Trinket “Question Block” Sound Jewelry

Created by Phillip Burgess

https://learn.adafruit.com/trinket-question-block-sound-jewelry

Last updated on 2021-11-15 06:19:31 PM EST
# Table of Contents

## Overview
- Prerequisite Guides 3
- Parts 4
- Tools & Supplies 4

## 3D Printing
- FDM 3D Printing 5
- Light Diffusion 5
- Slicing Software 6
- PLA or ABS Material 6

## Assembly
- Trinket Mini 5v Circuit Diagram 8
- Trinket M0 Circuit Diagram 8
- Measure Wires 9
- Prepare LiPoly backpack 9
- Solder LiPoly wires 10
- Prepare Vibration sensor 10
- Align Vibration sensor to Trinket 11
- Vibration sensor thinner pin 11
- LiPoly backpack wires 12
- LiPoly ground 12
- Solder LEDs 13
- Trinket tac 14
- Light pollution 14
- Mount Piezo 15
- Coil piezo wires 16
- Mount Trinket 17
- Insert slide switch 17
- Diffuser 17
- Position LEDs 18
- Wear it! 18

## Arduino Code

## CircuitPython Code
- Installing Libraries: 22
Overview

“Ba-DING!” The classic 8-bit chime from our favorite retro video game finally makes a triumphant come back as a Mario question block necklace.

It’s so tiny and adorable you’re probably wondering, “How much?” This is custom...you can only make it, not buy it!

In this DIY project, we'll show you how to build the Mario coin sound jewelry so you can wear it and learn about electronics and 3d printing.

This guide was written for the Trinket Mini and Gemma v2 boards. It has been updated to also support the Trinket M0 and Gemma M0 using CircuitPython. We recommend the Trinket M0 or Gemma M0 as they are easier to use and are more compatible with modern computers!

Prerequisite Guides

- Adafruit Trinket M0 (https://adafruit.it/zxF)
- Adafruit GEMMA M0 (https://adafruit.it/z1B)
- Introducing Trinket (https://adafruit.it/dhx)
- Introducing GEMMA (https://adafruit.it/duA)
Parts

- Adafruit Gemma M0 (https://adafru.it/ytb), Trinket M0 (https://adafru.it/zya), Trinket Mini (https://adafru.it/egk) or Gemma v2 (http://adafru.it/1222) microcontroller board (if Trinket, either the 3.3V (http://adafru.it/1500) or 5V (http://adafru.it/1501) type works). We recommend using the Gemma M0 (https://adafru.it/zya).
- LED Sequins (http://adafru.it/1758) (warm white, or any color if you prefer) (photo shows a single LED sequin, was later decided that two looks better)
- Slide Switch (http://adafru.it/805)
- 100mAh LiPoly battery (https://adafru.it/dAq)
- LiPoly backpack (https://adafru.it/e0w)- for recharging the batteries

Tools & Supplies

- Solder Iron (http://adafru.it/1204) + Solder (http://adafru.it/734)
- Silicone Wire (http://adafru.it/1877)
- PLA Filament (https://adafru.it/dtp)
- 3D Printer (https://adafru.it/tkf)
- Heat shrink tubing (https://adafru.it/diK)
- #2-56 machine screws
3D Printing

FDM 3D Printing

These parts are optimized to print with desktop 3D Printers capable of printing in ABS or PLA material with a minimum build area of 100mm x 100mm x 90mm. The two parts are designed to print without any support material.

Light Diffusion

The coinDiff.stl part should be printed in either transparent or a light colored material. This is the part that will diffuse the LED so that the question mark logo in the cover illuminates.

The standard case design fits a tiny 100 mAh LiPoly battery for maximum portability.
Slicing Software

The recommend settings above should work with most slicing software. However, you are encouraged to use your own settings since 3D printers and slicing software will vary from printer to printer.

PLA or ABS Material

We recommend using PLA material for an easier print with high quality. The tolerance has been tested with PLA filament but should also work with ABS. The parts do not require any support material or a raft.
## Assembly

<table>
<thead>
<tr>
<th>Trinket:</th>
<th>Gemma:</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>LEDs:</strong> 4 and GND</td>
<td><strong>LEDs:</strong> 0(-) 1(+)</td>
</tr>
<tr>
<td><strong>Vibration:</strong> 2 and GND</td>
<td><strong>Vibration:</strong> 2 and GND</td>
</tr>
<tr>
<td><strong>Piezo:</strong> 0(-) 1(+)</td>
<td><strong>Piezo:</strong> 0 (-) 1 (+)</td>
</tr>
<tr>
<td><strong>LiPoly backpack:</strong> 5v to USB+</td>
<td><strong>LiPoly backpack</strong> 5v to Vout</td>
</tr>
<tr>
<td>G to GND</td>
<td>G to GND</td>
</tr>
<tr>
<td>BAT to BAT+</td>
<td>BAT to 3vo</td>
</tr>
</tbody>
</table>

