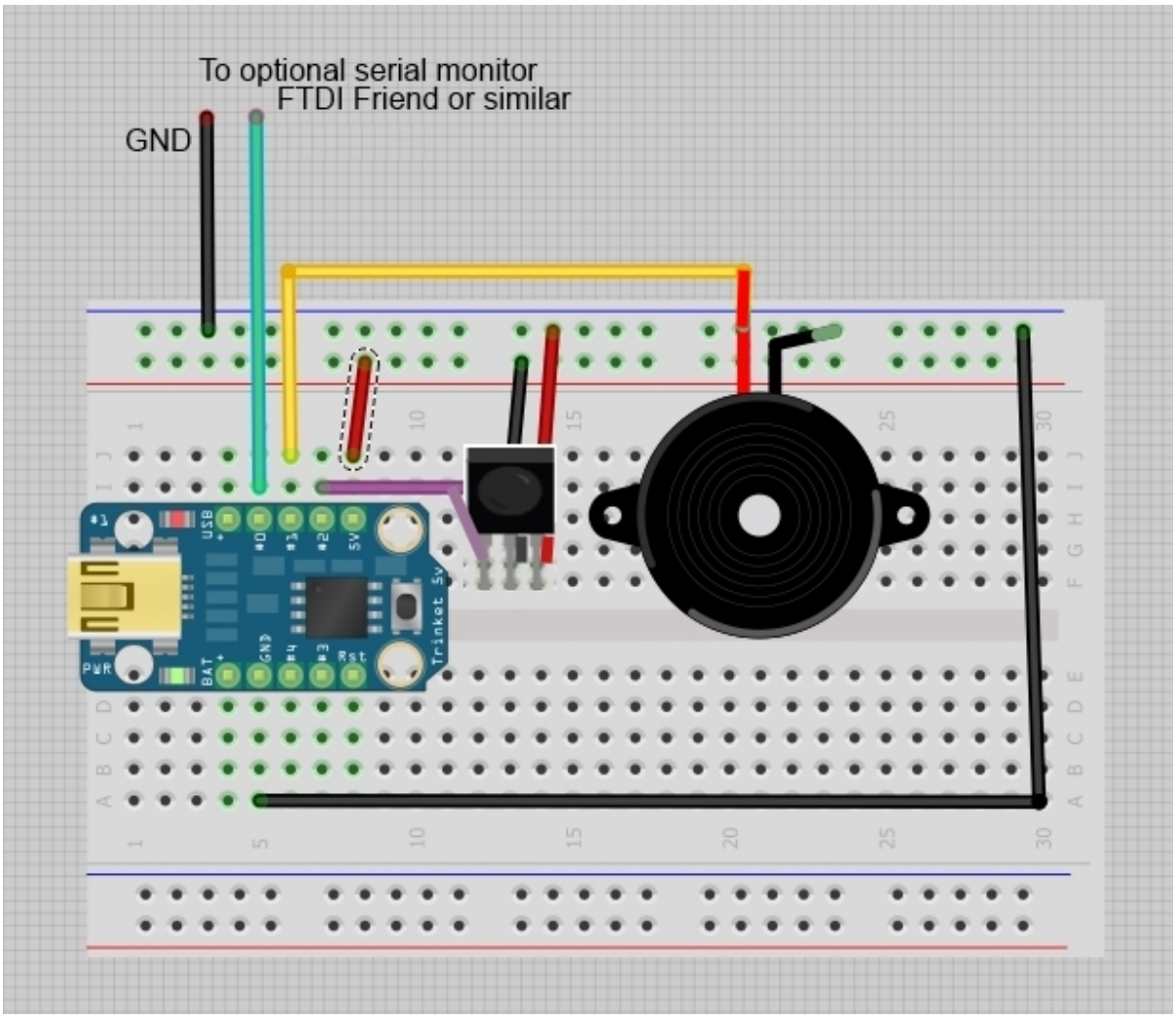
Trinket / Gemma IR Control

Created by Anne Barela



https://learn.adafruit.com/trinket-gemma-ir-remote-control

Last updated on 2025-12-09 04:41:19 PM EST

Table of Contents

Overview 3

Wiring 4

- Wiring Diagram #1 (above) - Gemma M0 or Trinket M0:
- Wiring Diagram #2 (above) - Gemma v2 with USB to TTL Serial Cable
- Wiring Diagram #3 (above) - Trinket Mini with FTDI Friend

