you a sample code file to start with. You can add/remove libraries as you need them to save space. Many boards that come with CircuitPython already on them are plug and play with USB. And you can edit the code in your text editor of choice, no need to special desktop software.

Adafruit

- A Minority and Woman-owned Business Enterprise
- “Adafruit was founded in 2005 by MIT engineer, Limor “Ladyada” Fried. Her goal was to create the best place online for learning electronics and making the best designed products for makers of all ages and skill levels.”



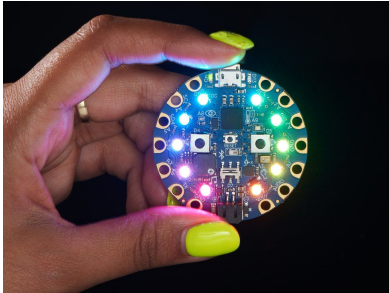
<https://www.adafruit.com/about>

<https://learn.adafruit.com/>

12

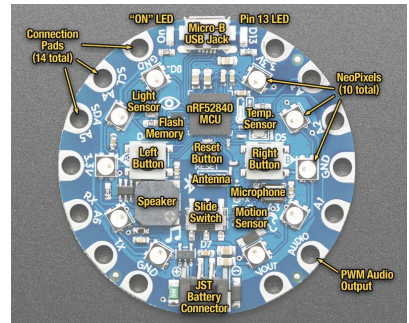
Adafruit, the company who created CircuitPython, is a minority and woman-owned business, and their main goal is to create a learning environment for everyone to be able to learn electronics and make cool gadgets. Lady Ada, as she is known, the founder of Adafruit, is the woman in the magazine picture with the pink hair.

CircuitPython w/ Circuit Playground Bluefruit



<https://cdn-shop.adafruit.com/970x728/4333-11.jpg>

- nRF52840 microcontroller
- Bluetooth Low Energy support for wireless connectivity
- 10 x mini NeoPixels, each one can display any color
- 1 x Motion sensor
- 1 x Temperature sensor (thermistor)
- 1 x Light sensor (phototransistor)
- 1 x Sound sensor (MEMS microphone)
- 1 x Mini speaker with class D amplifier
- 2 x Push buttons
- 1 x Slide switch
- And more!

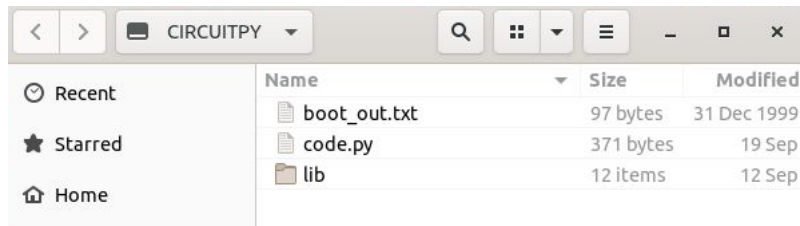


<https://learn.adafruit.com/adafruit-circuit-playground-bluefruit/guided-tour>

The board I will be using for my demonstrations is the Circuit Playground Bluefruit. It's the bluetooth version of the Circuit Playground Express. It's available from adafruit.com. It has...

Each pin on the microcontroller is connected to a peripheral (such as a sensor, button, LED etc.) or to voltage or ground. You will need to tell the microcontroller in the software what pin you want to use and how you want to use it in order to make something externally happen - like blink the LED or read the sensor value.

File Structure



Name	Size	Modified
boot_out.txt	97 bytes	31 Dec 1999
code.py	371 bytes	19 Sep
lib	12 items	12 Sep

14

When you plug in the board, it will mount to your computer. When you open it, you will usually see a `code.py` file and maybe some other boot up or startup code that it came with. If it's not there already, you should also add a `lib` folder to hold all of the libraries that you'll need. Because the microcontroller is running all on its own, it needs to have all of the library files on it at run-time. It can't pip install on the fly because it's not connected to your computer or the internet.

