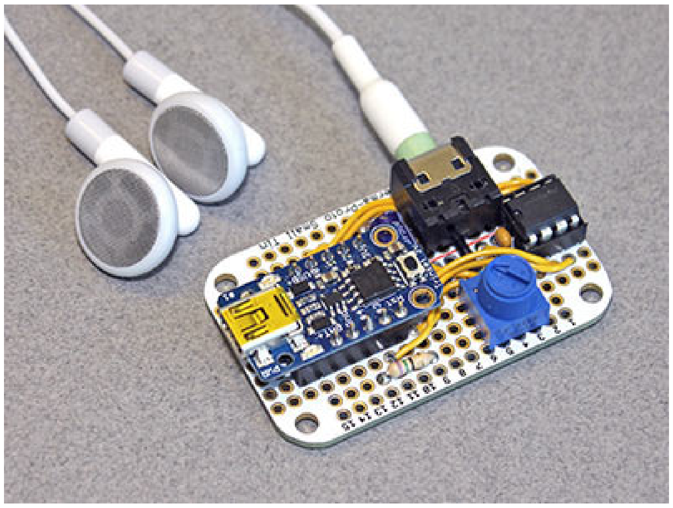
adafruit learning system

Trinket Audio Player

Created by Phillip Burgess



Last updated on 2015-09-29 06:19:06 PM EDT

Guide Contents

Guide Contents 2

Overview 3

Parts Needed 3

- For the “loading” stage: 4
- For the “playback” stage: 4

Software Needed 4

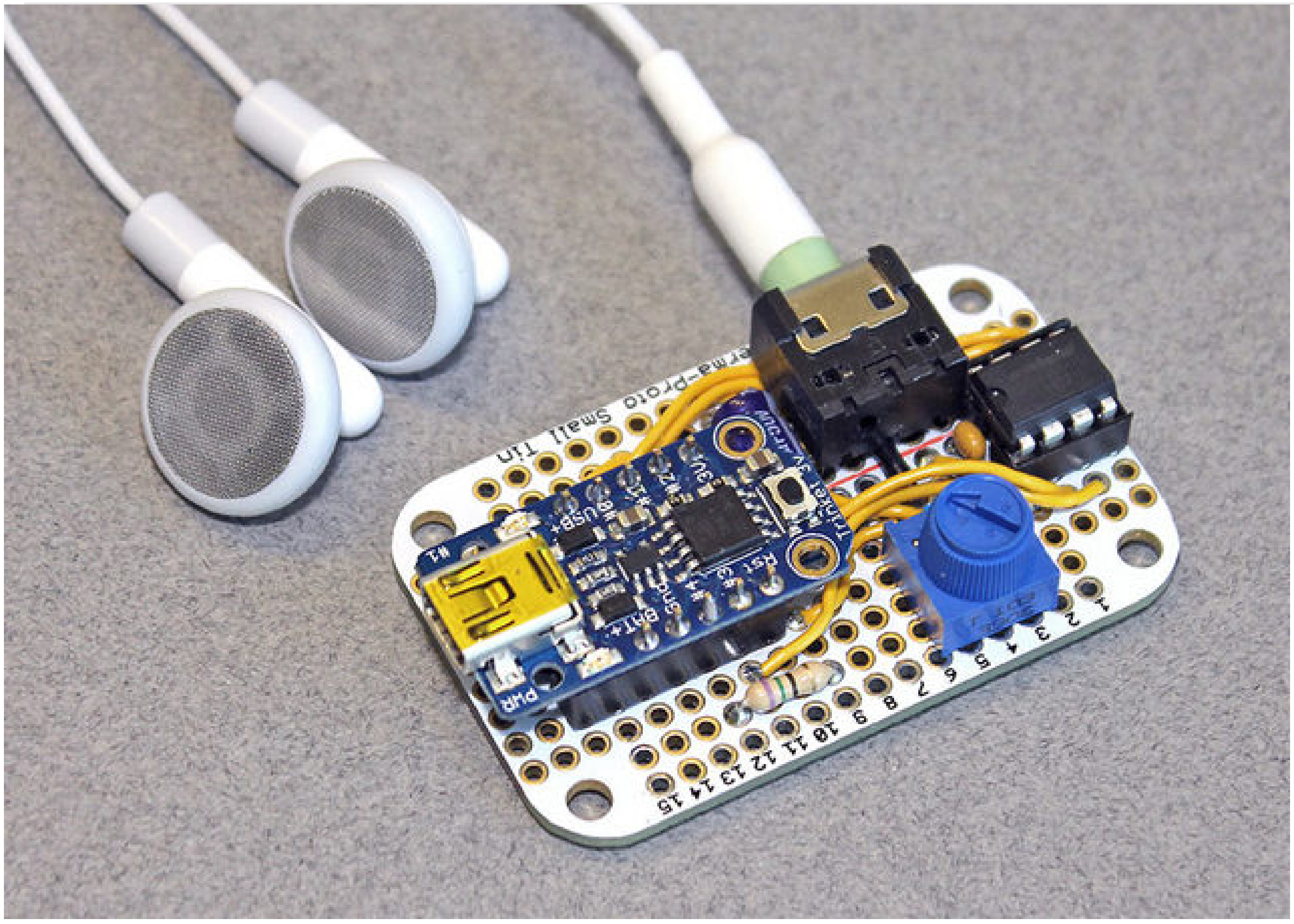
Loading Sounds 6

Chip Loading Circuit 7

Transferring Audio 9

Sound Playback 11

Overview



We usually think of the Adafruit Trinket as a tiny subset of a “real” Arduino; less RAM, less code space, less I/O. But this little chip has a couple tricks up its sleeve, things its larger brethren can’t do. One of these is a high-speed PWM mode. With just a few extra components this can be used for audio output. Not simply piezo beeps and buzzes…actual sampled digital sound!

