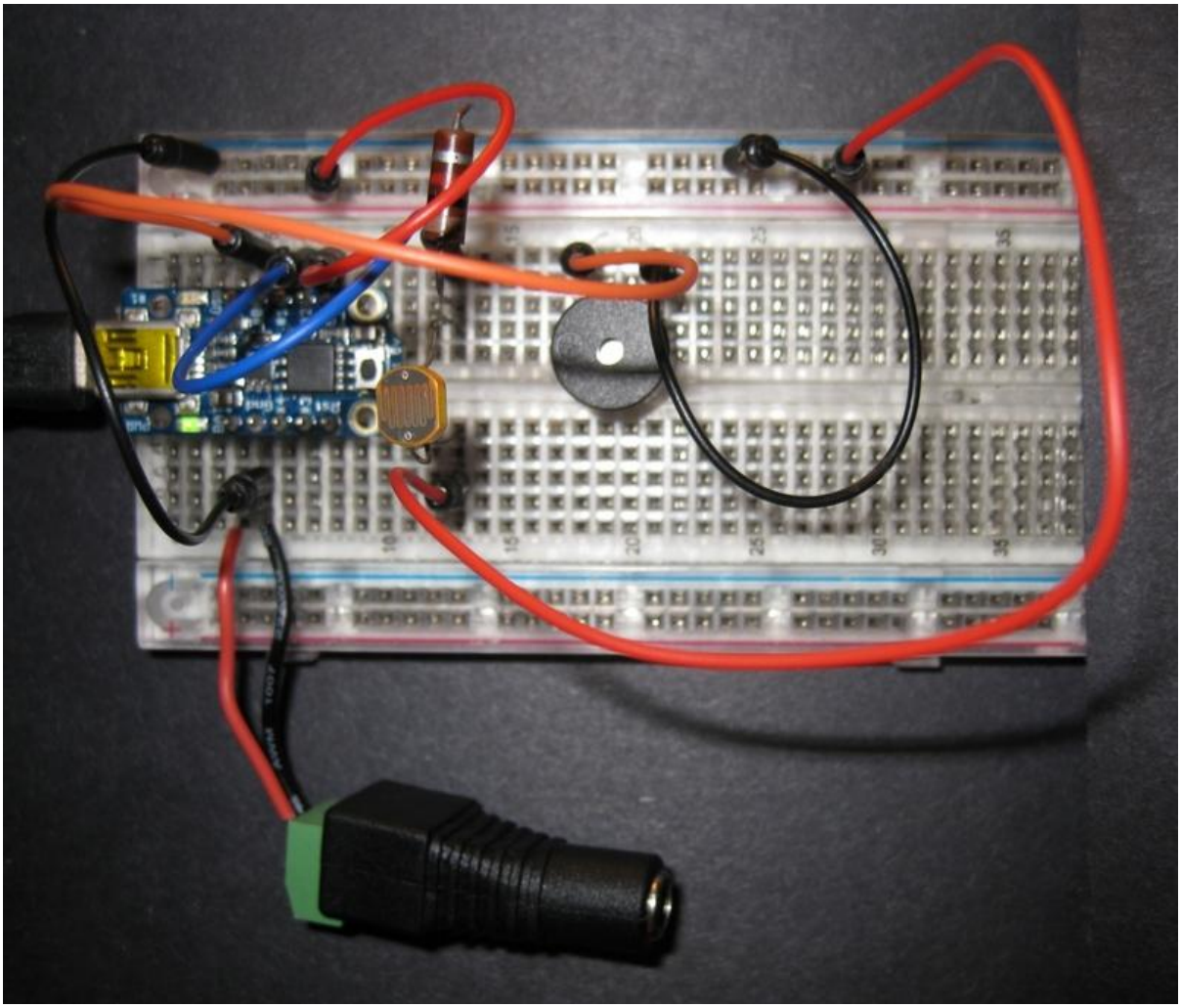
Trinket / Gemma Mini-Theramin

Created by Anne Barela

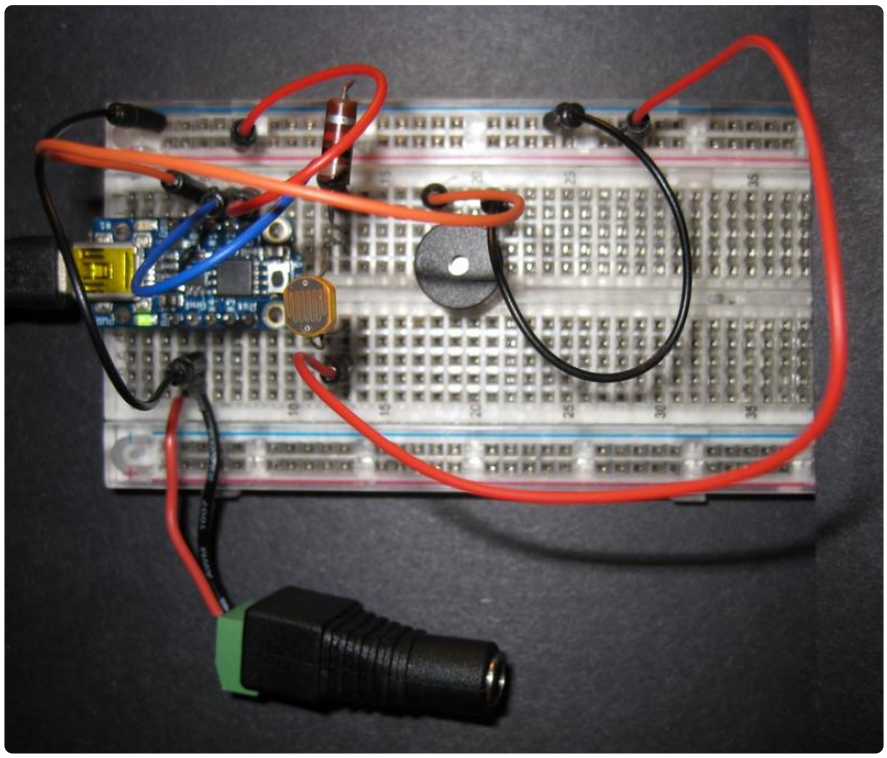


https://learn.adafruit.com/trinket-gemma-mini-theramin-music-maker

Last updated on 2022-12-01 02:03:05 PM EST

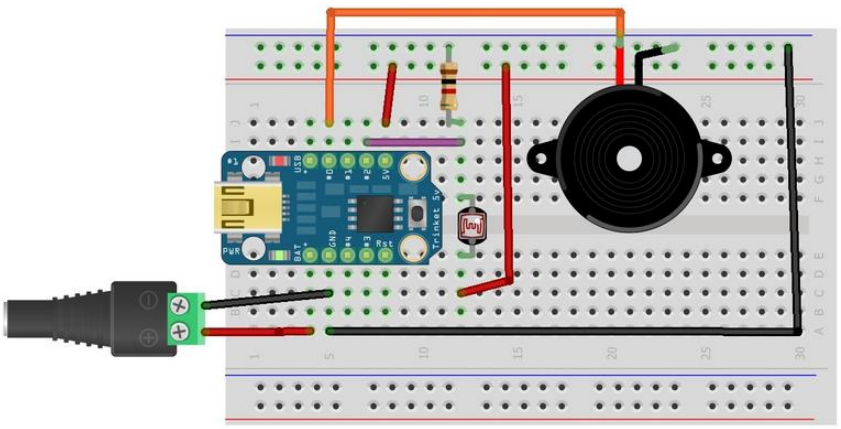
Table of Contents

Overview



In the Adafruit Arduino Lesson 10, Pseudo-Theramin (), an Arduino Uno is combined with a light-sensitive cadmium sulfide (CdS) photocell to make a light responsive music machine. This project allows you do do the same with an Adafruit Trinket or Gemma mini-microcontroller. This project costs less and works in smaller circuits or wearable projects.

