Circuit Playground Hourglass

Created by Carter Nelson

https://learn.adafruit.com/circuit-playground-hourglass
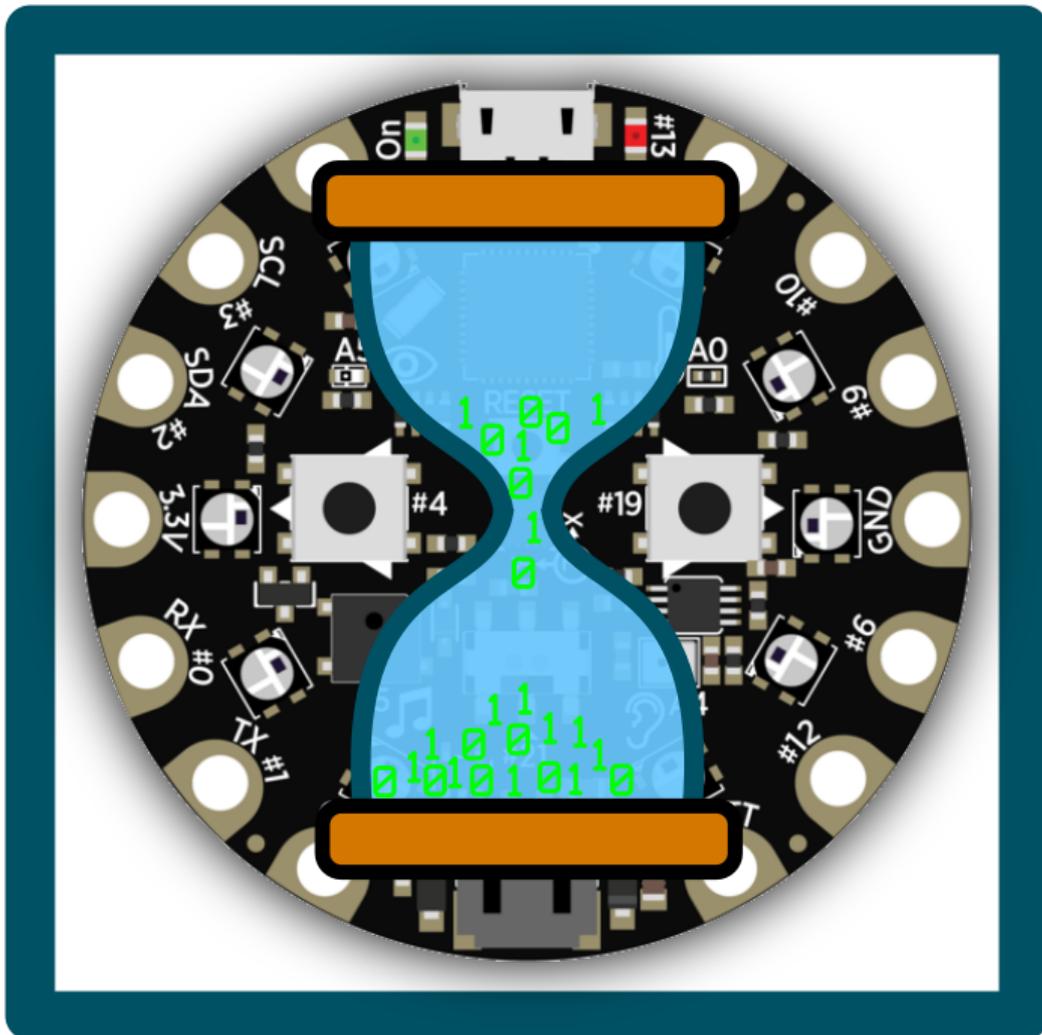
Last updated on 2021-11-15 06:50:29 PM EST

Table of Contents

Overview

In this guide we'll go through the steps to create an hourglass style timer with your Circuit Playground.

An hourglass (https://adafru.it/sSf) is an ancient timing device used to measure a fixed amount of time. It contains two chambers separated by a thin neck that allows sand to pass from one chamber to the other very slowly. The overall size and amount of sand determines the amount of time. Once all the sand has emptied into one side, the hourglass is simply inverted to start counting time again.

Here, we'll use the NeoPixels on the Circuit Playground to represent the grains of sand in an hourglass, having them "disappear" one at a time by turning them off. We can even have the hourglass "reset" by turning the Circuit Playground over. We'll build this up one step at a time and walk through each one.