GPIO - Output - LEDs blinking



```
import board
import digitalio
import time

led = digitalio.DigitalInOut(board.LED)
led.direction = digitalio.Direction.OUTPUT

while True:
    led.value = True
    time.sleep(1)
    led.value = False
    time.sleep(1)
```

```
from adafruit_circuitplayground import cp
import time

while True:
    cp.red_led = True
    time.sleep(1)
    cp.red_led = False
    time.sleep(1)
```

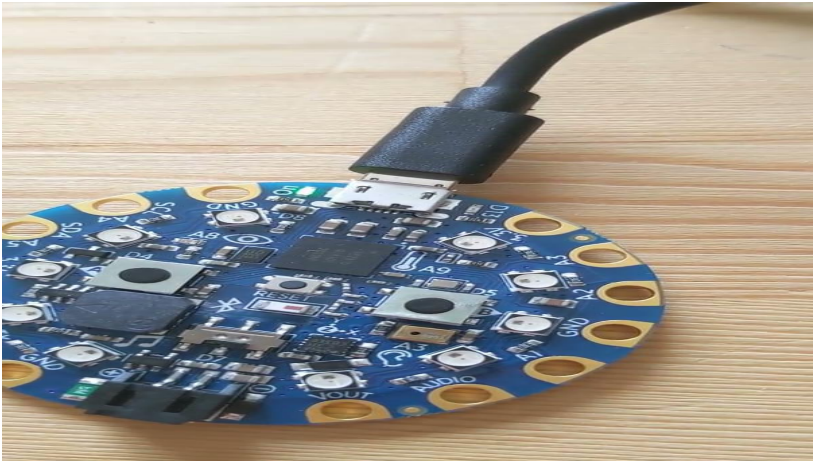
15

A GPIO is a general-purpose input/output. It's an uncommitted digital signal pin which can be used as input or output or sometimes both.

https://en.wikipedia.org/wiki/General-purpose_input/output

Two ways shown here are to do the same thing - blink an LED on for 1 second and off for 1 second. The first approach uses the digitalio library which requires more set up as you need to pick which pin on the board (board.LED) to use and tell it which direction it should be. Because we want to make it do something externally, it's an output. An input would be something like reading if a button is pressed or which way a switch is thrown. The second example on the right uses an Adafruit library specifically designed for this board, the Circuit Playground Bluefruit and the Circuit Playground Express. It hides even more of the set up for you. Here the board.LED is referred to as the red_led, and there's only one on this board, so it's easy to know which it means. The examples moving forward will be using the adafruit circuitplayground library. The code is usually shorter with this library but because some of the setup is obscured for you, keep in mind you might be missing out on some customization based on choices made by the library - like calibrating sensors or setting sensitivity thresholds.

Heartbeat blinking



16

Here's a video of the LED blinking. The Hello, World of hardware is what we call the heartbeat LED. We'll set an LED to blink on and off forever, so that we can tell if the board is up and running or if there's a problem (when the LED stops blinking).

Serial Console

- Can use print statements to help debug
- But where do the print statements go? There's no screen!
- On Linux, can you use something called "screen"

```
from adafruit_circuitplayground import cp
import time
```

```
while True:
    print("light on")
    cp.red_led = True
    time.sleep(1)
    print("light off")
    cp.red_led = False
    time.sleep(1)
```

Auto-reload is on. Simply save files over USB to run them or enter REPL to disable.

```
code.py output:
light on
light off
light on
light off
light on
light off
light on
light off
light on
light off
light on
light off
```

<https://learn.adafruit.com/welcome-to-circuitpython/kattni-connecting-to-the-serial-console>

17

Because when you're flashing or using USB power for the board, you are also able to use the serial console to see print statements. You can use print statements to help debug your code. To see the serial console output, on Linux you can use something called "screen". Adafruit's website has instructions for other OS's. Here's an example showing these two print statements while the LED is actually blinking.

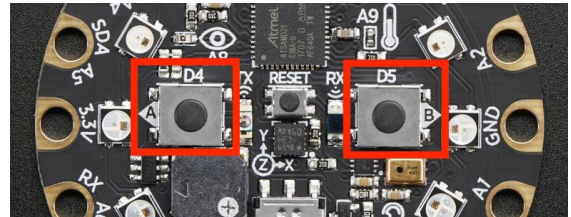
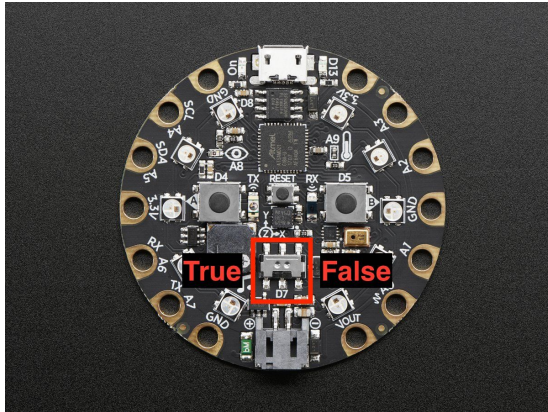
REPL