You could make an electronic greeting card with your own customized message or song, add a background soundtrack to a model train diorama, or create the world’s smartest whoopee cushion.

So will this play MP3s and stuff?

No. It’s a very simple circuit that just plays a short “raw” audio loop. For more sophisticated audio projects, check out our VS1053 Codec board (http://adafru.it/1381).

Parts Needed

There are two phases to this project. The first (using a regular Arduino) loads sound data onto a flash memory chip, the second (using Trinket) plays it back. Both stages use some parts in common:

- **Winbond 25Q80BV (http://adafru.it/1564)** serial flash memory (1 megabyte DIP). This can store about one minute of music or two minutes of voice, depending on quality.
  Note this chip is discontinued! If you use a different chip you may have to modify your code, see here for an example. (http://adafru.it/ikE)
- **Breadboard (http://adafru.it/64)(s)** and **jumper wires (http://adafru.it/153)**, (http://adafru.it/153) or **proto boards (http://adafru.it/571)** and related soldering paraphenalia

For the “loading” stage:

- **Arduino Uno (http://adafru.it/50)** or similar board
- **Capacitor (http://adafru.it/753):** one 0.1 µF
- **Resistors:** 3 each 470 Ohm and 1K Ohm
- You can optionally add an **LED (http://adafru.it/299)** (any color) and **220 Ohm resistor** for a status indicator

Not all of these parts are available from Adafruit. You may be able to swap out for different parts you already have on-hand or can acquire locally; the “Loading Sounds” page has some guidance on alternative parts selection.

For the “playback” stage:

- **Adafruit Trinket 3.3V (http://adafru.it/1500)** (NOT 5V!)
- **Capacitors:** one 10 µF, 2 each 0.1 µF (http://adafru.it/753)
- **Resistor:** one 68 Ohm
- **10K potentiometer (http://adafru.it/356)**
- **Battery** (or can use **USB power**)

and either:

- **Headphone jack** and **headphones** or (http://adafru.it/1363)**portable amplified speaker (http://adafru.it/1363)**

or:

- **Audio amplifier board (http://adafru.it/1552)** and **4 Ohm speaker (or 8 ohm) (http://adafru.it/1314)**

The “Sound Playback” page likewise has some guidance on alternative parts selection. There’s a lot of wiggle room, not everything needs to be a super-precise value.

Software Needed

You'll need sound files in WAV format. You can search for downloadable examples on the internet (movie quotes, cartoon sounds, etc.), or record or convert something from your music collection using software such as **Audacity** (http://adafru.it/c9T) (free download).

This project uses both **Processing** and the **Arduino IDE.** Both look very similar when running, which can lead to confusion, so make sure you're loading the right code in the right editor! **Processing** is for writing code to run on your computer, while **Arduino** is for writing microcontroller code.

Download version 2.0 (or later) of Processing from (http://adafru.it/aPt)**processing.org (http://adafru.it/aPt)** (our software won't work with the 1.5 version, if you currently have that installed).

If this is your first time using Trinket, work through the Introducing Trinket (http://adafru.it/cEu) guide first; you need to install and then customize some settings in the Arduino IDE. Once you have it up and running (test the "blink" sketch), then download and install the TinyFlash library:

Click to download the TinyFlash library

http://adafru.it/cQ7

Installing Arduino libraries is a frequent stumbling block. If this is your first time, or simply needing a refresher, please read the All About Arduino Libraries (http://adafru.it/aYM) tutorial.

When properly installed, in the Arduino IDE you should then have access the rollover menu **File→Sketchbook→Libraries→Adafruit_TinyFlash**

The "examples" folder included with the library contains all the code for this project; there's nothing else to download.

Why this weird flash memory chip? Why not an SD card?

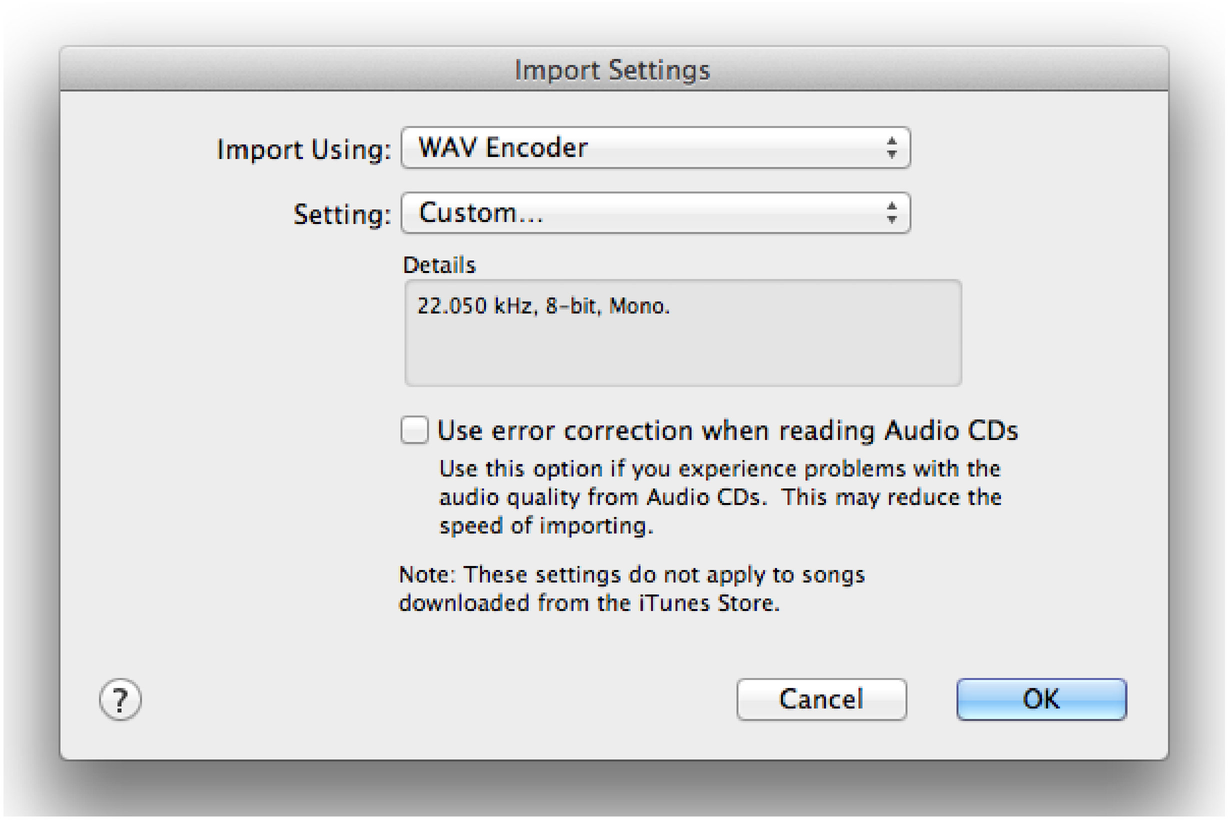
Good question! There are a couple of reasons:

- The flash chip is super affordable, so you can make it a permanent part of a small project. As it's in DIP chip form, you don't need to buy a special breakout board like you would for an SD card.
- Reading a FAT-formatted SD card with this tiny microcontroller is incredibly difficult; a single SD block would fill the chip's *entire* RAM! We have seen projects that do this, so it's not impossible but is nonetheless quite challenging. Perhaps we'll revisit this idea in the future.

Loading Sounds

Sound files for this project need to be in WAV format, uncompressed (PCM), 8- or 16-bit resolution. Mono, stereo or multi-channel are all acceptable…the software we'll use in a moment will automatically convert to 8-bit mono if needed.

If your audio is in a different format, you can convert it with a tool like Audacity (http://adafru.it/c9T) (free), Adobe Audition ($$$) or you may already have a utility on your computer that can produce something. Even iTunes can do this, if you tweak the import settings:



Import Using: WAV Encoder

Setting: Custom…

Details

22.050 kHz, 8-bit, Mono.

☐ Use error correction when reading Audio CDs

Use this option if you experience problems with the audio quality from Audio CDs. This may reduce the speed of importing.

Note: These settings do not apply to songs downloaded from the iTunes Store.

For voice recordings, 8 KHz is often a sufficient sample rate. For music, 16 KHz or more. Generally, higher sampling rates will produce better-sounding audio, but it requires more space. Also, the way the playback circuit works, there's diminishing returns above 25 KHz. Experiment!

The Winbond flash chip we're using has a capacity of 1,048,576 bytes (1 megabyte, often called "8 megabit" because marketing). Six bytes are used to store data about the length and sampling rate of the audio, leaving 1,048,570 bytes for the audio data itself. Each byte is one audio sample.

To estimate the maximum duration of audio you can store on the chip:

> **Max. duration (in seconds) = 1,048,570 ÷ sampling rate**

e.g. with 16,000 Hz (16 KHz) music:

> **1,048,570 ÷ 16,000 = ~65.5 seconds**

If your source audio file is too big for the available space, the end will be truncated to fit.

Chip Loading Circuit

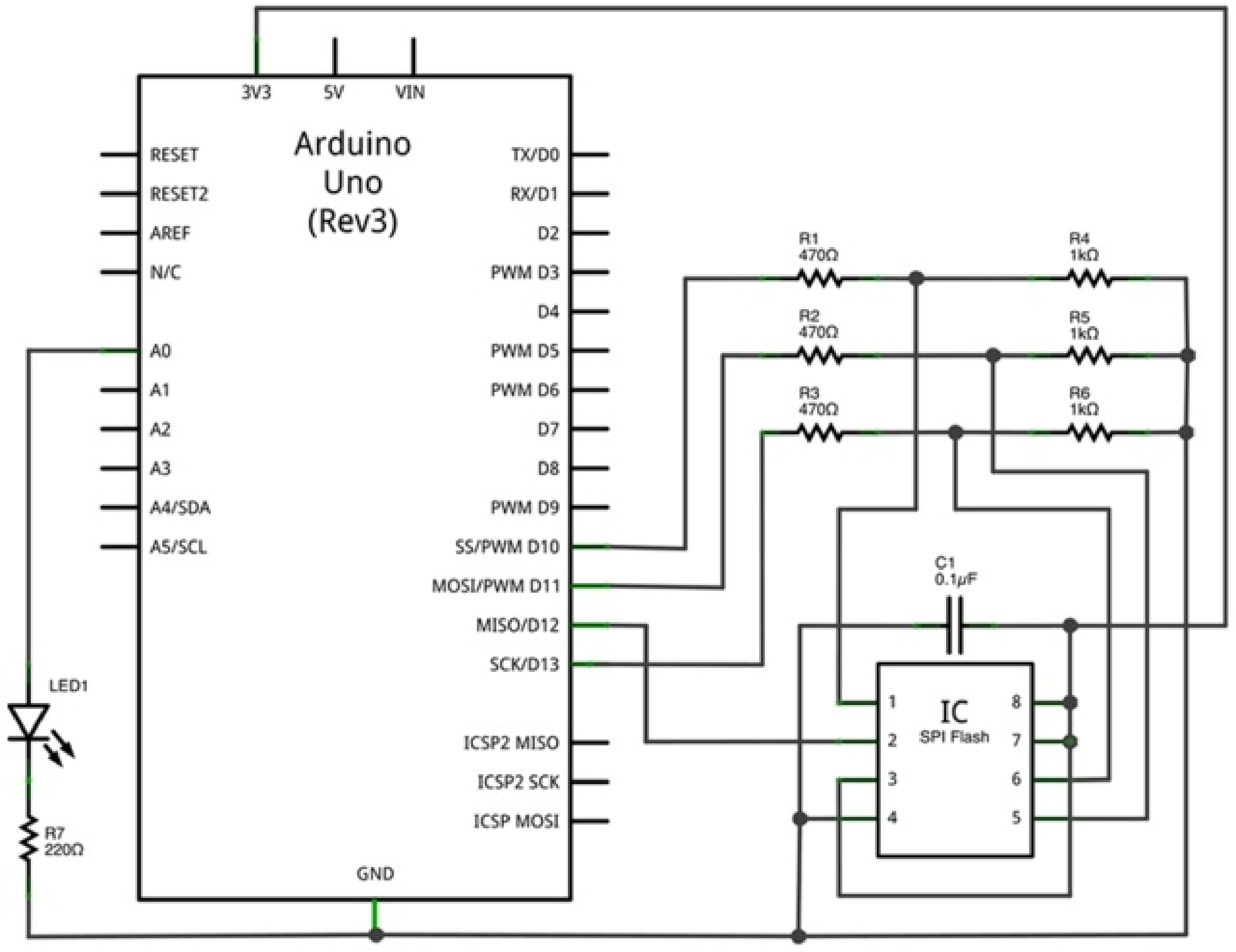
Because the Trinket can’t support traditional serial I/O, we’ll make temporary use of a regular Arduino board to help transfer data to the Winbond chip. Later we move it to a playback circuit.