Required Parts

This project uses the sensors already included on the Circuit Playground, either a Classic (http://adafru.it/3000) or an Express (http://adafru.it/3333). The only additional items needed are batteries for power and a holder for the batteries.
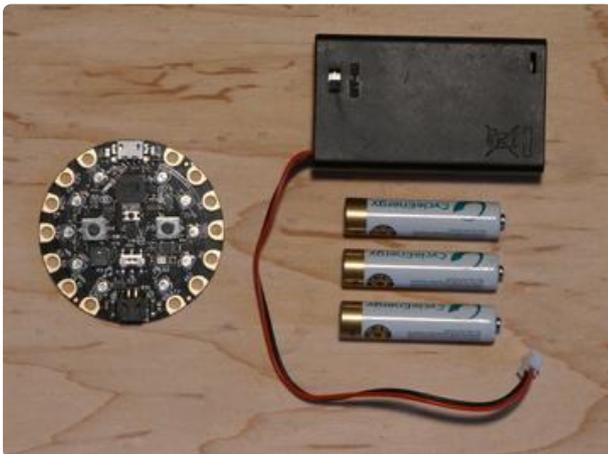
- Circuit Playground
  - Classic (http://adafru.it/3000)
  - Express (http://adafru.it/3333)
- 3 x AAA Battery Holder (http://adafru.it/727)
- 3 x AAA Batteries (NiMH work great!)

Before Starting

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

Circuit Playground Classic

-
-

Circuit Playground Express

-

Arduino

Arduino (https://adafru.it/lDg) versions of the code are provided on the following pages.

> The Arduino code on the following pages has been tested and verified to work on either a Circuit Playground Classic or Express.

Basic Timer

An hourglass does one thing, count a fixed period time. We can do the same thing very easily with the Circuit Playground using the `delay()` function from the core Arduino library (https://adafru.it/t9e). This function simply takes an amount of time, in milliseconds, and waits that amount of time.

> 1 second = 1000 milliseconds

So we can make a basic timer by turning on all of the NeoPixels, waiting the specified amount of time using `delay()`, and then turning off all of the NeoPixels to indicate the time has elapsed.

Easy peasy lemony squeezy, here's the code to do that:

```
/////////////////////////////////////////////////////////////////////
// Circuit Playground Basic Timer
//
// Author: Carter Nelson
// MIT License (https://opensource.org/licenses/MIT)
```

```
#include &lt;Adafruit_CircuitPlayground.h&gt;

/////////////////////////////////////////////////////////////////////////
void setup() {
  // Initialize the Circuit Playground
  CircuitPlayground.begin();
}

/////////////////////////////////////////////////////////////////////////
void loop() {
  // Turn ON all the NeoPixels
  for (int p=0; p&lt;10; p++) {
    CircuitPlayground.setPixelColor(p, 255, 255, 255);
  }

  // Wait 5 seconds (5000 milliseconds)
  delay(5000);

  // Turn OFF all the NeoPixels
  CircuitPlayground.clearPixels();

  // Wait for button press to reset timer
  while (!CircuitPlayground.leftButton()   &amp;&amp;
         !CircuitPlayground.rightButton())  {
    // Do nothing, just waiting for a button press...
  }
}
```

In the code above, the timer was hard coded for 5 seconds. Since there are 1000 milliseconds in a second, we need to multiply 5 by 1000 to get the value for `delay()`.
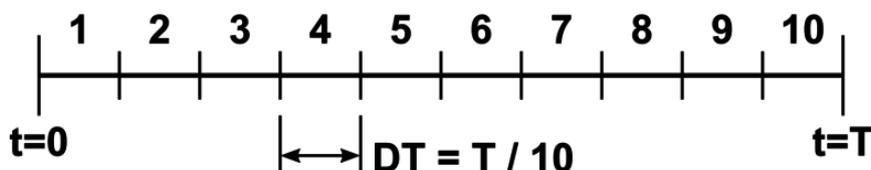
5 seconds X 1000 milliseconds/second = 5000 milliseconds

Next

In a real hourglass, the sands fall through slowly, not all at once. So next we'll look at how to turn off the NeoPixels one a time to simulate the grains of sand.

Less Basic Timer

Now let's see how to turn off the NeoPixels one at a time during the count time. The basic idea is shown in the figure below.