Press any key to enter the REPL. Use CTRL-D to reload.

```
Adafruit CircuitPython 6.2.0 on 2021-04-05; Adafruit Circuit Playground Bluefruit with nRF52840
>>> import board
>>> import digitalio
>>> led = digitalio.DigitalInOut(board.LED)
>>> led.direction = digitalio.Direction.OUTPUT
>>> led.value = True
>>> █
```

There is also a REPL, and you can write code in the REPL and it will actually run on the Circuit Playground Bluefruit board itself. Here I am turning on the LED from the REPL. (Do in real-time?)

GPIO - Input - Buttons and switch



<https://learn.adafruit.com/circuitpython-made-easy-on-circuit-playground-express/buttons>

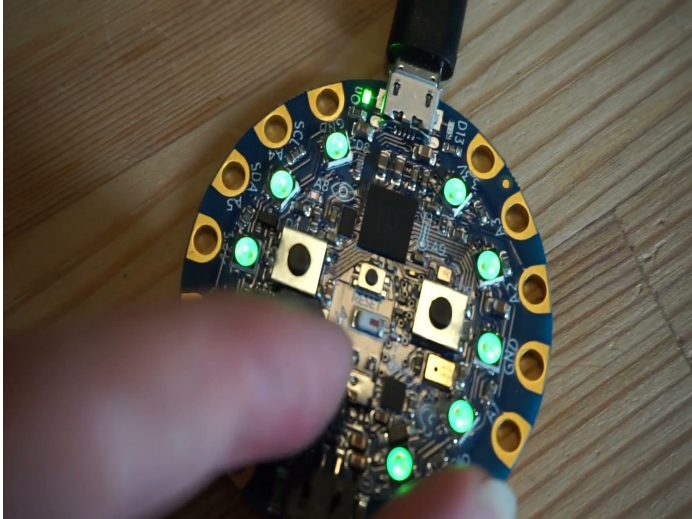
<https://circuitpython.readthedocs.io/projects/circuitplayground/en/latest/api.html>

19

There is a switch on the board. When it's in the left position, the micro reads it as true, and when it's in the right position, the micro reads it as false. Reading the position of the switch allows you to make more interesting logic as you can base it off whether someone moved the switch to the left or the right.

There are also two buttons on the board. When a button is pressed, we can read that as either true or false on the micro as well.

Buttons and switch



```
from adafruit_circuitplayground import cp

cp.pixels.brightness = 0.01

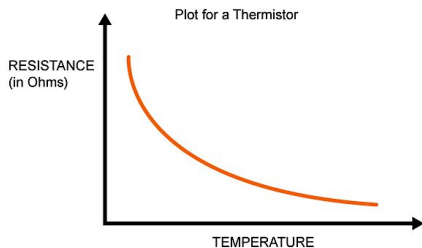
green = (0, 255, 0)
pink = (255, 100, 150)
red = (255, 0, 0)
yellow = (255, 255, 0)

while True:
    if cp.switch:
        cp.pixels.fill(green)
    else:
        if cp.button_a:
            cp.pixels.fill(pink)
        elif cp.button_b:
            cp.pixels.fill(yellow)
        else:
            cp.pixels.fill(red)
```

20

Here's an example of doing multiple things based on input from the buttons and switch. The main while loop is the main control loop, and it executes forever. It first checks if the switch is true, meaning is it to the left, if it is, it will fill the pixels green. Else if button A is pressed, it will do pink, then if button B is pressed, it will do yellow. If neither button is pressed, it will do red. If both button A and button B will pressed, it will go through each branch of the logic there, but it will be so quick, that the pixels will fill with yellow, and your eyes won't even see it fill them with pink.