Changes in light intensity on the photocell will change the pitch of a note on the piezo speaker as you wave your hand in front of the cell. While not a true theramin (which uses changes in a circuit's reactance), this project is much simpler to build.



This guide was written for the 'original' Trinket/GEMMA boards, but works with either the original or M0 Gemma. We recommend M0 boards as they're easier to use and more compatible with modern computers!

Wiring

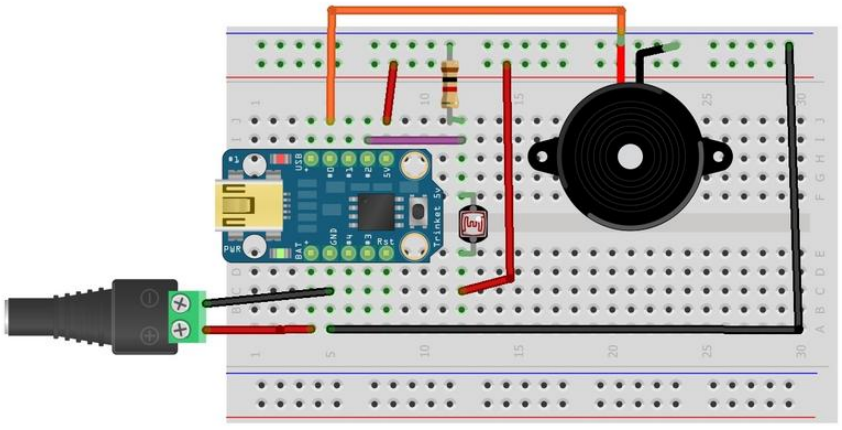
Schematic

If using a solderless breadboard, first solder pin headers on your Trinket board (they come included). For GEMMA you can use alligator jumpers. The parts you need are at right. You will need a resistor, nominally 1K to 10K ohms but this can vary a bit if you do not have that value, just do not make it too low or high.