Our timer starts at time zero (t=0) and counts for a total amount of time T (t=T). Since we have 10 NeoPixels on the Circuit Playground, we can break that time period up in to 10 equal slices. Each slice has a wait time of DT, which is equal to the total time T, divide by the number of NeoPixels, thus:

DT = T / 10

The idea now is to loop over each NeoPixel and wait this smaller amount of time DT. Once that time has elapsed, turn the NeoPixel off and move on to the next NeoPixel, etc.

Here's the previous code modified to do this:

```
/////////////////////////////////////////////////////////////////////////
// Circuit Playground Less Basic Timer
//
// Author: Carter Nelson
// MIT License (https://opensource.org/licenses/MIT)

#include <Adafruit_CircuitPlayground.h>

/////////////////////////////////////////////////////////////////////////
void setup() {
  // Initialize the Circuit Playground
  CircuitPlayground.begin();
}

/////////////////////////////////////////////////////////////////////////
void loop() {
  // Turn ON all the NeoPixels
  for (int p=0; p<10; p++) {
    CircuitPlayground.setPixelColor(p, 255, 255, 255);
  }

  // Compute DT
  unsigned long DT = 5000/10;

  // Turn OFF the NeoPixels one at a time, waiting DT each time
  for (int p=0; p<10; p++) {
    delay(DT);
    CircuitPlayground.setPixelColor(p, 0, 0, 0);
  }

  // Wait for button press to reset timer
  while (!CircuitPlayground.leftButton()  &&
         !CircuitPlayground.rightButton())  {
    // Do nothing, just waiting for a button press...
  }
}
```

As you can see, the `delay()` command has been moved inside a loop. The loop runs 10 times, so the `delay()` command gets called 10 times. However, the total amount of time counted remains the same 5 seconds as before.

Next Step

Hourglasses typically don't have buttons that are pressed to reset them like our Circuit Playground code is doing. An hourglass is reset by simply flipping it over and letting the sand run through in the other direction. How can we simulate this action with the Circuit Playground? Well, maybe we can use the accelerometer. Let's give that a try.

Flip Detect

To make things easy, let's assume the hourglass timer will be used with the Circuit Playground facing up. The NeoPixels are on that side, so this makes sense. It also let's us use the Z axis of the accelerometer to detect when the Circuit Playground is flipped over. If you want more info on the accelerometer, see the this guide (https://adafru.it/t9f).

The figure below shows a time history of the value returned from `CircuitPlayground.motionZ()` with the Circuit Playground facing up and down.
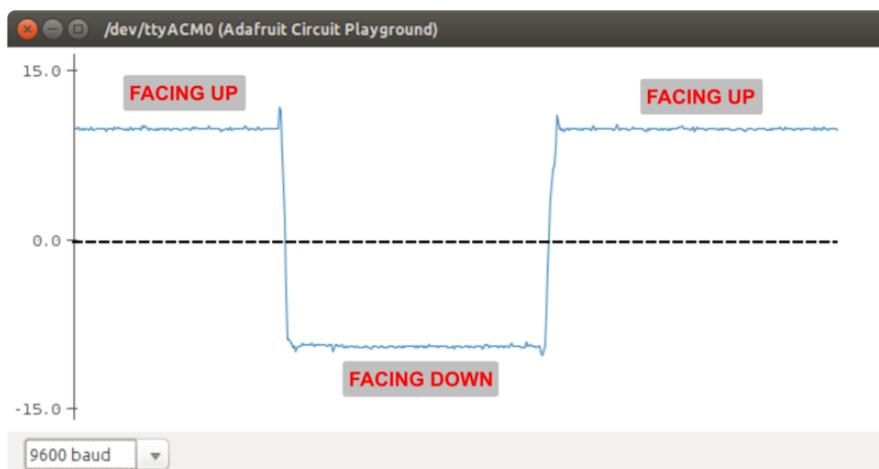
The value of 9.8 m/s^2 relates to earth gravity. With the Circuit Playground facing up, it is a positive value. With the Circuit Playground facing down, it is a negative value. So we can detect a flip by simply checking the sign of the value.

We start with the Circuit Playground face up and a positive value. The first step is to wait for the value to become negative. Then, once it is negative, we wait for it to become positive again. That's a flip!