ADC - Temperature



```
from adafruit_circuitplayground import cp
import time

cp.pixels.brightness = 0.01
delay_time = 0.5
colors = [(0, 0, 200), (0, 0, 220), (0, 0, 255),
          (100, 0, 215), (200, 0, 150), (200, 0, 200),
          (215, 0, 40), (255, 0, 0), (255, 0, 0), (255, 0, 0)]

while True:
    temperature = cp.temperature
    for pixel in range(min(temperature / 4, 10)):
        cp.pixels[pixel] = colors[pixel]
    time.sleep(delay_time)
```

21

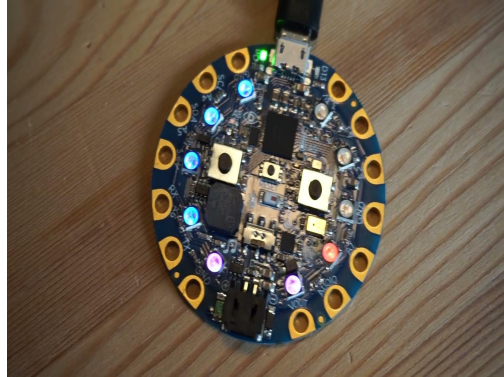
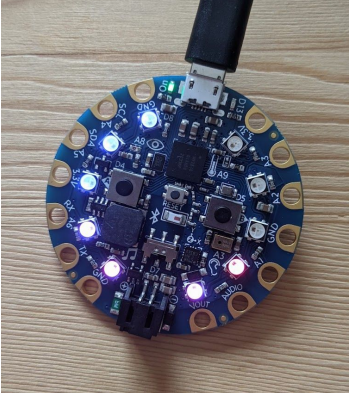
ADC or analog to digital converters convert an analog signal to a digital signal. Microcontrollers speak digitally and do not understand many real-time/world signals which are continuous signals with continuous values. Temperature is one such signal. Microcontrollers can actually only see different levels of voltages, which depends on the resolution of the ADC and the system voltage. ADCs convert the analog signal to digital by sampling the continuous signal and then quantifying it to determine the resolution and then set binary values that are read by the microcontroller as a digital signal.

One way to measure temperature is by using a thermistor. A thermistor's resistance changes in response to temperature. With the change in resistance, the voltage is also changed proportionally, and the microcontroller is able to convert it to a digital value.

The example here uses the Circuit Playground library again in which you do not have to know any of the details of the thermistor or the conversion on the low-level. It's done for you. You can read the value using `cp.temperature`. It's in Celsius.

<https://www.arrow.com/en/research-and-events/articles/engineering-resource-basics-of-analog-to-digital-converters>

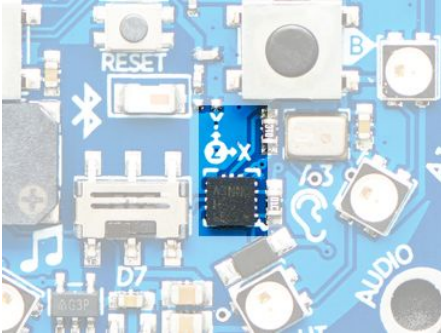
Temperature w/ LEDs



Here's an example that converts the temperature into a visual display. The warmer the temperature, the more LEDs will light up. The colors corresponding to each LED are kept in an array, and used if that LED should be lit or not based on the temperature.

ADC - Accelerometer

Tippy Lights Example modified from
<https://www.linkedin.com/learning/learning-circuitpython-with-circuit-playground-express/understanding-basic-circuits>



Can detect 3-axis acceleration

```
from adafruit_circuitplayground import cp
import time

pink = (200, 0, 200)
clear = (0, 0, 0)
cp.pixels.brightness = 0.1
delay_time = 0.05

while True:
    for pixel in range(10):
        cp.pixels[pixel] = pink
        time.sleep(delay_time)

        cp.pixels[pixel] = clear
        delay_time = abs(cp.acceleration.y - 10)* 0.03
```