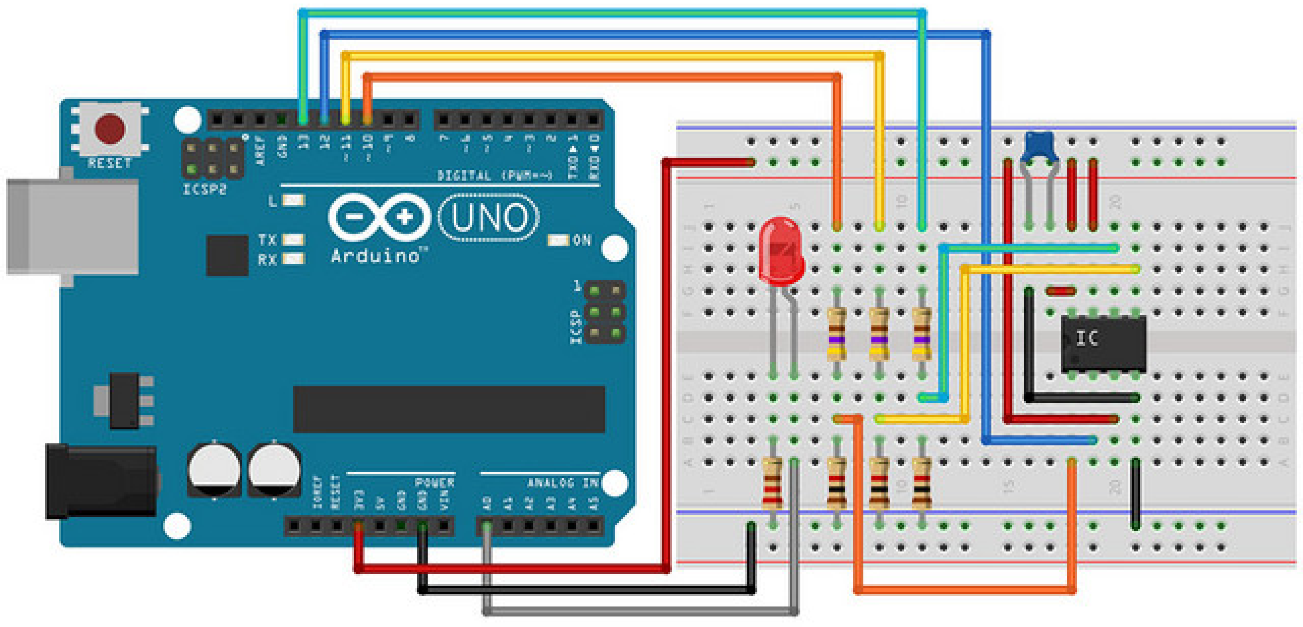
We suggest using an Arduino Uno, because it has easy access to the SPI pins used to communicate with the flash chip. This *can* be done with a Leonardo, Mega or other board, but you’ll need to adapt the wiring to use the 6-pin ICSP header (MISO, MOSI and SCK pins, specifically).

The serial flash chip is a 3.3 Volt device, whereas the Arduino is 5 Volts. To avoid unleashing the Blue Smoke Monster (http://adafru.it/691), it’s necessary to power the chip from the Arduino’s 3.3V (not 5V) pin, and then use level-shifting circuitry to drive the control lines. There are chips that do this (http://adafru.it/735); use ’em if you got ’em, or a simple approximation can be made using resistors to create a voltage divider: output signal pins from the Arduino each connect to a 470 Ohm resistor, the other end connected both to an input pin on the flash chip and a 1K resistor to ground (see diagrams below). If you don’t have *exactly* these values of resistors on-hand, that’s okay, it’s not rocket surgery…you can substitute other values with approximately a 1:2 ratio, for example 1K and 2.2K (or use two 1K resistors in series for the latter). 1K/2.2K is about the upper limit on values; don’t go higher than this. Won’t harm anything, just won’t work very reliably. Also, note that the data output line back into the Arduino (pin 12, blue wire) connects directly, there are no resistors on this line.