Here's a simple sketch that demonstrates this logic being used to detect a flip and then play a tone.

```
///////////////////////////////////////////////////////////////////////
// Circuit Playground Flip Detect
//
// Author: Carter Nelson
// MIT License (https://opensource.org/licenses/MIT)

#include <Adafruit_CircuitPlayground.h>

///////////////////////////////////////////////////////////////////////
void setup() {
  // Initialize the Circuit Playground
  CircuitPlayground.begin();
}

///////////////////////////////////////////////////////////////////////
void loop() {
  // Wait for Circuit Playground to be flipped over (face down)
  while (CircuitPlayground.motionZ() > 0) {};

  // A little debounce
  delay(500);

  // Wait for Circuit Playground to be flipped back over (face up)
  while (CircuitPlayground.motionZ() < 0) {};

  // Make a beep noise.
  CircuitPlayground.playTone(800,1000);
}
```

The little lines of code that look like:

```
while (CircuitPlayground.motionZ() > 0) {};
```

are called "parking loops". If we wanted them to actually do something, we would place that code in the `{}` brackets. In this case all we want to do is wait until the condition being tested becomes true. So the program execution just "parks" here until that happens.

The line of code with the small delay:

```
delay(500);
```

is needed to prevent false flip detection. This can occur as the processor runs fast enough that it might see a brief positive value immediately after the first negative value, due to noise or small amounts of real motion. By waiting a small amount of time before checking the accelerometer value again, we prevent this.

Next Step

OK, let's use this flip logic to alter our previous code so it behaves more like an hourglass. Instead of resetting the hourglass with the push buttons, we'll use this flip detect.

Basic Hourglass

Let's add the flip detect logic from the previous section to the less basic timer code to make a basic hourglass.

Here's the code:

```
/////////////////////////////////////////////////////////////////////////////
// Circuit Playground Basic Hourglass
//
// Author: Carter Nelson
// MIT License (https://opensource.org/licenses/MIT)

#include <Adafruit_CircuitPlayground.h>

/////////////////////////////////////////////////////////////////////////////
void setup() {
  // Initialize the Circuit Playground
  CircuitPlayground.begin();
}

/////////////////////////////////////////////////////////////////////////////
void loop() {
  // Turn ON all the NeoPixels
  for (int p=0; p<10; p++) {
    CircuitPlayground.setPixelColor(p, 255, 255, 255);
  }

  // Compute DT
  unsigned long DT = 5000/10;

  // Turn OFF the NeoPixels one at a time, waiting DT each time
  for (int p=0; p<10; p++) {
    delay(DT);
    CircuitPlayground.setPixelColor(p, 0, 0, 0);
  }

  // Wait for Circuit Playground to be flipped over (face down)
  while (CircuitPlayground.motionZ() > 0) {};

  // A little debounce
  delay(500);

  // Wait for Circuit Playground to be flipped back over (face up)
  while (CircuitPlayground.motionZ() < 0) {};
}
```

So at this point we have a basic working hourglass. It counts for a period of 5 seconds, turning off the NeoPixels one at a time, and can be reset by flipping the Circuit Playground over.

Changing Count Time

If you want to set the timer to a different amount of time, simply change this line of code:

```
unsigned long DT = 5000/10;
```

and replace 5000 with whatever value you want. However, a better way to do this is to define this value globally at the top of the code. Try this - first, modify the code by adding a #define statement near the top as shown below.

```
#include &lt;Adafruit_CircuitPlayground.h&gt;

#define COUNT_TIME    5000   // milliseconds
```

Now change the line that computes DT as follows:

```
unsigned long DT = COUNT_TIME / 10;
```

This makes it easier to find and change the value in the future.

Next Step

Try setting COUNT_TIME to something longer than 5 seconds, like 30 seconds:
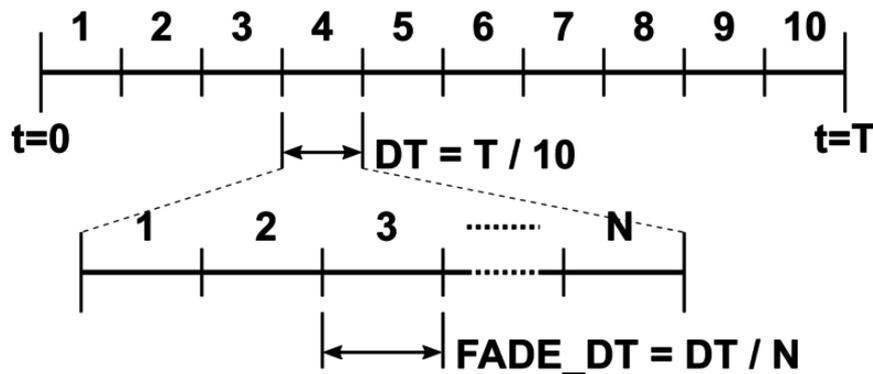
```
#define COUNT_TIME    30000     // milliseconds
```