Arduino Code 8

- IR Reader
- IR Remote Speaker

CircuitPython Code 12

- IR Reader
- IR Remote Speaker

Use and Going Further 16

- Video
- Going Further

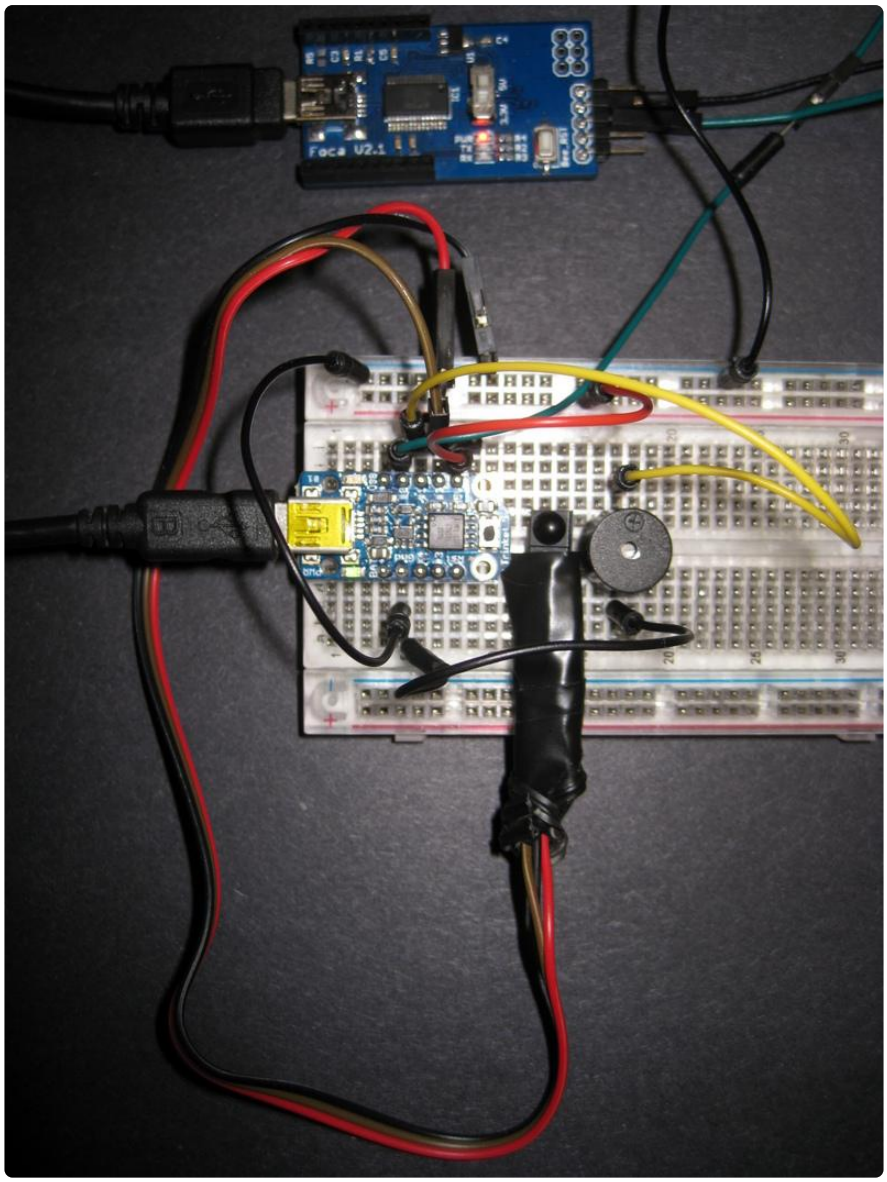
Overview

Trinket and Gemma are perfect for small projects needing to receive some external event, triggering your own defined output. This project uses the Adafruit IR Sensor to first receive IR commands from a remote, then to use those codes in controlling a project of your own.

To learn about IR signals and how they are decoded by a microcontroller, see the IR Sensor Tutorial (https://adafru.it/cJH) from which this tutorial is based.

Many larger Arduino projects use the excellent IRRemote library (https://adafru.it/cJI) by Ken Shirriff. This library allows for multiple protocols and is quite flexible. But it relies on hardware specific to larger Arduino processors. This project simplifies the process of obtaining codes and using them to scale to the limits of the Trinket and Gemma boards.

