Buzzing Mindfulness Bracelet

Created by Becky Stern

https://learn.adafruit.com/buzzing-mindfulness-bracelet

Last updated on 2023-08-29 02:57:16 PM EDT

Table of Contents

Overview

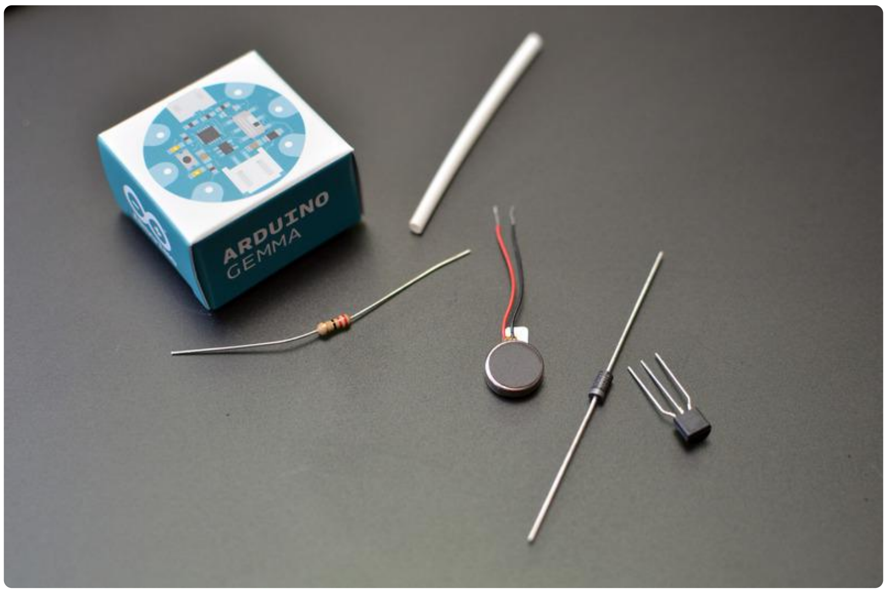
This guide was written for the Gemma board, but can be done with either the v1, v2 or Gemma M0. We recommend the Gemma M0 as it is easier to use and is more compatible with modern computers!

Build yourself a buzzing bracelet for subtle haptic feedback as time passes! It's great for reminding yourself to get up and walk away from your desk for a few minutes each hour, or just as a way to have a new awareness of how the perception of passing time varies based on what you're doing.

You'll whip up a vibrating motor circuit using a transistor, resistor, and diode, and use GEMMA to control the frequency of vibration in between low-power microcontroller naps. The circuit lives inside a linked leather/rubber bracelet, but you could build it into whatever you please. This project involves some precision soldering, but is otherwise quite easy!

Before you begin, make sure you've read the following prerequisite guides:

- Gemma M0 guide () or Classic Introducing GEMMA guide ()
- Adafruit Guide To Excellent Soldering ()
- Battery Powering your Wearable Electronics ()