This guide was written for the Trinket Mini 5v board. The pins used are different for the Trinket M0 (see the second diagram). We recommend the Trinket M0 as it is easier to use and is more compatible with modern computers!
Trinket Mini 5v Circuit Diagram

To “–” pad on back of Trinket

Trinket M0 Circuit Diagram

To "-" pad on back of Trinket M0
Two connections will be made on the back of the Trinket, using one of the pads for the optional JST connector.

Measure Wires

Lay the LiPoly backpack on top of the Trinket and cut each wire so that they are just long enough reach the through-holes on the Trinket.

Prepare LiPoly backpack

Use a filing tool to remove the trace to the battery output line to enable use with a slide switch.
Solder LiPoly wires

Add a slide switch to each pin of the battery output line. Tin the pins for the LiPoly backpack and add the wires that will connect it to the Trinket.

Prepare Vibration sensor

Bend the thicker pin of the vibration sensor so that it curves level to the outer cylinder.
Align Vibration sensor to Trinket

Use a small amount of tac to hold the vibration sensor in place while aligning the pins to the pads on the back of the Trinket.

Vibration sensor thinner pin

Bend back the thinner pin so that it reaches into pin #2.
LiPoly backpack wires

Use tweezers to help align each wire into the Trinket through-holes.

LiPoly ground

Share the ground with the vibration sensor on the back pad of the Trinket.
This makes it easier to solder the LED wires on top of the Trinket.

(Photo shows a single LED...later it was decided that two LEDs in parallel worked better. See diagram at the top of this page for wiring. Later photos show placement of two LEDs.)

**Solder LEDs**

Measure the length needed to fully diffuse the inside of the case. LEDs have a specific polarity...+ should go to pin #4, – should go to GND. Two should be used in parallel (see diagram at the top of this page)...it should be possible to fit two wires into each hole on the Trinket.
Trinket tac

Use a small amount of tac to help secure the LiPoly backpack to the top of the Trinket.

Light pollution

Use strips of black tape to help clear the enclosure of any red or green lights from the LiPoly and Trinket LEDs.
Mount Piezo

Use two #2 56 phillip screws to mount the piezo to the back of the enclosure.

Don't solder the piezo to the Trinket until the wires are threaded through the back of the enclosure!
Coil piezo wires

To fit the circuit inside the tight space, coil the piezo wires inside of the standoffs for the Trinket.
Mount Trinket

Use one more #2 Phillips screw to securely mount the Trinket to inside of the enclosure.

Insert slide switch

Use curved (https://adafruit.it/dxN) tipped tweezers to insert the slide switch into the two clips near the top.

Diffuser

Snap the diffuser inside of the top cover part to help soften the light from the LEDs.
Position LEDs

Test that the diffusion is even by moving the two LEDs around in the enclosure until the lighting is even.

Wear it!

Attach a split ring and necklace to rock out with an audible bling to your step!

Arduino Code

The Arduino code presented below works well on Gemma v2 and Trinket original (ATtiny85). But if you have an M0 board you must use the CircuitPython code on the next page of this guide, no Arduino IDE required!
If this is your first time using Trinket or Gemma, work through the Introducing Trinket (https://adafruit.it/cEu) or Introducing Gemma (https://adafruit.it/cHH) guide first; you need to customize some settings in the Arduino IDE. Once you have it up and running (test the “blink” sketch), then continue...

In the Arduino IDE, create a new sketch (File → New), then copy and paste the following code (click the “copy code” link at the top right, switch to the Arduino IDE and select Edit → Paste).