Run this and watch what happens when the count gets to the end. It works fine, but since the total time (COUNT_TIME) is now longer, so is the delay (DT) at each NeoPixel. This makes it difficult to see the last bit of time elapsing. The last NeoPixel is on, then suddenly off.

So how can we indicate the time progress for each NeoPixel? How about fading them? That might work, let's give it a try.
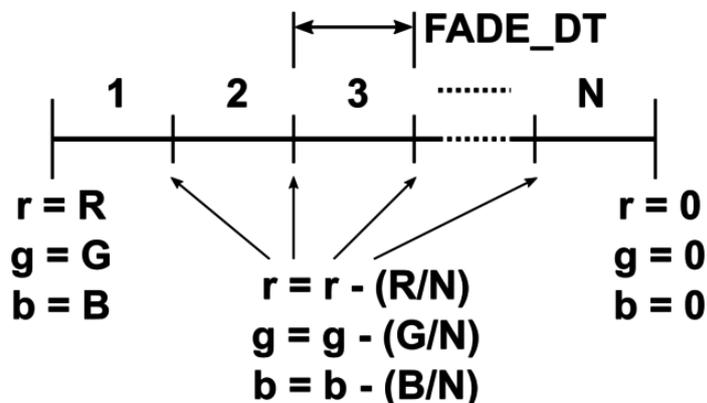
Fading Hourglass

Our previous code simply waited at each NeoPixel the computed amount of time DT, and then turned the NeoPixel off. Now let's change things so that the NeoPixel fades out over this period of time.

Basically, we need to go one level deeper on our time line. Now, within each time slice DT, instead of doing nothing, we'll be fading the NeoPixel. See the figure below.

The value N is just an arbitrary number of steps over which the fading will occur. This further divides the time slice DT into smaller slices, called FADE_DT in the figure above.

The simplest way to fade a NeoPixel is to linearly change the setting from its starting value down to zero. The NeoPixels on the Circuit Playground have red, green, and blue LEDs in them, so we do the same thing for each of the color components individually. This idea is shown in the figure below.

So at each of the fading time steps, we decrease the red, green, and blue values by a small amount. After doing this N times, the values will reach 0.

OK, here's our final code, with flip-to-reset and fading NeoPixel grains of sand.

```
/////////////////////////////////////////////////////////////////////////////
// Circuit Playground Fading Hourglass
//
// Author: Carter Nelson
// MIT License (https://opensource.org/licenses/MIT)

#include <Adafruit_CircuitPlayground.h>

#define COUNT_TIME    30000 // milliseconds
#define FADE_STEPS    100   // NeoPixel fade steps
#define R_SAND        255   // Sand color RED value
#define G_SAND        255   // Sand color GREEN value
#define B_SAND        255   // Sand color BLUE value

unsigned long DT;
float r, g, b;
float dr, dg, db;

/////////////////////////////////////////////////////////////////////////////
void setup() {
  Serial.begin(9600);

  // Initialize the Circuit Playground
  CircuitPlayground.begin();

  // Compute per NeoPixel wait time
  DT = COUNT_TIME / 10;

  // Compute the color value change per fade step
  dr = float(R_SAND) / float(FADE_STEPS);
  dg = float(G_SAND) / float(FADE_STEPS);
  db = float(B_SAND) / float(FADE_STEPS);
}

/////////////////////////////////////////////////////////////////////////////
void loop() {
  // Turn ON all the NeoPixels
  for (int p=0; p<10; p++) {
    CircuitPlayground.setPixelColor(p, uint8_t(R_SAND),
                                       uint8_t(G_SAND),
                                       uint8_t(B_SAND));
  }

  // Loop over each NeoPixel
  for (int p=0; p<10; p++) {
    // Set the start RGB values
    r = float(R_SAND);
    g = float(G_SAND);
    b = float(B_SAND);
    // Loop over each fading step
    for (int n=0; n<FADE_STEPS; n++) {
      delay(DT/FADE_STEPS); // Per fade step delay
      r = r - dr;           // Decrease the red value
      g = g - dg;           //    "       "  green "
      b = b - db;           //    "       "  blue  "
      CircuitPlayground.setPixelColor(p, uint8_t(r),
                                         uint8_t(g),
                                         uint8_t(b));
    }
  }

  // Wait for Circuit Playground to be flipped over (face down)
  while (CircuitPlayground.motionZ() > 0) {};

  // A little debounce
```

```
    delay(500);

    // Wait for Circuit Playground to be flipped back over (face up)
    while (CircuitPlayground.motionZ() &lt; 0) {};
}
```