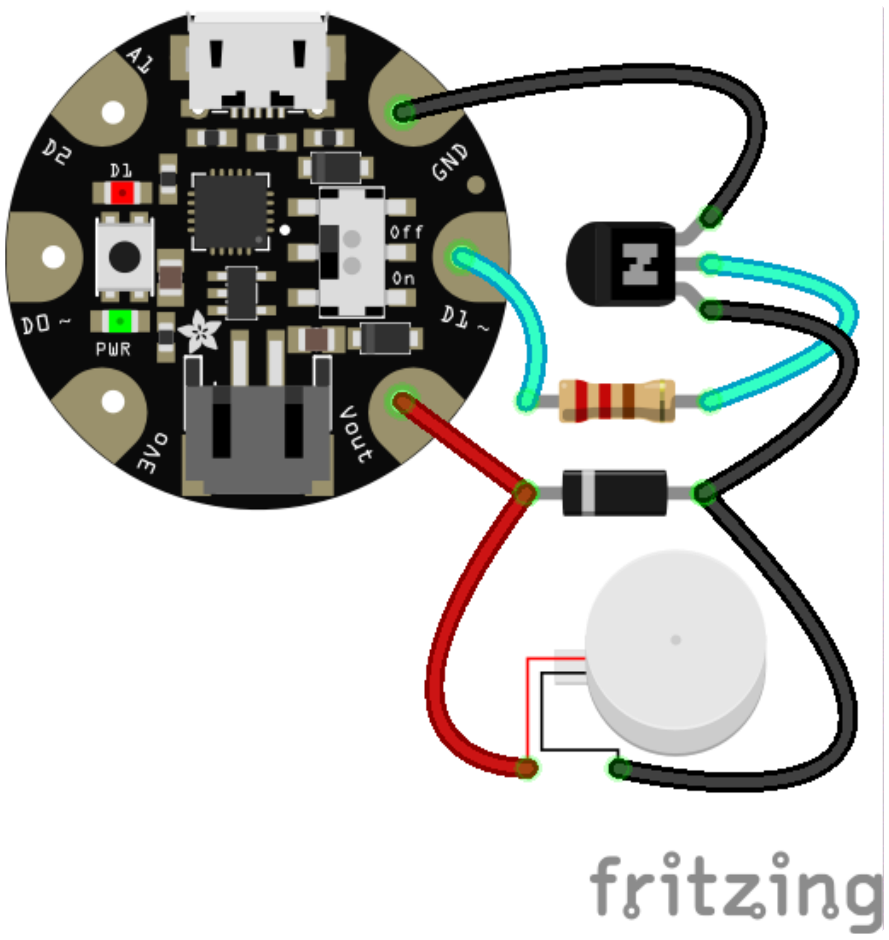


For this project, you will need:

- Gemma M0 () or (http://adafru.it/1138)Gemma v2 (http://adafru.it/1222) or Gemma v1 (discontinued) ()

- vibrating mini disc motor ()
- 1N4001 diode ()
- PN2222 NPN transistor ()
- ~200-1K ohm resistor
- 100mAh lipoly battery ()
- heat shrink tubing ()
- soldering iron and accessories ()
- scraps of leather or bike inner tube
- scissors
- snaps and snap setting tool or velcro tape
- utility knife
- ruler
- pen or marker

Circuit Diagram

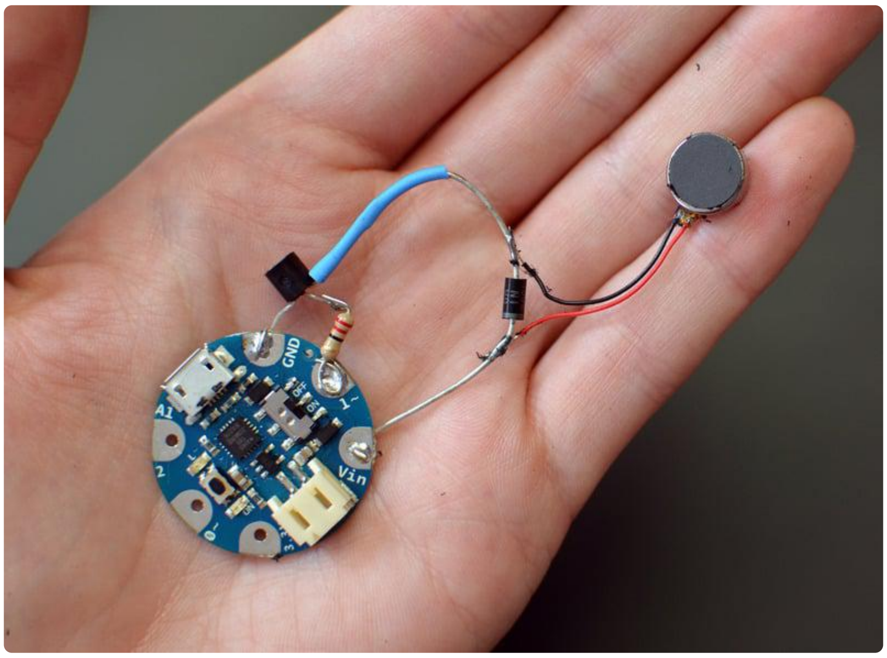


This diagram uses the Gemma v2 but you can also use the Gemma M0 or the original Gemma v1 with the exact same wiring!

A see-through bracelet would show off the circuit above:

- ~220-1K ohm resistor to GEMMA pin 1
- NPN transistor base (center pin on PN2222) to resistor
- NPN transistor emitter to GEMMA ground
- NPN transistor collector to motor black wire AND diode anode
- GEMMA Vin to motor red wire AND diode cathode (gray stripe)
- Battery plugs into tan JST port

It's difficult to fit the circuit inside the bracelet if you build it first. We'll be building it as we go, interlocked with pieces of the bracelet for the most compact fit.