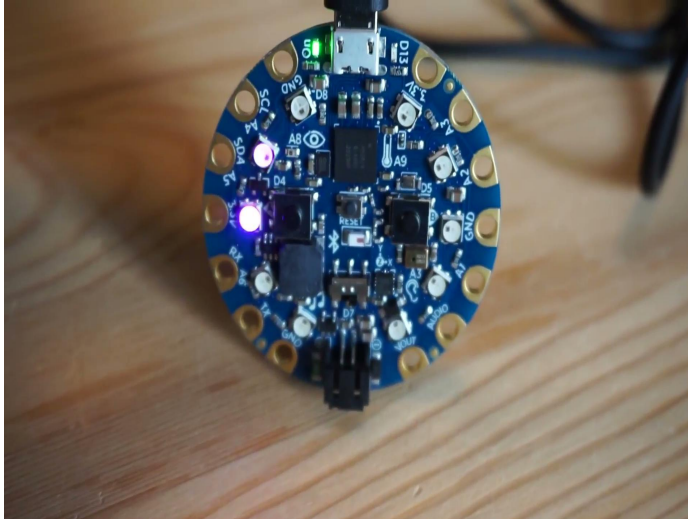
23

<https://learn.adafruit.com/bluefruit-playground-app/accelerometer>

Print acceleration to serial console while tilting it

Another ADC input is the on-board accelerometer. It can detect 3-axis acceleration. This program here will make the lights light up in a circle at different speeds based on the acceleration in the y-axis. This example is based on an example in a CircuitPython LinkedIn class called tippy lights.

Accelerometer - Tippy Lights



24

Here you can see that the lights spin faster when the board is flat on the table, that's when there is acceleration in the y-axis is greatest due to gravity. The delay time between lighting the LEDs is very close to zero. When I hold the board up, the acceleration in the y-axis is essentially zero, the delay time increases, and the lights slow down.

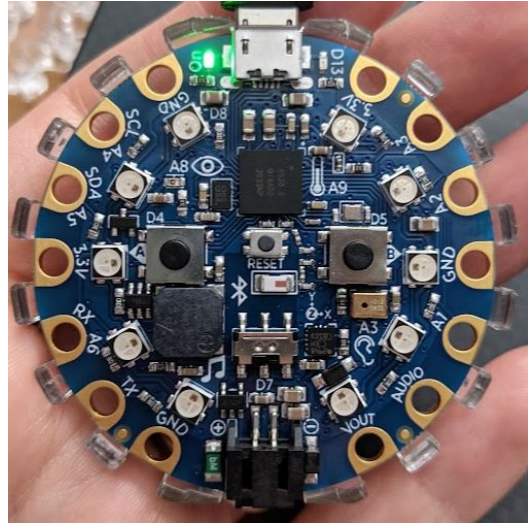
Capacitive Touch and Sound

```
import time
from adafruit_circuitplayground import cp

scale = [523, 587, 659, 698, 784, 880, 988]
cp.adjust_touch_threshold(200)

while True:
    if cp.touch_A1:
        cp.play_tone(scale[0], 0.25)
    if cp.touch_A2:
        cp.play_tone(scale[1], 0.25)
    if cp.touch_A3:
        cp.play_tone(scale[2], 0.25)
    if cp.touch_A4:
        cp.play_tone(scale[3], 0.25)
    if cp.touch_A5:
        cp.play_tone(scale[4], 0.25)
    if cp.touch_A6:
        cp.play_tone(scale[5], 0.25)
    if cp.touch_TX:
        cp.play_tone(scale[6], 0.25)

    time.sleep(0.05)
```



25

Capacitive touch is another interesting technology. When you touch a screen or in this case these copper contact points around the board, an electric circuit is completed at the point of contact made by your finger on the screen. This contact changes the electrical charge at that location. The change in electrical charge can be detected by the microcontroller.

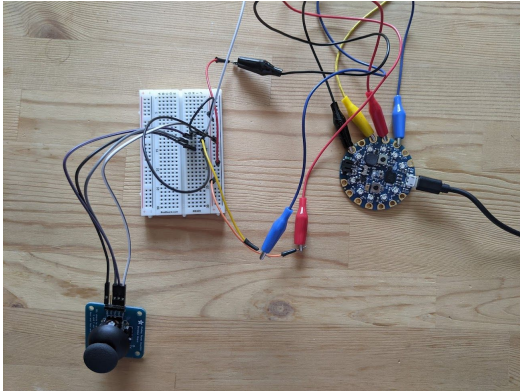
Cap Touch and Play a Note



26

Here's an example that plays a different tone with the on-board speaker depending on which touchpad is touched. This uses a play tone library feature where you give it the frequency of the tone, but you can also add audio files onto the board and you can play custom tunes as well with the on-board speaker.