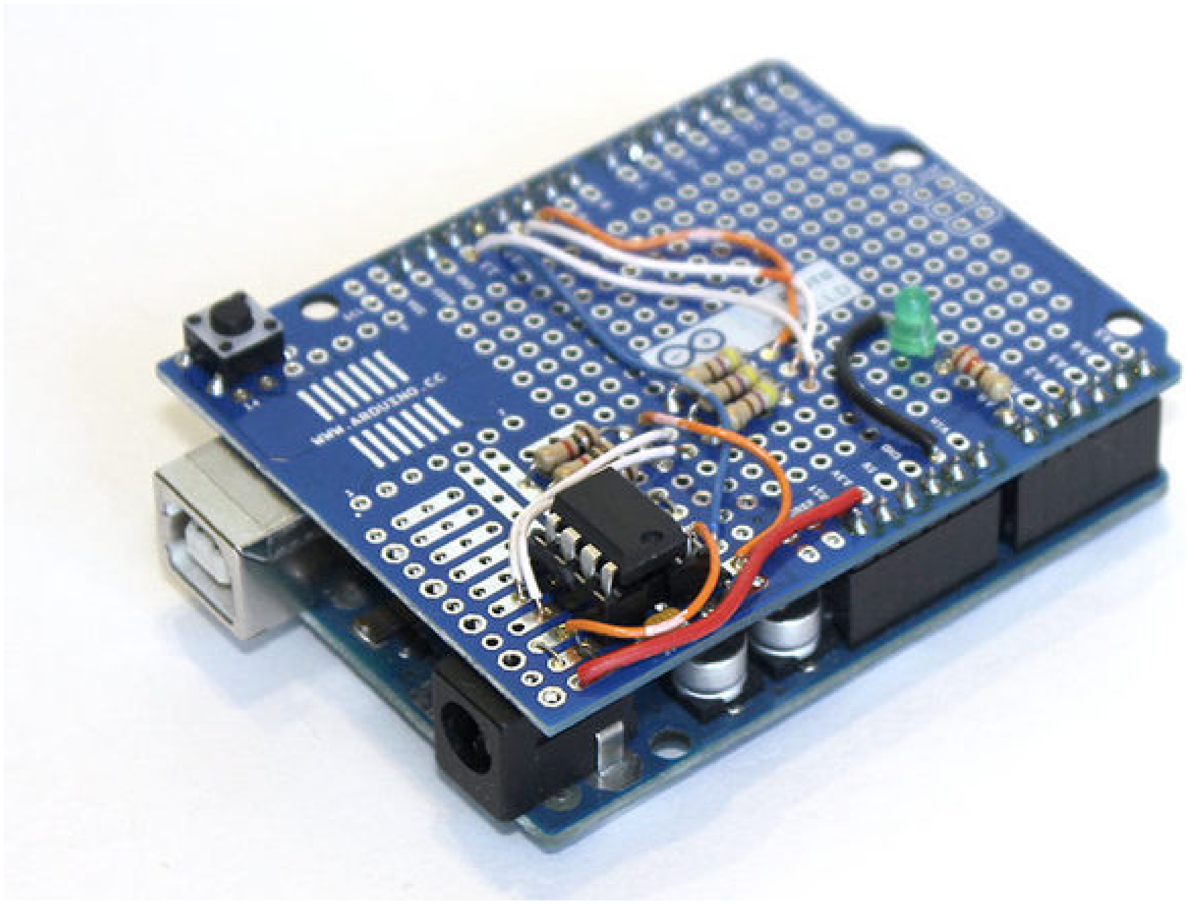
An LED and 220 Ohm resistor can be added between pin A0 and GND to provide a simple status display. The LED will blink to indicate an error, or flickers during the data transfer. This is not essential and can be left out of the circuit if you don’t have the parts.

Finally, there’s a 0.1 µF capacitor between the flash chip’s VCC supply and GND. In a pinch you can get by without this, but it’s good form to have it there, keeps electrical gremlins away.





Breadboarding works fine for occasional use. Knowing I'd be programming *a whole lot* of chips while debugging this project, I put all the components on an Arduino proto shield, with a socket so the flash chip can be easily swapped out:



Now launch the Arduino IDE and load the AudioLoader sketch:

File→Sketchbook→Libraries→Adafruit_TinyFlash→AudioLoader

Select the board type and serial port from the **Tools** menu, and upload this code to the board. If you don't have the flash memory chip installed (or if it's positioned incorrectly, or turned around), the status LED should blink. If it's working properly, you won't see anything from the LED, but you can check the serial monitor (at 57600 baud). It should display:

```
HELLO
1048576
```

If not, something may be amiss with your wiring. Double-check all the connections against the schematic.

Transferring Audio

Close the Arduino serial monitor if you still have it open; the other code won't work if it's there.

Launch Processing and load the AudioXfer sketch. It's inside the Adafruit_TinyFlash folder that you downloaded earlier, in a sub-folder called "Processing." Sorry to make you hunt through the Arduino folder for this, but it was less troublesome that requiring a separate download.

Processing and the Arduino IDE look very similar. If you encounter strange errors, make sure you're loading the right code in the right environment.

When you run the AudioXfer sketch, all serial ports on the system are scanned until an Arduino running the AudioLoader sketch is identified. If you already know the name of the port (previously selected from the **Tools→Serial Port** menu in the Arduino IDE), there's a line you can uncomment to open this port directly and bypass the whole lengthy port scan:

```
portname = "/dev/tty.usbmodem1a1331"; // bypass scan
```

(That's just an example port name one might see on Mac. In Windows it might be something like "COM6".)