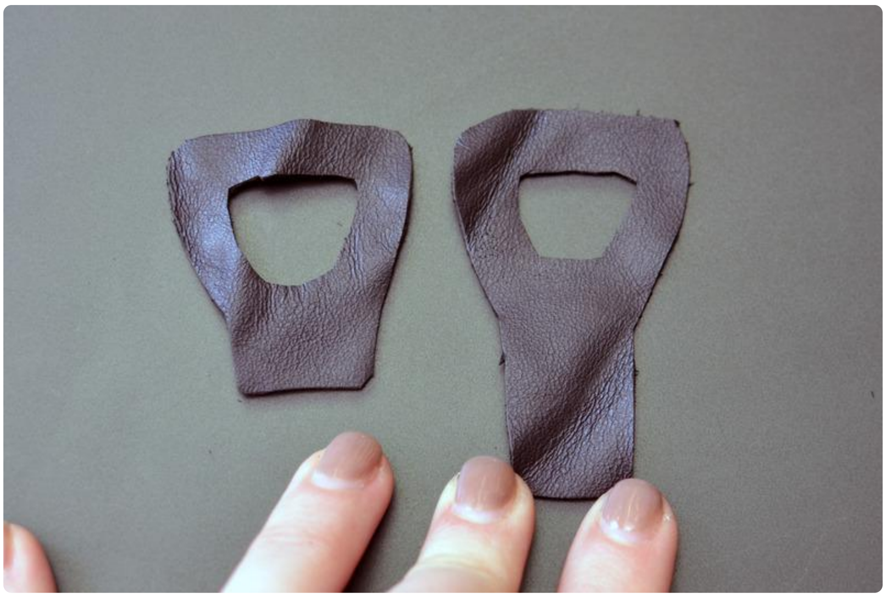
Bracelet

The bracelet is created from folded figure-8 shapes cut from leather or rubber. You can easily draw it yourself, or download this image and print it out ~1 inch wide (or trace the screen!):

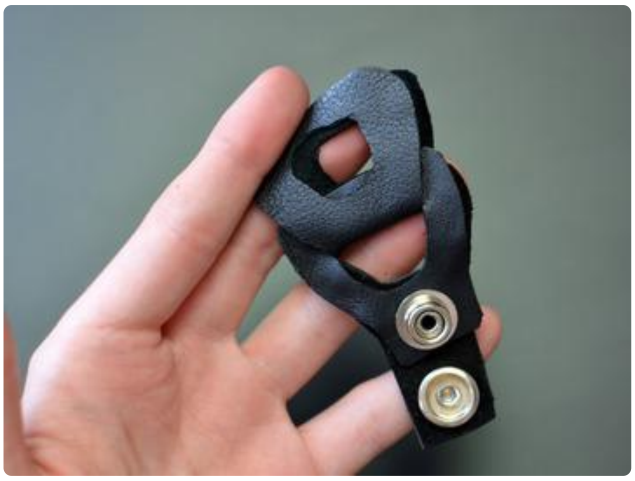
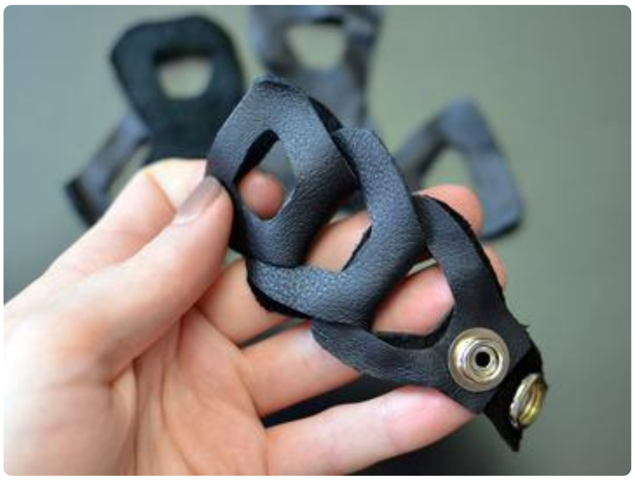
Draw or glue the template to a piece of thick paper, illustration board, or piece of cardboard. Cut out the figure-8 shape using a sharp utility knife or craft blade.

On the wrong side of your material, trace the template many times and carefully cut out a small pile of pieces using sharp scissors.

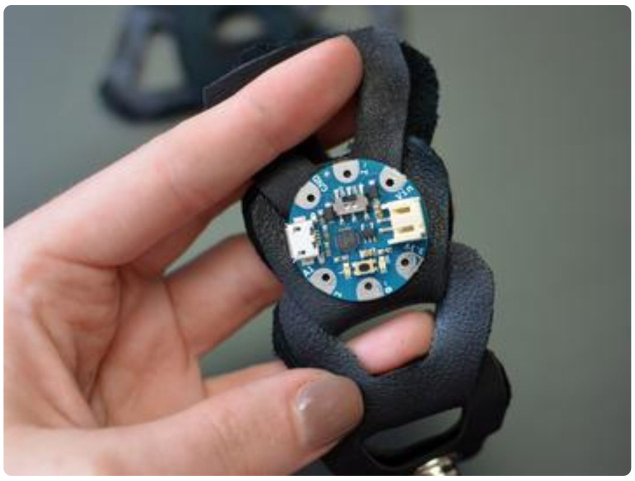
To create the bracelet closure, cut out two special pieces with elongated tabs as shown:

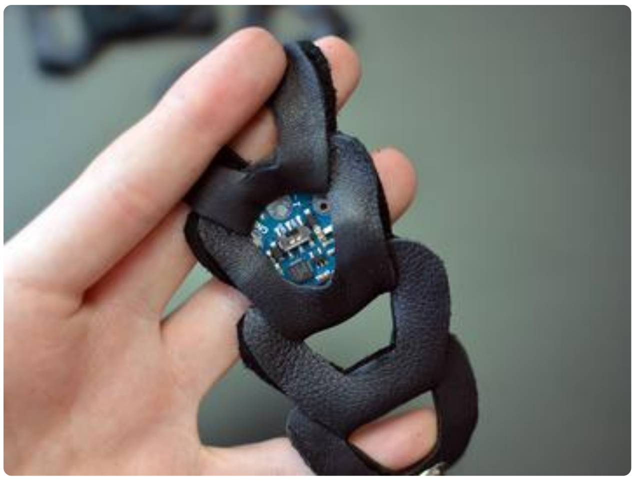


Use a snap setting tool to attach the snap parts, or use velcro tape instead.

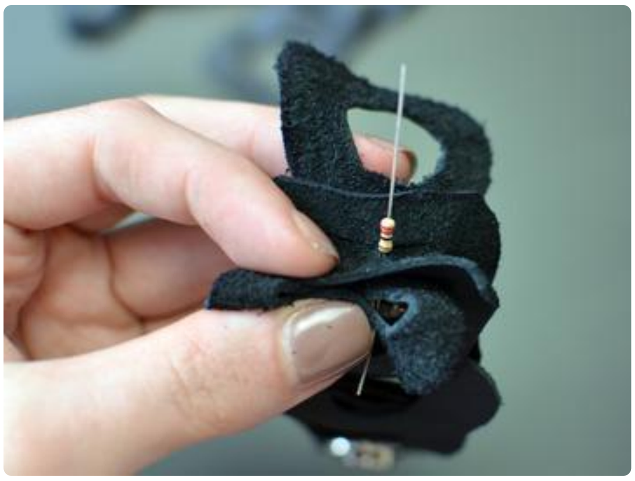
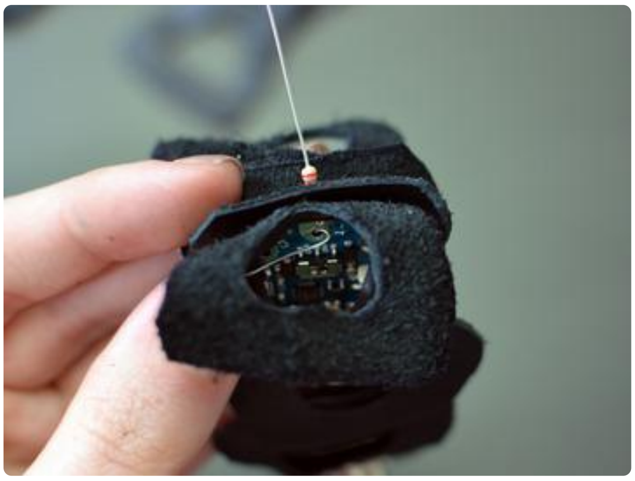



Sandwich the two pieces wrong sides together, and place a figure-8 through the center hole. Fold the figure-8 flat so it creates a new sandwich for inserting the next piece. Repeat to build up a few links.