Adding external components - potentiometer (ADC)



```
from adafruit_circuitplayground import cp
import time
import board
import analogio

cp.pixels.brightness = 0.05
cp.pixels.auto_write = False

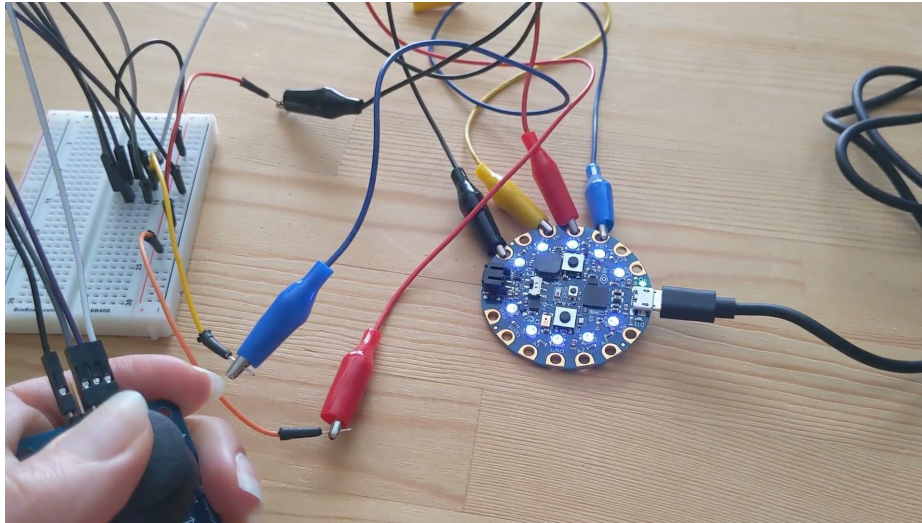
knob = analogio.AnalogIn(board.A6)

while True:
    cp.pixels.fill((255, 0, 0))
    pot_value = round(knob.value/65535 * 100)
    cp.pixels.brightness = pot_value/100
    cp.pixels.show()
    time.sleep(0.1)
```

27

You can extend the bluefruit board to include other components. Here's an example where I hooked up a joystick to the bluefruit. You could also use a potentiometer which is a three-terminal resistor with a sliding or rotating contact that forms an adjustable voltage divider. This again is another type of ADC input; the different voltage levels correspond to different readings. I am able to read the value into the knob.value variable, and I translate that into a pot_value. The knob.value (ADC value) ranges from 0 to 65535, and by multiplying that raw reading by 100 and dividing by 65535, I can normalize it so that pot_value ranges from 0 to 100. I then divide that by 100 to determine the brightness of the lights. Which in turn makes this a light dimmer.

Joystick LED Dimmer and Color Changer



```
from adafruit_circuitplayground import cp
import time
import board
import analogio

cp.pixels.brightness = 0.05
cp.pixels.auto_write = False

pink = (255, 50, 174)
blue = (0, 0, 255)
green = (0, 255, 0)

knob_x = analogio.AnalogIn(board.A6)
knob_y = analogio.AnalogIn(board.A5)

while True:
    pot_value_x = round(knob_x.value/65535 * 100)
    pot_value_y = round(knob_y.value/65535 * 100)

    brightness = (pot_value_x + pot_value_y)/2000
    cp.pixels.brightness = brightness

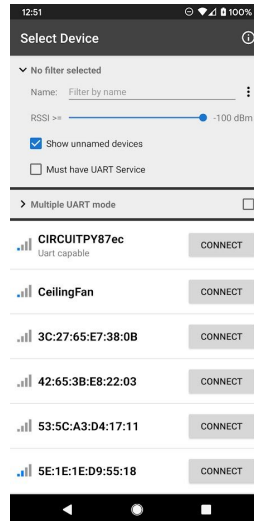
    if brightness < 0.03:
        cp.pixels.fill(blue)
    elif brightness < 0.05:
        cp.pixels.fill(green)
    else:
        cp.pixels.fill(pink)

    cp.pixels.show()
    time.sleep(0.1)
```

28

With the joystick, I'm able to read different potentiometer values for the x and y directions. I use the combination of those to values to determine what the brightness of the lights should be as well as what color to fill the lights. Here's you can see me move the joystick around and have the lights adjust based on the position of the stick.