Pins used:

- GPIO #0 (Digital 0) is connected to the signal pin (+) on the piezo speaker.
- GPIO #2 (Analog 1) is connected to the junction of the photocell and the resistor.

You can power the circuit from the USB connection or from an external supply connected to BAT (or Vout on GEMMA) as shown below (4 to 6 volts DC).



For a 3.3V Trinket or a Trinket M0, use the pin labeled '3V' instead of '5V.' For GEMMA (any variety), use the '3Vo' pin.

You should not hook a low impedance speaker in place of the piezo without additional circuitry. A speaker has a low DC resistance and may try to draw too much current from the microcontroller. A 100 ohm resistor in series may help, and it could be coupled with a 10 to 50 microfarad capacitor.

Arduino Code

The Arduino code presented below works equally well on all Trinket/GEMMA versions: v1, v2 and M0. But if you have an M0 board, consider using the CircuitPython code on the next page of this guide, no Arduino IDE required!

Here's the sketch, it's pretty simple.

- As more or less light hits the face of the CdS photo cell, the resistance changes
- When the resistance changes, the voltage on the analog pin will vary up and down
- We read the analog voltage from the analog pin
- We 'scale' the analog voltage to a frequency, then 'beep' the piezo buzzer to match
- ...and repeat!

Don't forget! You need to set the speaker pin as an output like in the setup function. While the beep waveform will appear on the pin without it, it will not drive the piezo correctly.

```
// SPDX-FileCopyrightText: 2017 Limor Fried/ladyada for Adafruit Industries
// SPDX-FileCopyrightText: 2017 Phillip Burgess for Adafruit Industries
//
// SPDX-License-Identifier: MIT
/* Adafruit Trinket/Gemma Example: Simple Theramin

  Read the voltage from a Cadmium Sulfide (CdS) photocell voltage
  divider and output a corresponding tone to a piezo buzzer

  Wiring: Photocell voltage divider center wire to GPIO #2
  (analog 1) and output tone to GPIO #0 (digital 0)

  Note: The Arduino tone library does not work for the ATTiny85 on the
  Trinket and Gemma.  The beep function below is similar.  The beep
  code is adapted from Dr. Leah Buechley at
  http://web.media.mit.edu/~leah/LilyPad/07_sound_code.html
*/

#define SPEAKER   0    // Speaker connected to this DIGITAL pin #
#define PHOTOCELL 1    // CdS photocell connected to this ANALOG pin #
#define SCALE     2.0  // You can change this to change the tone scale

void setup() {
  // Set SPEAKER pin to output to drive the piezo buzzer (important)
  pinMode(SPEAKER, OUTPUT);
}

void loop() {
  // Read PHOTOCELL analog pin for the current CdS divider voltage
  int reading = analogRead(PHOTOCELL);
  // Change the voltage to a frequency.  You can change the values
  // to scale your frequency range.
```

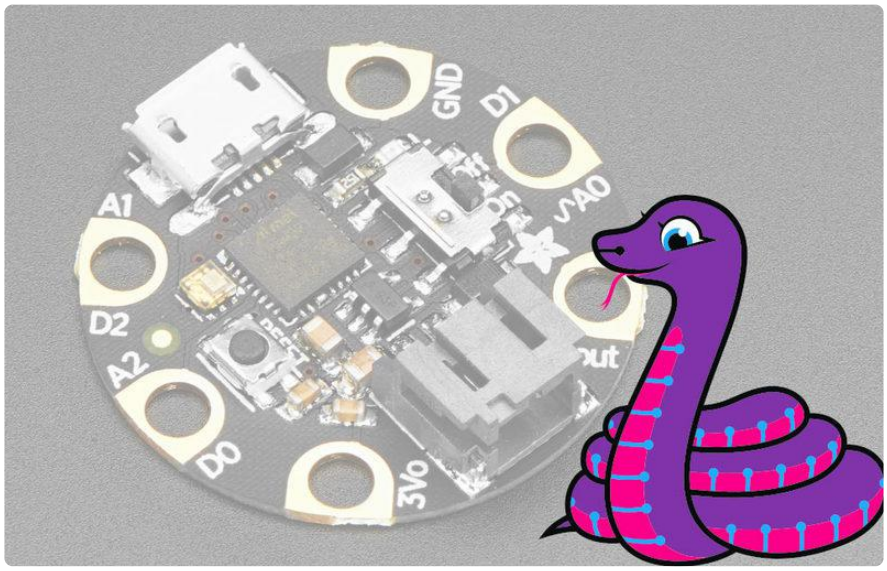
```
  int freq = 220 + (int)(reading * SCALE);
  // Output the tone to SPEAKER pin.  You can change the '400'
  // to different times (in milliseconds).
  beep(SPEAKER, freq, 400);
  delay(50); // Delay a bit between notes (also adjustable to taste)
}

// The sound-producing function
void beep (unsigned char speakerPin, int frequencyInHertz, long timeInMilliseconds)
{ // http://web.media.mit.edu/~leah/LilyPad/07_sound_code.html
  int  x;
  long delayAmount = (long)(1000000 / frequencyInHertz);
  long loopTime = (long)((timeInMilliseconds * 1000) / (delayAmount * 2));
  for (x = 0; x < loopTime; x++) {
    digitalWrite(speakerPin, HIGH);
    delayMicroseconds(delayAmount);
    digitalWrite(speakerPin, LOW);
    delayMicroseconds(delayAmount);
  }
}
```