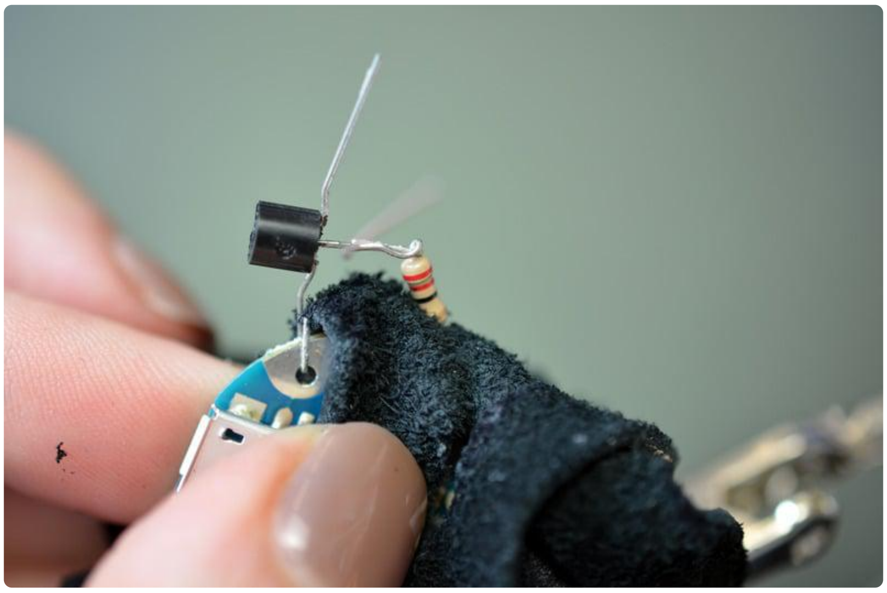




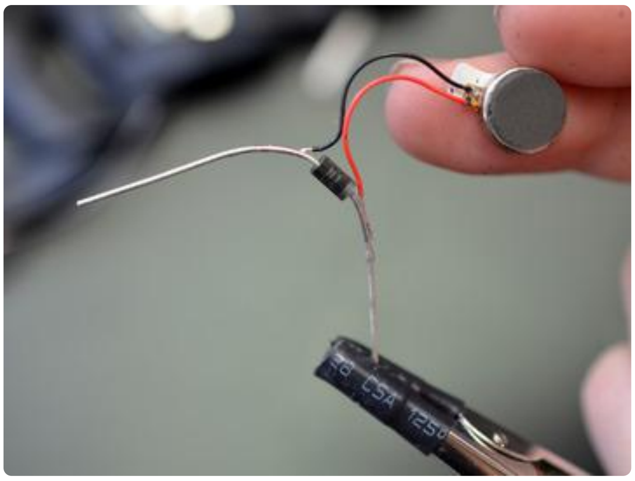
GEMMA fits snugly between the layers, which expose the power switch but hide the USB connector and JST battery connector. Slide GEMMA in from the side, with digital pin 1 facing towards the working end of the chain.

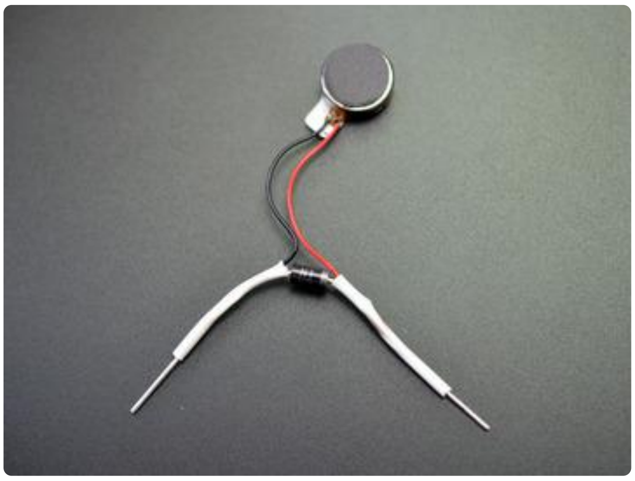
Poke a small hole in the center of the next link and insert one leg of the resistor (doesn't matter which), then also through GEMMA's pin 1. Twist the lead so it makes good mechanical contact with the solder pad.



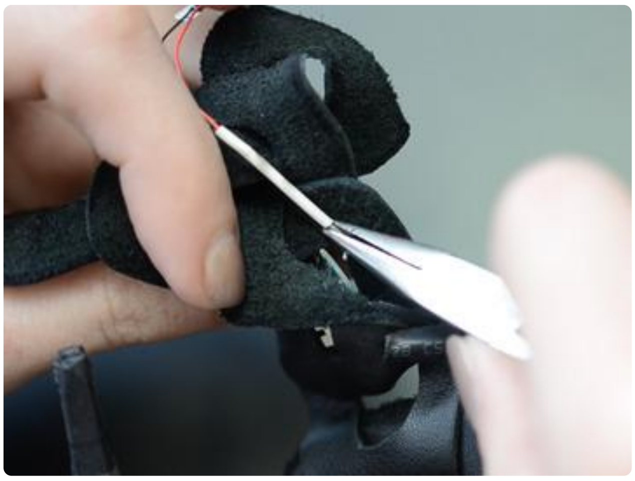
Next up is the transistor. Splay the leads away from each other, and use pliers to carefully bend them into place. Pictured above, the flat side of the transistor faces the same way as the flat side of GEMMA. Solder the base and emitter pins, being careful not to singe the material of the bracelet (although black leather can be very forgiving in this regard).