If the software detects an Arduino running the AudioLoader sketch, and if it reports a flash memory chip is connected, you'll be prompted to select a WAV file to transfer. **When you select a WAV file and click "Open," the chip is erased at that point. There is no undo.**

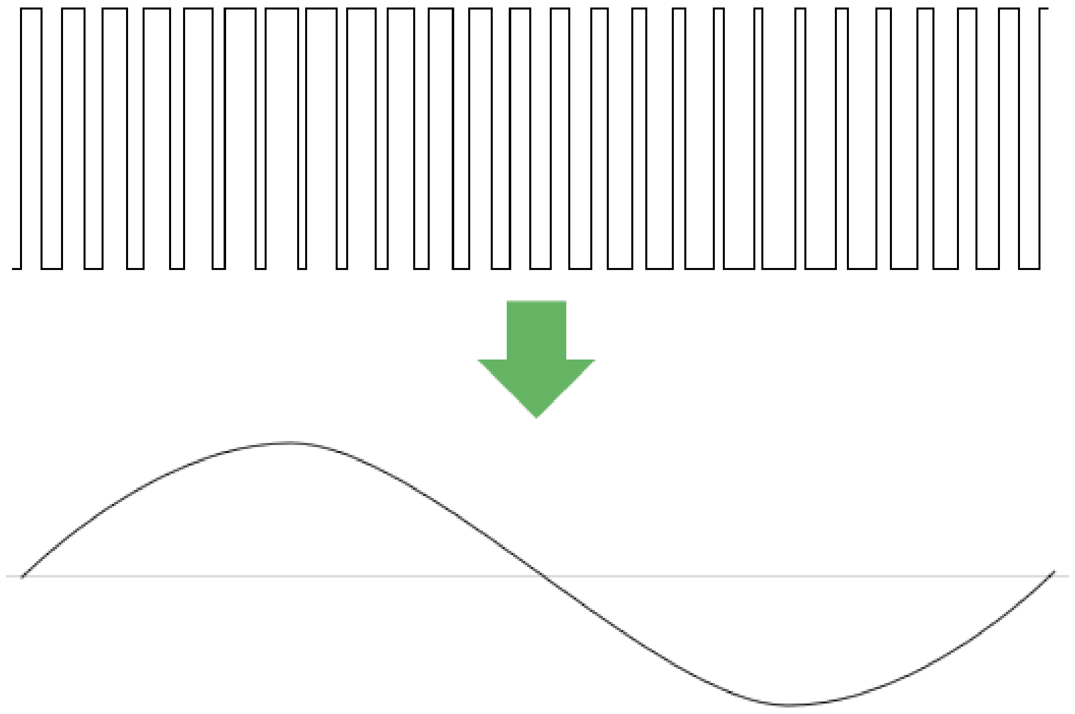
AudioXfer is a pretty barebones program…other than the file selection, there is no fancy GUI, it just prints text to the console. You'll see a long line of dots (and the LED will flicker) as data is transferred to the Arduino and written to the chip. It can take several minutes to load a 1 megabyte chip to capacity. So you may want to test with just a short sound at first.

If all goes well, the software reports "done!" If an error was encountered, you'll instead see a message with some indication of the problem.

Once a sound is successfully loaded, disconnect the Arduino from USB, remove the flash memory chip from the breadboard or socket, and then we'll move it over to the playback circuit…