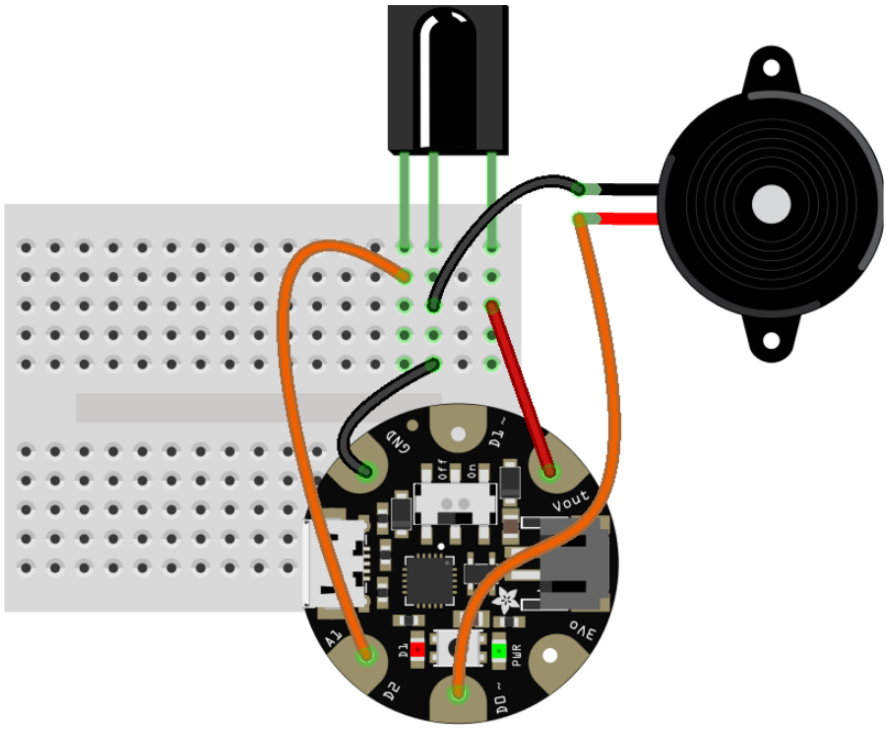
guide was written for the Trinket Mini and Gemma v2 boards. It has been ated to also support the Trinket M0 and Gemma M0 using CircuitPython. ecommend the Trinket M0 and Gemma M0 as it is easier to use and is e compatible with modern computers!



Wiring

Below we have three different wiring diagrams. We recommend using the first one (**Gemma M0 or Trinket M0**) which requires the least amount of wiring, code while providing maximum debug ability.

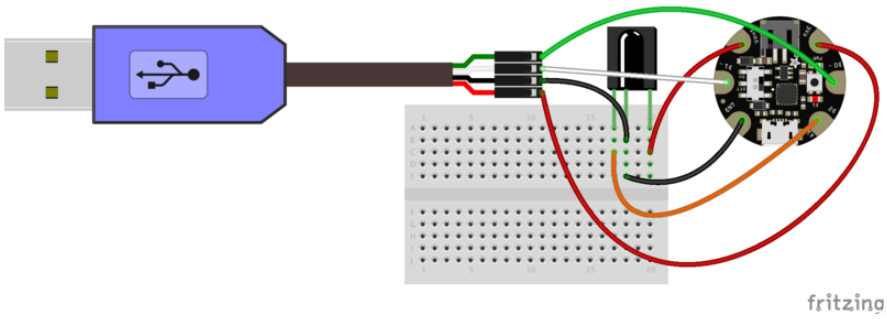
1. **Gemma M0 or Trinket M0** - Minimal wiring with built in console debug ability, CircuitPython only.
2. **Gemma v2 with USB to TTL Serial Cable** - (also works with the Trinket Mini) - with USB to TTL Serial Cable. Arduino only.
3. **Trinket Mini with FTDI Friend** (also works with Gemma v2) - Arduino only.



Wiring Diagram #1 (above) - **Gemma M0 or Trinket M0**:

No need for a serial breakout with this setup. It is built right into the M0 series. This wiring is straight forward. This setup will only work with the CircuitPython example code.