You will note the tone function used in the Uno lesson is not used. The Arduino IDE does not implement tone for the ATtiny85 used for the Trinket and Gemma.

The beep function in the code above will work in a similar fashion. It does not use the hardware pulse width modulation on the microcontroller, so you may use it to output a tone signal on any digital pin you have free.

CircuitPython Code



Trinket M0 and GEMMA M0 boards can run CircuitPython — a different approach to programming compared to Arduino sketches. In fact, CircuitPython comes factory pre-loaded on these boards. If you've overwritten it with an Arduino sketch, or just want to learn the basics of setting up and using CircuitPython, this is explained in the Adafruit Trinket M0 () and Adafruit GEMMA M0 () guides.

These directions are specific to the “M0” boards. The original Trinket & GEMMA with an 8-bit AVR microcontroller don’t run CircuitPython...for those boards, use the Arduino sketch on the “Arduino code” page of this guide.

Below is CircuitPython code that works similarly to the Arduino sketch shown on the prior page. To use this, plug the Trinket/GEMMA M0 into USB...it should show up on your computer as a small flash drive...then edit the file “main.py” with your text editor of choice. Select and copy the code below and paste it into that file, entirely replacing its contents (don’t mix it in with lingering bits of old code). When you save the file, the code should start running almost immediately (if not, see notes at the bottom of this page).

If Trinket or GEMMA M0 doesn’t show up as a drive, follow the corresponding guide link above to prepare the board for CircuitPython.

This code requires version 2.1 or later of CircuitPython. Earlier versions didn’t yet support PWM output. The Trinket M0 or Gemma M0 introductory guides explain how to load or update CircuitPython if needed.

```
# SPDX-FileCopyrightText: 2017 Limor Fried/ladyada for Adafruit Industries
# SPDX-FileCopyrightText: 2017 Phillip Burgess for Adafruit Industries
#
# SPDX-License-Identifier: MIT

"""
Adafruit Trinket/Gemma Example: Simple Theramin
Read the voltage from a Cadmium Sulfide (CdS) photocell voltage
divider and output a corresponding tone to a piezo buzzer

Photocell voltage divider center wire to GPIO #2 (analog 1)
and output tone to GPIO #0 (digital 0)
"""

import time

import analogio
import board
import pwmio

photocell_pin = board.A1  # CdS photocell connected to this ANALOG pin
speaker_pin = board.D0  # Speaker is connected to this DIGITAL pin
scale = 0.03  # Change this to adjust tone scale

# Initialize input/output pins
photocell = analogio.AnalogIn(photocell_pin)
pwm = pwmio.PWMOut(speaker_pin, variable_frequency=True, duty_cycle=0)

while True:  # Loop forever...
    # Read photocell analog pin and convert voltage to frequency
    pwm.frequency = 220 + int(scale * float(photocell.value))
    pwm.duty_cycle = 32767  # 50% duty cycle
    time.sleep(0.4)  # Play for 400 ms (adjust to your liking)
    pwm.duty_cycle = 0  # Stop playing
    time.sleep(0.05)  # Delay 50 ms between notes (also adjustable)
```

Demonstration and Going Further

Here is the circuit in action:

Going Further

You may change either the SCALE variable or even the entire calculation of the frequency freq. There is nothing you can do to "break" the sound values generated to feel free to vary the numbers generated, the time the tone is generated, and the delay between tones.

You can generate precise notes by selecting the correct frequency. Which notes correspond to which frequencies may be found at http://en.wikipedia.org/wiki/Mathematics_of_musical_scales ()