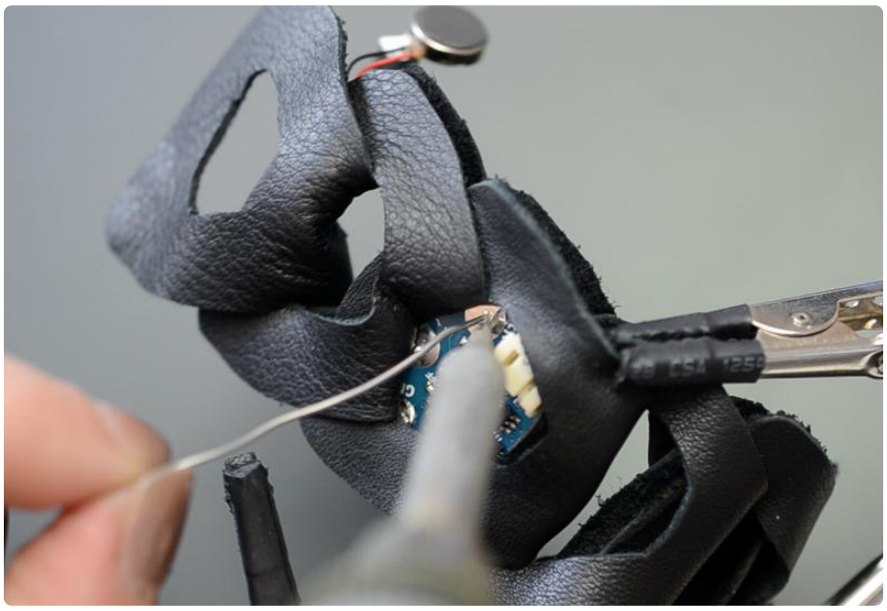
To prepare the motor/diode assembly, set up your diode in a third hand tool and tin the leads close to the diode itself. Hold the delicate motor wires up to the leads and remelt the solder to affix the motor.



Use heat shrink tubing to insulate the leads and help protect the tiny stranded wires connected to the motor.



Solder the motor/diode assembly to the transistor's remaining pin (collector) and GEMMA Vin. The diode leads travel around the next link in the chain, concealing and protecting the circuit.



Arduino Code

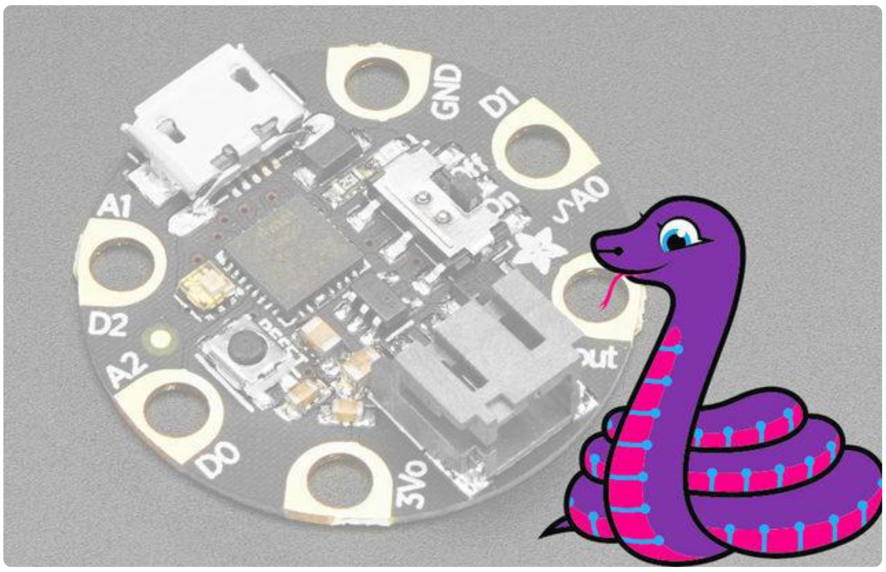
This guide was written for the original Gemma and Gemma v2 boards. If you have a Gemma M0 you must use CircuitPython. We recommend the Gemma M0 or Circuit Playground Express as they are easier to use and are more compatible with modern computers!