The program is fairly small but uses some advanced techniques, so don’t be alarmed if a lot of it is unfamiliar. The important stuff you’ll actually be editing is on the next page.

```c
#include <avr/power.h>
#include <avr/sleep.h>

/* -----------------------------------------------------------------------
Super Mario Bros-inspired coin sound for Adafruit Trinket & Gemma.

Requires piezo speaker between pins 0 & 1, vibration sensor or
momentary button between pin 2 & GND. Tap for "bling!" noise.
Optional LED+resistor on pin 4 for light during play.

Runs equally well on a 16 MHz or 8 MHz Trinket, or on Gemma. Use what
you've got, no need to get all HOMG MOAR MEGAHURTZ!!! about it. :)

This is NOT good beginner code to learn from...there's very little
resemblance to a "normal" Arduino sketch as we poke around with ATtiny
peripheral registers directly; will NOT run on other Arduino boards.
Commented like mad regardless, might discover fun new stuff.

Written by Phillip Burgess for Adafruit Industries. Public domain.
----------------------------------------------------------------------- */

volatile uint8_t quietness;
volatile uint16_t count;

// ONE-TIME INITIALIZATION -----------------------------------------------
void setup() {
  // ATtiny85 has a special high-speed 64 MHz PLL mode than can be used
  // as an input to Timer/Counter 1. The ATmega chips don't have this!
  // Requires a little song and dance to set this up...
  PLLCSR |= _BV(PILLE);  // Enable 64 MHz PLL
}
```

©Adafruit Industries
delayMicroseconds(100); // Allow time to stabilize
while(!(PLLCSR & _BV(PLOCK))); // Wait for it...wait for it...
PLLCSR |= _BV(PCKE); // Timer1 source = PLL!

// Normally the Arduino core library uses Timer/Counter 1 for functions
// like delay(), millis(), etc. Having changed the cycle time above,
// and turning off the overflow interrupt, these functions won't work
// after this. Keeping track of time is our own responsibility now.

// The Timer/Counter 1 PWM output doesn't time the output square wave;
// the frequency (250 kHz) is much too fast for that. Rather, the
// speaker itself physically acts as a filter, with the average duty
// cycle determining the cone position; center-off, 0,255=extremes.
// A separate timer (using the actual sound frequency) then toggles the
// duty cycle to adjust amplitude, providing volume control...

void loop() {
  // To maximize power savings, pins are set to inputs with pull-up
  // resistors enabled (except for pins 1&4, because LEDs).
  DDRB = B00000000; PORTB = B00101101;

  // The chip is then put into a very low-power sleep mode...
  // Only the two timer/counters are re-enabled on wake. All other
  // peripherals remain off for power saving. This means no ADC, I2C, etc.
  power_all_disable(); // All peripherals off
  GIMSK |= _BV(INT0); // Enable external interrupt
  set_sleep_mode(SLEEP_MODE_PWR_DOWN); // Deepest sleep
  sleep_mode(); // Stop, will resume here on wake
  GIMSK &= ~(_BV(ISC01) | _BV(ISC00)); // Low level (GND) trigger
  // Code resumes when pin 2 is pulled to GND (e.g. button press).

  // Play first note. B5 = 987.77 Hz (round up to 988)
  pitch(988); // Sets up Timer/Counter 1 for this frequency
  // The pitch() function configures the timer for 2X the frequency, an
  // interrupt then alternates between the 'high' and 'low' parts of the
  // square wave. 988 Hz = 1976 interrupts/sec. 'count' keeps track.
  // First note is 0.083 sec. 1976 * 0.083 = 164 interrupt counts.
  // Combined length of notes is 0.92 sec, or 1818 interrupt counts at
  // this frequency. The amplitude (volume) fades linearly through the
  // duration of both notes. So this calculates the portion of that drop
  // through the first note...
  while(count < 164) quietness = 128L * count / 1818;
  // This uses fixed-point (integer) math, because floating-point is slow
  // on the AVR and uses lots of program space. A large integer multiply
  // (32-bit intermediate result) precedes an integer division, result is
  // effectively equal to floating-point multiply of 128.0 * 0.0 to 1.0.
pitch(1319); // Init second note.  E6 = 1318.51 Hz, round up to 1319. // 1319 Hz tone = 2638 Hz interrupt.  To maintain the duration and make // the volume-scaling math continue from the prior level, counts need to // be adjusted to take this timing change into account. The total // length at this rate would be 2638 * 0.92 = 2427 counts, and first note duration would have been 2638 * 0.083 = 219 counts... count = 219; // Rather than counting up to the duration, just keep playing until the // effective volume is zero. do { quietness = 128L * count / 2427; } while (quietness < 127); // Finished playing both notes. Disable the timer interrupt... TIMSK = 0; // Before finishing, the piezo speaker is eased in a controlled manner // from the volume-neutral position (127) to its off position (0) in // order to avoid an audible 'pop' when the code goes to sleep. for (uint8_t i = 127; i--; ) { OCR1A = i; // Speaker position for (volatile uint16_t x = F_CPU/32000; --x; ); // Easy, not too fast } // ----------------------------------------------------------------------- // These tables list available timer/counter prescaler values and their // configuration bit settings. Normally I'd PROGMEM these, but for these // short tables the code actually compiles a little smaller this way! const uint16_t prescale[] = {1, 8, 64, 256, 1024}; const uint8_t tccr[] = {_BV(WGM02) | _BV(CS00), _BV(WGM02) | _BV(CS01), _BV(WGM02) | _BV(CS01) | _BV(CS00), _BV(WGM02) | _BV(CS02), _BV(WGM02) | _BV(CS02),}; // Configure Timer/Counter 0 for the requested frequency void pitch(uint16_t freq) { uint8_t i; uint16_t f2 = freq << 1; // 2X frequency uint32_t o; // Find CPU prescale that can accommodate requested frequency for (i = 0; i < (sizeof(prescale) / sizeof(prescale[0])) && (o = (((F_CPU / (uint32_t)prescale[i]) + freq) / (uint32_t)f2) - 1) >= 256L; i++); TCCR0B = tccr[i]; // Prescale config bits OCR0A = (uint8_t)o; // PWM interval for 2X freq count = 0; // Reset waveform counter TIMSK = _BV(OCIE0A); // Enable compare match interrupt } // TIMER0_0VF vector is already claimed by the Arduino core library, // can't use that. So the compare vector is used instead... ISR(TIMER0_COMPA_vect) { // Bit 0 of count indicates high or low side of square wave. // OCR1A sets average speaker pos, quietness adjusts amplitude. OCR1A = (count++ & 1) ? 255 - quietness : quietness; }
From the Tools → Board menu, select Adafruit Trinket 8 MHz or Adafruit Gemma as appropriate. Connect the USB cable between the computer and board, press the reset button, then click the upload button (right arrow icon) in the Arduino IDE. In a moment you should get a light show from the LEDs. (If it doesn’t, check your wiring against the schematics. If the code refuses to compile, most likely the TinyWireM library isn’t correctly installed, or the anim.h file is mis-named.)