CircuitPython

CircuitPython (https://adafru.it/AFI) versions of the code are provided on the following pages.

CircuitPython only works on the Circuit Playground Express.

Basic Timer

An hourglass does one thing, count a fixed period time. We can do the same thing very easily with the Circuit Playground using the `sleep()` function from the time module (https://adafru.it/Cgy). This function simply takes an amount of time, in seconds, and waits that amount of time.

1 second = 1000 milliseconds

So we can make a basic timer by turning on all of the NeoPixels, waiting the specified amount of time using `sleep()`, and then turning off all of the NeoPixels to indicate the time has elapsed.

Easy peasy lemony squeezy, here's the code to do that:

```
# Circuit Playground Express Basic Timer
#
# Author: Carter Nelson
# MIT License (https://opensource.org/licenses/MIT)
import time
from adafruit_circuitplayground.express import cpx

while True:
    # Turn ON all the NeoPixels
    cpx.pixels.fill((255, 255, 255))

    # Wait 5 seconds
    time.sleep(5)

    # Turn OFF all the NeoPixels
    cpx.pixels.fill((0, 0, 0))

    # Wait for button press to reset timer
    while not cpx.button_a or cpx.button_b:
        pass # do nothing
```
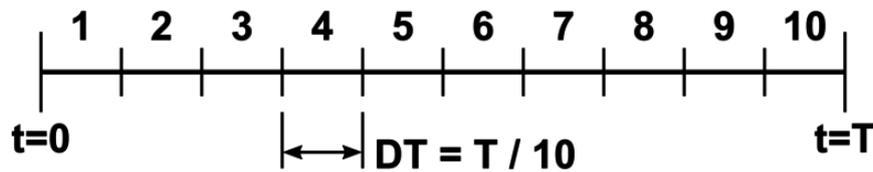
Less Basic Timer

Now let's see how to turn off the NeoPixels one at a time during the count time. The basic idea is shown in the figure below.

Our timer starts at time zero (t=0) and counts for a total amount of time T (t=T). Since we have 10 NeoPixels on the Circuit Playground, we can break that time period up in to 10 equal slices. Each slice has a wait time of DT, which is equal to the total time T, divide by the number of NeoPixels, thus:

DT = T / 10

The idea now is to loop over each NeoPixel and wait this smaller amount of time DT. Once that time has elapsed, turn the NeoPixel off and move on to the next NeoPixel, etc.

Here's the previous code modified to do this:

```
# Circuit Playground Express Less Basic Timer
#
# Author: Carter Nelson
# MIT License (https://opensource.org/licenses/MIT)
import time
from adafruit_circuitplayground.express import cpx

while True:
    # Turn ON all the NeoPixels
    cpx.pixels.fill((255, 255, 255))

    # Compute DT
    DT = 5 / 10

    # Turn OFF the NeoPixels one at a time, waiting DT each time
    for p in range(10):
        time.sleep(DT)
        cpx.pixels[p] = (0, 0, 0)

    # Wait for button press to reset timer
    while not cpx.button_a or cpx.button_b:
        pass # do nothing
```

Flip Detect

To make things easy, let's assume the hourglass timer will be used with the Circuit Playground facing up. The NeoPixels are on that side, so this makes sense. It also let's us use the Z axis of the accelerometer to detect when the Circuit Playground is flipped over. If you want more info on the accelerometer, see the this guide (https://adafru.it/t9f).