Bluetooth with the Bluefruit



Add picture of options

<https://learn.adafruit.com/bluefruit-le-connect>

29

Adafruit has an app called Bluefruit LE Connect that you can use to connect to the Bluefruit board. Here's what it looks like when you select the device and connect.
<https://learn.adafruit.com/bluefruit-le-connect>

UART - Acceleration



```
from adafruit_ble import BLERadio
from adafruit_ble.advertising.standard import ProvideServicesAdvertisement
from adafruit_ble.services.nordic import UARTService
from adafruit_circuitplayground import cp
import time

ble = BLERadio()
uart = UARTService()
advertisement = ProvideServicesAdvertisement(uart)

ble.start_advertising(advertisement)
print("Waiting to connect")

while True:
    while not ble.connected:
        pass

    result = (round(cp.acceleration.x, 2), round(cp.acceleration.y, 2), round(cp.acceleration.z, 2))

    if result:
        try:
            uart.write(str(result).encode("utf-8"))
        except Exception as e:
            print(repr(e))

    time.sleep(0.5)
```

30

Once connected there are many options and things to control or read from the board, but here I am going to show you that you can send the acceleration data over UART to the app. UART is a communication protocol where you send messages serially. Here the message is just the x, y, z value of the acceleration.

You can also plot data. There's a control pad and buttons that you can use to control the board. There's also a color picker if you want to pick an RGB color and send it to the device to fill the pixels.

UART - Universal asynchronous receiver/transmitter is a communication protocol. They transmit data asynchronously, meaning there is no clock signal that synchronizes the transmitter and receiver. Two UARTS communicate directly with each other, meaning there's one transmission line and one receiving line on each and they are only connected to each other. The app and the board can communicate with each other this way, and you can send/receive data.

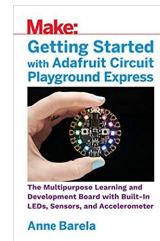
External Libraries

- External libraries available for various microcontrollers and peripherals
- Can download all of the latest library files in a bundle here:
<https://circuitpython.org/libraries>
- Tool available to keep your libraries up to date on your board:
<https://pypi.org/project/circup/>
- Due to memory constraints, you may not be able to include everything on your board. Only include what you need.

There are many other libraries available out there to be explored and used. You can download the latest library files in a bundle at the link listed. There's also a tool available to keep the libraries up to date on the board. Just keep in mind that due to memory constraints, you might not be able to include everything on your board. It's best to just copy them over as you need them.

More Resources

- <https://learn.adafruit.com/>
- <https://learn.adafruit.com/category/circuit-playground>
- <https://circuitpython.readthedocs.io>
- <https://www.linkedin.com/learning/learning-circuitpython-with-circuit-playground-express/understanding-basic-circuits>
- <https://www.amazon.com/Getting-Started-Adafruit-Circuit-Playground/dp/1680454889>
- <https://makecode.adafruit.com/>



More resources to get started are listed here.

Thanks!

Any questions?

Resources located at:

<https://github.com/kirakirakira/hardware-circuitpython>

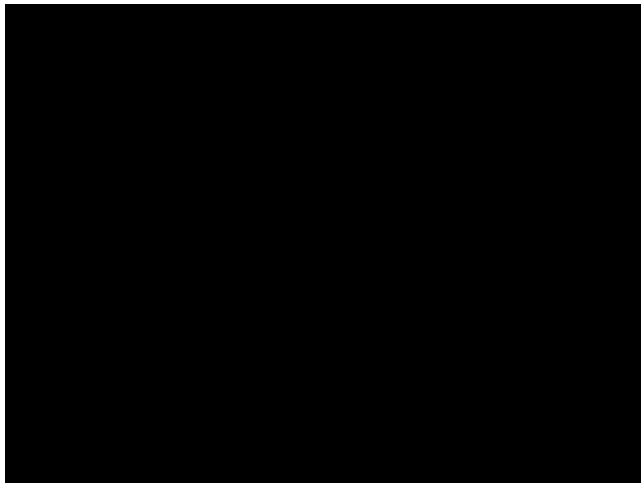
Contact me at kira.hartlage@gmail.com

Credits

Special thanks to all the people who made and released these awesome resources for free:

- Presentation template by [SlidesCarnival](#)
- Photographs by [Unsplash](#)

Light Dimmer Video #2



Control from the app?

36

UART - Universal asynchronous receiver/transmitter is a communication protocol. They transmit data asynchronously, meaning there is no clock signal that synchronizes the transmitter and receiver. Two UARTS communicate directly with each other, meaning there's one transmission line and one receiving line on each and they are only connected to each other. The app and the board can communicate with each other this way, and you can send/receive data.

Here's an example of sending the acceleration data to the app.

Can control the lights from the phone app