Load up the following code on Arduino GEMMA using the Arduino IDE. It should buzz the motor once per minute. If it's working, then adjust the interval variable to your desired number of seconds and reload the code.

```
// SPDX-FileCopyrightText: 2018 Mikey Sklar for Adafruit Industries
//
// SPDX-License-Identifier: MIT

// Mindfulness Bracelet sketch for Adafruit/Arduino Gemma.  Briefly runs
// vibrating motor (connected through transistor) at regular intervals.
// This code is not beginner-friendly, it does a lot of esoteric low-level
// hardware shenanigans in order to conserve battery power.

const uint32_t           // These may be the only lines you need to edit...
  onTime   =  2 * 1000L, // Vibration motor run time, in milliseconds
  interval = 60 * 1000L; // Time between reminders, in milliseconds
                         // It gets progressively geekier from here...

// Additional power savings can optionally be realized by disabling the
// power-on LED, either by desoldering or by cutting the trace from 3Vo
// on the component side of the board.

// This sketch spends nearly all its time in a low-power sleep state...
#include <avr/power.h>
#include <avr/sleep.h>

// The chip's 'watchdog timer' (WDT) is used to wake up the CPU when needed.
// WDT runs on its own 128 KHz clock source independent of main CPU clock.
// Uncalibrated -- it's "128 KHz-ish" -- thus not reliable for extended
// timekeeping.  To compensate, immediately at startup the WDT is run for
// one maximum-duration cycle (about 8 seconds...ish) while keeping the CPU
// awake, the actual elapsed time is noted and used as a point of reference
// when calculating sleep times.  Still quite sloppy -- the WDT only has a
// max resolution down to 16 ms -- this may drift up to 30 seconds per hour,
// but is an improvement over the 'raw' WDT clock and is adequate for this
// casual, non-medical, non-Mars-landing application.  Alternatives would
// require keeping the CPU awake, draining the battery much quicker.

uint16_t          maxSleepInterval;  // Actual ms in '8-ish sec' WDT interval
volatile uint32_t sleepTime     = 1; // Total milliseconds remaining in sleep
volatile uint16_t sleepInterval = 1; // ms to subtract in current WDT cycle
volatile uint8_t  tablePos      = 0; // Index into WDT configuration table
```

```
void setup() {

  // Unused pins can be set to INPUT w/pullup -- most power-efficient state
  pinMode(0, INPUT_PULLUP);
  pinMode(2, INPUT_PULLUP);

  // LED shenanigans.  Rather that setting pin 1 to an output and using
  // digitalWrite() to turn the LED on or off, the internal pull-up resistor
  // (about 10K) is enabled or disabled, dimly lighting the LED with much
  // less current.
  pinMode(1, INPUT);               // LED off to start

  // AVR peripherals that are NEVER used by the sketch are disabled to save
  // tiny bits of power.  Some have side-effects, don't do this willy-nilly.
  // If using analogWrite() to for different motor levels, timer 0 and/or 1
  // must be enabled -- for power efficiency they could be turned off in the
  // ubersleep() function and re-enabled on wake.
  power_adc_disable();             // Knocks out analogRead()
  power_timer1_disable();          // May knock out analogWrite()
  power_usi_disable();             // Knocks out TinyWire library
  DIDR0 = _BV(AIN1D) | _BV(AIN0D); // Digital input disable on analog pins
  // Timer 0 isn't disabled yet...it's needed for one thing first...

  // The aforementioned watchdog timer calibration...
  uint32_t t = millis();                       // Save start time
  noInterrupts();                              // Timing-critical...
  MCUSR &= ~_BV(WDRF);                         // Watchdog reset flag
  WDTCR  =  _BV(WDCE) | _BV(WDE);              // WDT change enable
  WDTCR  =  _BV(WDIE) | _BV(WDP3) | _BV(WDP0); // 8192-ish ms interval
  interrupts();
  while(sleepTime);                            // Wait for WDT
  maxSleepInterval  = millis() - t;            // Actual ms elapsed
  maxSleepInterval += 64;                      // Egyptian constant
  power_timer0_disable();  // Knocks out millis(), delay(), analogWrite()
}

const uint32_t offTime = interval - onTime; // Duration motor is off, ms

void loop() {
  pinMode(1, INPUT);        // LED off
  ubersleep(offTime);       // Delay while off
}

// WDT timer operates only in specific intervals based on a prescaler.
// CPU wakes on each interval, prescaler is adjusted as needed to pick off
// the longest setting possible on each pass, until requested milliseconds
// have elapsed.
const uint8_t cfg[] PROGMEM = { // WDT config bits for different intervals
  _BV(WDIE) | _BV(WDP3) |                         _BV(WDP0), // ~8192 ms
  _BV(WDIE) | _BV(WDP3)                                    , // ~4096 ms
  _BV(WDIE) |             _BV(WDP2) | _BV(WDP1) | _BV(WDP0), // ~2048 ms
  _BV(WDIE) |             _BV(WDP2) | _BV(WDP1)            , // ~1024 ms
  _BV(WDIE) |             _BV(WDP2) |             _BV(WDP0), //  ~512 ms
  _BV(WDIE) |             _BV(WDP2)                        , //  ~256 ms
  _BV(WDIE) |                         _BV(WDP1) | _BV(WDP0), //  ~128 ms
  _BV(WDIE) |                         _BV(WDP1)            , //   ~64 ms
  _BV(WDIE) |                                     _BV(WDP0), //   ~32 ms
  _BV(WDIE)                                                  //   ~16 ms
}; // Remember, WDT clock is uncalibrated, times are "ish"

void ubersleep(uint32_t ms) {
  if(ms == 0) return;
  tablePos      = 0;                  // Reset WDT config stuff to
  sleepInterval = maxSleepInterval;   // longest interval to start
  configWDT(ms);                      // Set up for requested time
  set_sleep_mode(SLEEP_MODE_PWR_DOWN); // Deepest sleep mode
  sleep_enable();
  while(sleepTime && (tablePos < sizeof(cfg))) sleep_mode();
  noInterrupts();                      // WDT off (timing critical)...
  MCUSR &= ~_BV(WDRF);
  WDTCR  = 0;
  interrupts();
}

static void configWDT(uint32_t newTime) {
  sleepTime = newTime; // Total sleep time remaining (ms)
  // Find next longest WDT interval that fits within remaining time...
  while(sleepInterval > newTime) {
    sleepInterval /= 2;                          // Each is 1/2 previous
    if(++tablePos >= sizeof(cfg)) return;        // No shorter intervals
  }
  uint8_t bits = pgm_read_byte(&cfg[tablePos]);  // WDT config bits for time
  noInterrupts();                                // Timing-critical...
  MCUSR &= ~_BV(WDRF);
  WDTCR  =  _BV(WDCE) | _BV(WDE);                // WDT change enable
  WDTCR  =  bits;                                // Interrupt + prescale
  interrupts();
}

ISR(WDT_vect) { // Watchdog timeout interrupt
  configWDT(sleepTime - sleepInterval); // Subtract, setup next cycle...
}
```