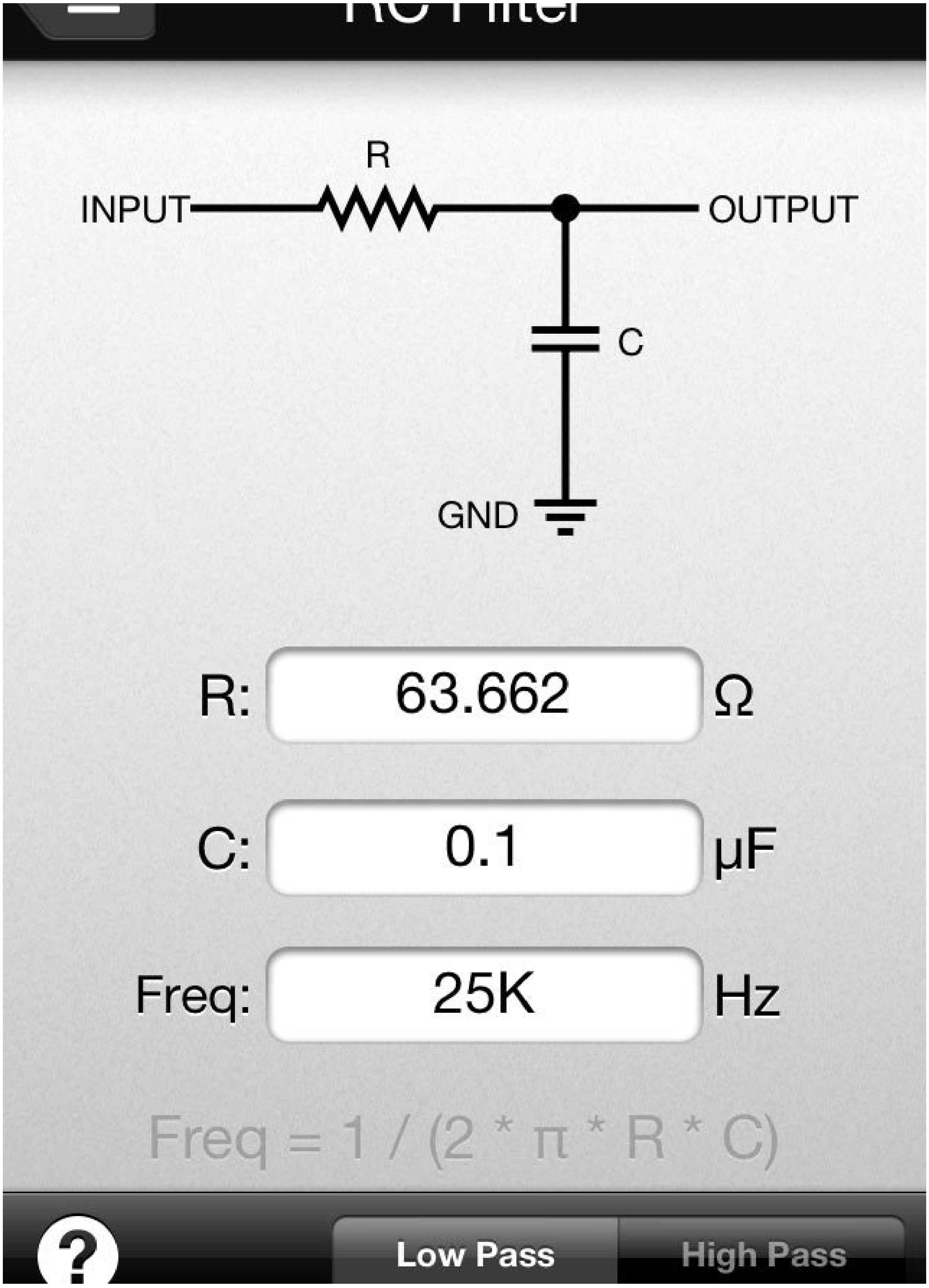
Sound Playback

The ATtiny85 chip at the heart of Trinket has the novel ability to produce a 250 KHz 8-bit PWM signal. That's four times what the Arduino Uno can muster. A low pass filter circuit then smooths the "square" PWM into a usable audio waveform:



A very basic low pass filter can be made from just a capacitor and resistor. First we need to know the filter's *cutoff frequency* — frequencies below this pass through, while higher frequencies (like the PWM signal) are *attenuated.* A rule of thumb with PWM audio is that the highest usable audio frequency (our cutoff frequency) is about 1/10 the PWM rate. The latter we've already established is 250 KHz, so a good cutoff would be 25 KHz.

There's a relationship between the capacitor and resistor values and the resulting cutoff frequency. Given any two values, we can compute the third. Already having a ton of 0.1 microfarad capacitors around, I just needed to know the corresponding resistor to achieve the desired 25 KHz cutoff (you could also do it the other way — some resistor you have around, determining a suitable capacitor). Rather than bore you with the math, you can just whip out the Circuit Playground (http://adafru.it/cQb) app for iOS (select "Circuit Calculators," then "RC Cutoff Filter," or you can Google search for "low pass filter calculator" resources on the web and plug in the two known values you have.