The figure below shows a time history of the value returned from `cpx.acceleration[2]` with the Circuit Playground facing up and down.
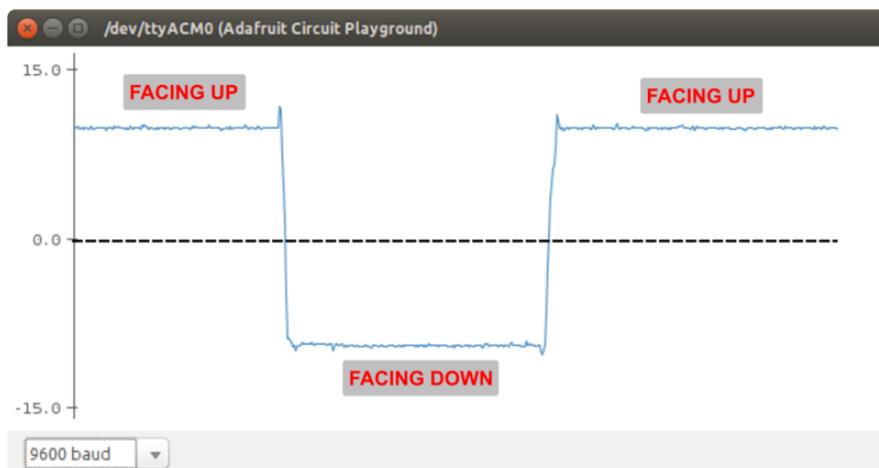
The value of 9.8 m/s^2 relates to earth gravity. With the Circuit Playground facing up, it is a positive value. With the Circuit Playground facing down, it is a negative value. So we can detect a flip by simply checking the sign of the value.

We start with the Circuit Playground face up and a positive value. The first step is to wait for the value to become negative. Then, once it is negative, we wait for it to become positive again. That's a flip!

Here's a simple sketch that demonstrates this logic being used to detect a flip and then play a tone.

```
# Circuit Playground Express Flip Detect
#
# Author: Carter Nelson
# MIT License (https://opensource.org/licenses/MIT)
import time
from adafruit_circuitplayground.express import cpx

while True:
    # Wait for Circuit Playground to be flipped over (face down)
    while cpx.acceleration[2] &gt; 0:
        pass # do nothing
```

```
    # A little debounce
    time.sleep(0.5)

    # Wait for Circuit Playground to be flipped back over (face up)
    while cpx.acceleration[2] &lt; 0:
        pass # do nothing

    # Make a beep noise.
    cpx.play_tone(800, 1)
```

Basic Hourglass

Let's add the flip detect logic from the previous section to the less basic timer code to make a basic hourglass.

Here's the code:

```
# Circuit Playground Express Basic Hourglass
#
# Author: Carter Nelson
# MIT License (https://opensource.org/licenses/MIT)
import time
from adafruit_circuitplayground.express import cpx

cpx.pixels.brightness = 0.2 # make less bright!

while True:
    # Turn ON all the NeoPixels
    cpx.pixels.fill((255, 255, 255))

    # Compute DT
    DT = 5 / 10

    # Turn OFF the NeoPixels one at a time, waiting DT each time
    for p in range(10):
        time.sleep(DT)
        cpx.pixels[p] = (0, 0, 0)

    # Wait for Circuit Playground to be flipped over (face down)
    while cpx.acceleration[2] &gt; 0:
        pass # do nothing

    # A little debounce
    time.sleep(0.5)

    # Wait for Circuit Playground to be flipped back over (face up)
    while cpx.acceleration[2] &lt; 0:
        pass # do nothing
```
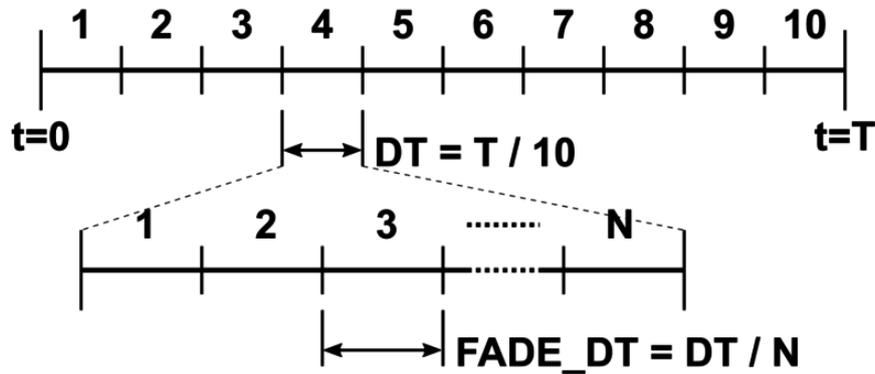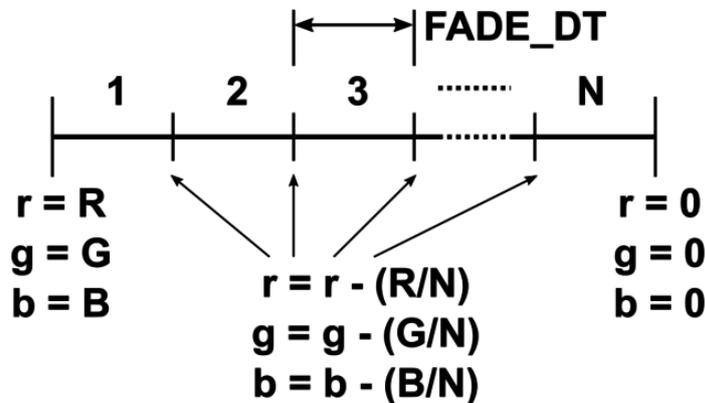
Fading Hourglass