CircuitPython Code



GEMMA M0 boards can run CircuitPython — a different approach to programming compared to Arduino sketches. In fact, CircuitPython comes factory pre-loaded on GEMMA 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 GEMMA M0 guide ().

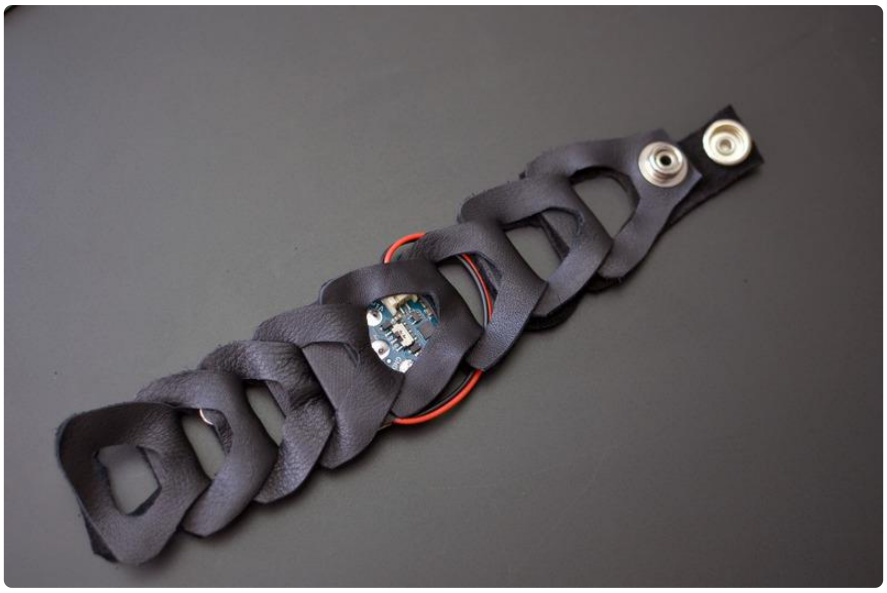
These directions are specific to the "M0" GEMMA board. The original GEMMA with an 8-bit AVR microcontroller doesn't run CircuitPython...for those boards, use the Arduino sketch on the "Arduino code" page of this guide.

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

```
# SPDX-FileCopyrightText: 2018 Mikey Sklar for Adafruit Industries
#
# SPDX-License-Identifier: MIT

# Mindfulness Bracelet sketch for Adafruit Gemma.  Briefly runs
# vibrating motor (connected through transistor) at regular intervals.

import time
import board
from digitalio import DigitalInOut, Direction

# vibrating disc mini motor disc connected on D1
vibrating_disc = DigitalInOut(board.D1)
vibrating_disc.direction = Direction.OUTPUT

on_time = 2     # Vibration motor run time, in seconds
interval = 60   # Time between reminders, in seconds

start_time = time.monotonic()

while True:

    timer = time.monotonic() - start_time

    if timer >= interval and timer <= (interval + on_time):
        vibrating_disc.value = True

    elif timer >= (interval + on_time):
        vibrating_disc.value = False
        start_time = time.monotonic()
```

Wear it!



Add enough links after your circuit for your bracelet to fit your wrist. Peel off the paper backing from the motor and stick it in between layers of material.

Put some black tape or heat shrink tubing around your battery and plug it in. Wind the wire through the bracelet and tuck the battery between the layers of material. Loop the closure tab through the last link and wear it!

Your bracelet is not waterproof! You can take steps to ruggedize your circuit () to make it more splash resistant, if that's your thing.

You can customize your bracelet's vibration by changing up the Arduino code. Maybe try using PWM to give the vibration more nuance or create different buzz patterns for different spans of time passed-- how would you make this project your own?

GND