R: 63.662 Ω

C: 0.1 µF

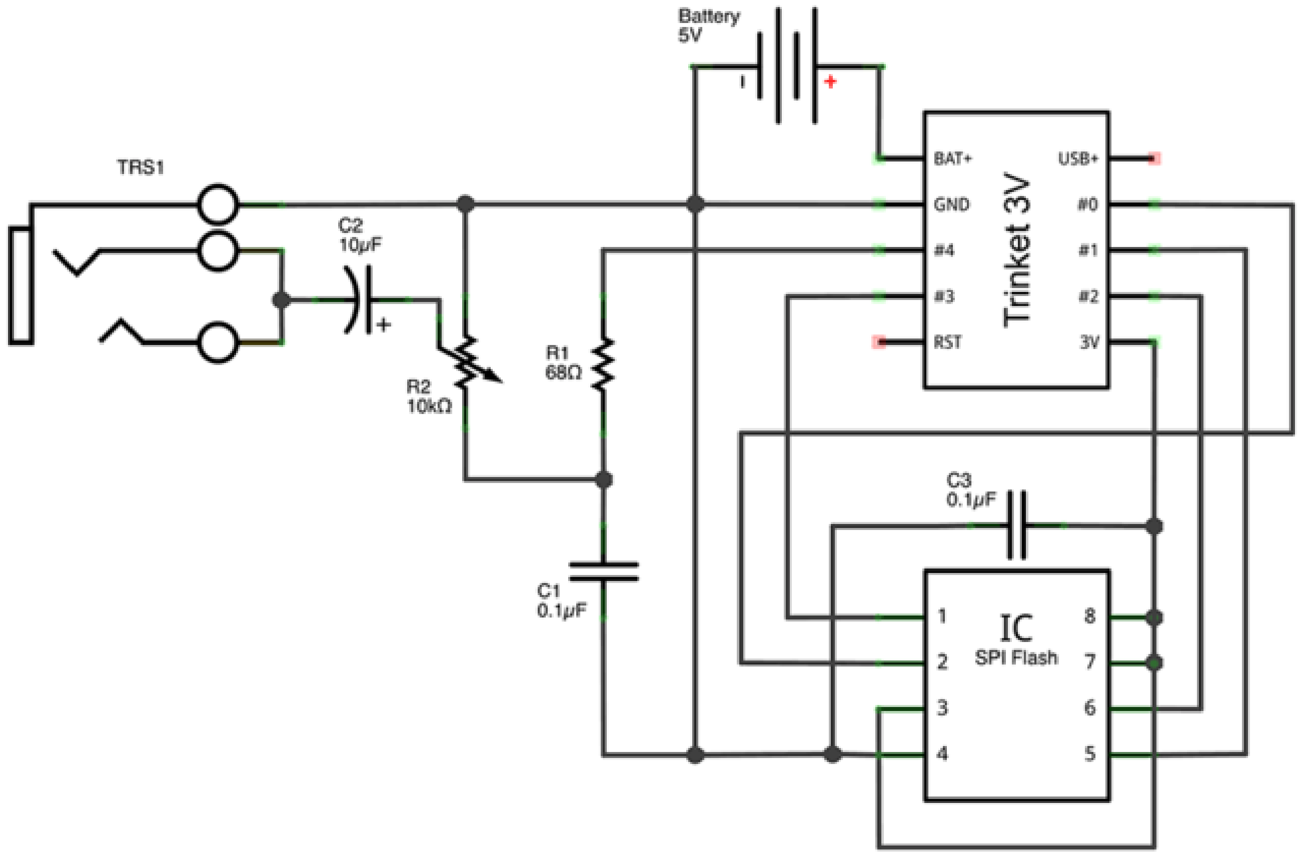
Freq: 25K Hz

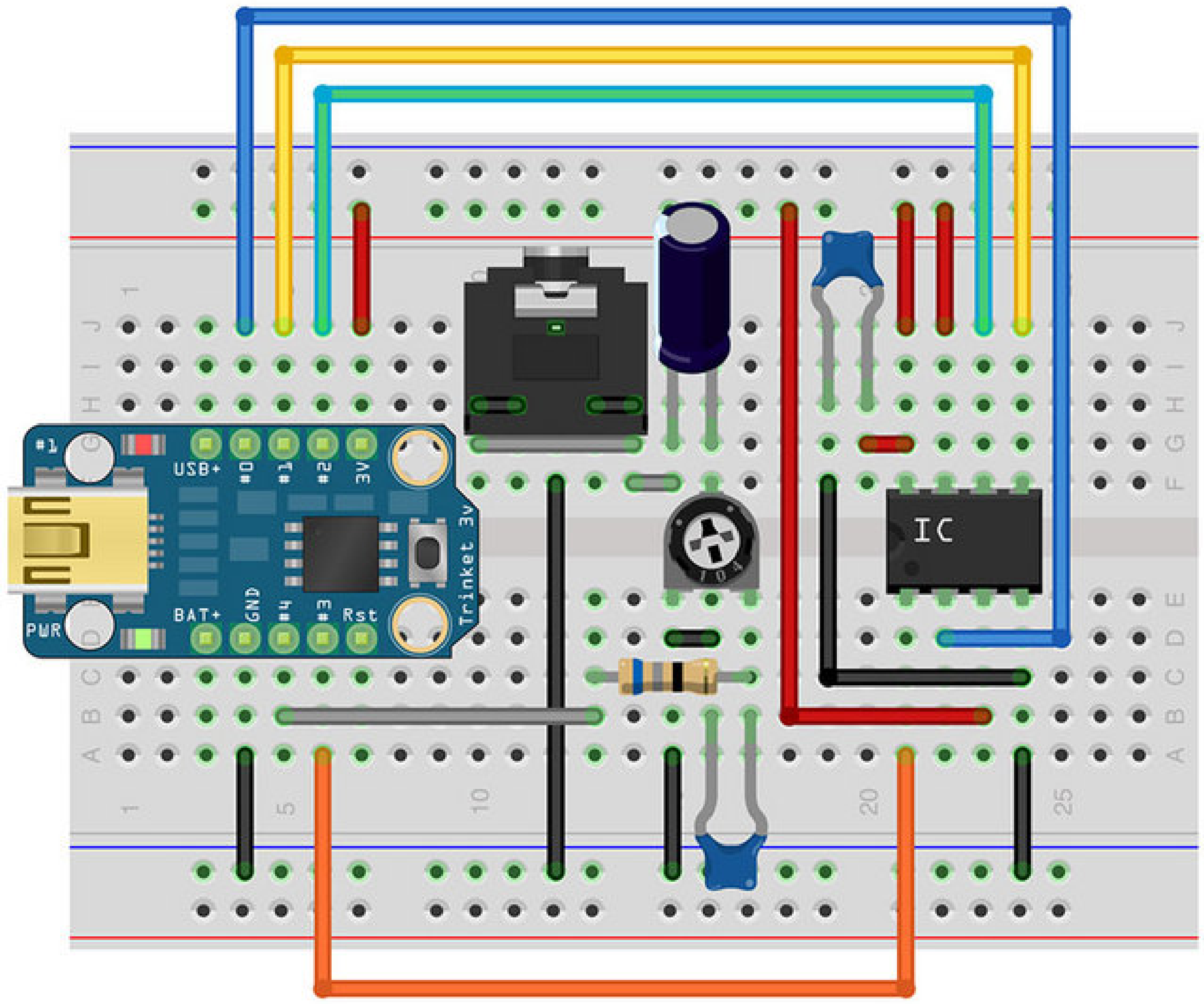
Freq = 1 / (2 * π * R * C)

For a 25 KHz cutoff and 0.1 microfarad cap, the calculator suggests a 63 Ohm resistor. That’s not a standard value you’ll find anywhere, so we just select the next common size up from there: **68 Ohms**. I happened to have a bunch of 75 Ohm resistors around…that’s close enough, this isn’t precision work.

The filtered output is then fed into a **10K potentiometer** for volume adjustment (you can leave this part out, but the volume will always be at the maximum) and then through a **10 microfarad** capacitor that provides “AC coupling” — so the audio waveform is centered at 0 Volts rather than 1.65V (one half the Trinket’s operating voltage). The output is split to both the right and left channels of a 1/8" phono jack, to which headphones or an amplified speaker can be connected.

It’s very important that a **3.3V Trinket** is used; this voltage is directly compatible with the flash memory chip. Adding the level shifting circuitry would defeat the smallness of the Trinket.





The power source can be anything the Trinket can handle: a small 3.7V LiPo cell, three or four AA or AAA alkaline cells, etc. You can also plug into USB, but there's a 10 second timeout while the bootloader runs its course before the playback sketch runs.

Trinket Pin #4 (audio output) MUST BE DISCONNECTED before code can be uploaded to the board! If you solder the circuit permanently in a proto board, it's strongly recommended that the Trinket be socketed, or add a jumper between pin 4 and the RC filter so it can be disconnected when new code is uploaded to the chip.

The audio connection interferes with USB. Disconnect pin 4 or unplug Trinket from the breadboard before uploading code. Then reconnect afterward.

Load the sound playback sketch in the Arduino IDE:

File→Sketchbook→Libraries→Adafruit_TinyFlash→TrinketPlayer

Select “Adafruit Trinket 8 MHz” from the **Tools→Board** menu. Disconnect pin 4 (or remove Trinket from the circuit), press the reset button, then click Upload. After uploading, assuming all else is wired properly, your audio should start playing immediately. As the code is currently written, the sound will loop forever. You could change this to stop after the music plays, then use the reset button to restart.

This code only works on the Trinket. It uses special registers and will not compile on the Uno or other Arduino boards.