CircuitPython Code

Trinket M0 boards can run CircuitPython — a different approach to programming compared to Arduino sketches. In fact, CircuitPython comes factory pre-loaded on Trinket M0. 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 (https://adafruit.it/zxF) Trinket M0 guide (https://adafruit.it/zxF).

These directions are specific to the "M0" boards. The original Trinket with an 8-bit AVR microcontroller doesn’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 (though not exactly the same) as the Arduino sketch shown on a prior page. To use this, plug the Trinket M0 into USB...it should show up on your computer as a small flash drive...then edit the file “code.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 M0 doesn’t show up as a drive, follow the Trinket M0 guide link above to prepare the board for CircuitPython.

```python
import time
import board
import simpleio
import pwmio
import digitalio

# PWM is not available on Trinket D1
vibration_pin = board.D1  # vibration switch is connected
speaker_pin = board.D2   # PWM speaker
pwm_leds = board.D4     # PWM "fading" LEDs

# initialize PWM for LEDs
pwm = pwmio.PWMOut(pwm_leds, frequency=256, duty_cycle=50)
led_fade_delay = .001   # delay in seconds makes color fade visible
led_fade_step = 1024    # fade amount

def led_fade(brightness):
    pwm.duty_cycle = brightness
    brightness_start = brightness

    while brightness >= (brightness_start / 2):
        brightness -= led_fade_step
        pwm.duty_cycle = brightness
        time.sleep(led_fade_delay)

while True:
    # wait for vibration sensor detect (reverse logic)
    # play Super Mario Bros. coin sound
    # fade LEDs
    if not vpin.value:
        led_fade((2 ** 16) - 1)  # full brightness
        simpleio.tone(speaker_pin, 988, 0.083)  # tone1 - B5
        led_fade(2 ** 15)          # half brightness
        simpleio.tone(speaker_pin, 1319, 0.83) # tone2 - E6
        led_fade(2 ** 14)          # quarter brightness
        pwm.duty_cycle = 0         # turn off LEDs
```

©Adafruit Industries
Installing Libraries:

The simpleio library must be installed for the above code to run correctly. The latest version of the Adafruit CircuitPython Library Bundle (https://adafru.it/Ayy) contains this library. You want to download the latest stable mpy bundle which will have a filename like this:

**adafruit-circuitpython-bundle-x.x.x-mpy-date.zip**

The Trinket M0 has limited space, but so in this case we will be selective about which files are copied over to the CIRCUITPY drive. A detailed explanation for installing libraries is available (https://adafru.it/ABU).

Copy the following file from the unzip'd CircuitPython Library Bundle to the CIRCUITPY drive to a new folder called 'lib':

- simpleio.mpy