Our previous code simply waited at each NeoPixel the computed amount of time DT, and then turned the NeoPixel off. Now let's change things so that the NeoPixel fades out over this period of time.

Basically, we need to go one level deeper on our time line. Now, within each time slice DT, instead of doing nothing, we'll be fading the NeoPixel. See the figure below.

The value N is just an arbitrary number of steps over which the fading will occur. This further divides the time slice DT into smaller slices, called FADE_DT in the figure above.

The simplest way to fade a NeoPixel is to linearly change the setting from its starting value down to zero. The NeoPixels on the Circuit Playground have red, green, and blue LEDs in them, so we do the same thing for each of the color components individually. This idea is shown in the figure below.

So at each of the fading time steps, we decrease the red, green, and blue values by a small amount. After doing this N times, the values will reach 0.

OK, here's our final code, with flip-to-reset and fading NeoPixel grains of sand.

```
# Circuit Playground Express Fading Hourglass
#
# Author: Carter Nelson
# MIT License (https://opensource.org/licenses/MIT)
import time
from adafruit_circuitplayground.express import cpx
```

```python
    # Make less bright (not blinding!)
    cpx.pixels.brightness = 0.2

    COUNT_TIME      = 30  # seconds
    FADE_STEPS      = 100 # NeoPixel fade steps
    R_SAND          = 255 # Sand color RED value
    G_SAND          = 255 # Sand color GREEN value
    B_SAND          = 255 # Sand color BLUE value

    # Compute per NeoPixel wait time
    DT = COUNT_TIME / 10

    # Copmute the color value change per fade steps
    dr = R_SAND / FADE_STEPS
    dg = G_SAND / FADE_STEPS
    db = B_SAND / FADE_STEPS

    while True:
        # Turn ON all the NeoPixels
        cpx.pixels.fill((R_SAND, G_SAND, B_SAND))

        # Loop over each NeoPixel
        for p in range(10):
            # Set the start RGB values
            r = R_SAND
            g = G_SAND
            b = B_SAND
            # Loop over each fading steps
            for n in range(FADE_STEPS):
                time.sleep(DT/FADE_STEPS)
                r = r - dr;
                g = g - dg;
                b = b - db;
                cpx.pixels[p] = (int(r),int(g),int(b))

        # Wait for Circuit Playground to be flipped over
        while cpx.acceleration[2] &gt; 0:
            pass

        # A little debounce
        time.sleep(0.5)

        # Wait for Circuit Playground to be flipped back over
        while cpx.acceleration[2] &lt; 0:
            pass
```

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.

Questions

- How would this code behave if the Circuit Playground had more than 10 NeoPixels? How about less?
- What is the maximum amount of time that the timer can be set for?
- Why is `delay()` or `time.sleep()` called before setting the NeoPixel colors in the loop?
- For the Arduino version, why were floats used in the Fading Hourglass sketch for the RGB values? (hint: what does `int value = 5/3;` give?) In CircuitPython, what's the difference between evaluating `5 / 3` and `5 // 3`?

Code Challenges

- Use `playTone()` or `cpx.play_tone()` to add an audio alarm when the countdown ends.
- Change the fade effect to be non-linear.
- Allow for a flip reset during the count period.