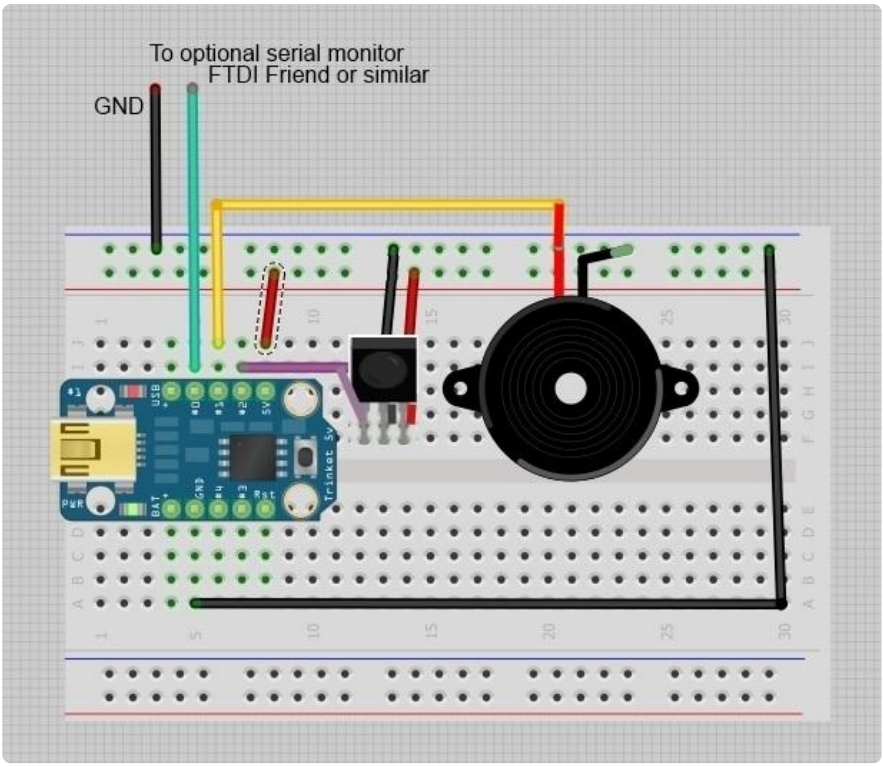
- Gemma M0 D0 <--> Piezzo Speaker (+)
- Gemma M0 D2 <--> IR Output
- Gemma M0 Vout <--> IR (+)
- Gemma M0 GND <--> IR (-)
- Gemma M0 GND <--> Piezo Speaker (-)



Wiring Diagram #2 (above) - **Gemma v2 with** [USB to TTL Serial Cable](http://adafru.it/954) **(http://adafru.it/954)**

This version allows for a debug console using the older Gemma v2 (ATtiny85 based). This setup is good for detecting IR codes and watching them print to the console. The wiring goes as follows:

- Gemma v2 D0 (RX) <--> FTDI Green (TX out of the USB port)
- Gemma v2 D1 (TX) <--> FTDI White (RX out of the USB port)
- Gemma v2 D2 <-> IR Output
- Gemma v2 3v3 <--> FTDI Red (+)
- Gemma v2 GND <--> IR (-)
- Gemma v2 GND <--> FTDI Black (-)



Wiring Diagram #3 (above) - **Trinket Mini with FTDI Friend**

The diagram is shown below. The IR data pin is connected to the Trinket GPIO #2 (Gemma Pin D2) and is connected to power and ground. To read codes, you will need to connect Trinket GPIO #0 / Gemma D0 to a serial to USB board such as the FTDI Friend (http://adafru.it/284) receive RX pin (cross connect).

To demonstrate how the Trinket or Gemma may process IR commands into an action of your choice, a piezo speaker is connected to Trinket Pin GPIO #1 (Gemma Pin D1) to output a tone when a certain IR code is received.

Going further, you can use an IR code to change NeoPixels, a servo, a solenoid, or

any other output.

- If using an FTDI Friend: The green **#0** wire should connect to the **RX** pin and the black **GND** wire connects to the **GND** pin

- If using a USB console cable: The green **#0** wire connects to the **White Wire on the console cable** and the black **GND** wire to the **Black** ground wire on the console cable.

Arduino Code

Arduino code presented below works well on Gemma v2 and Trinket Mini. if you have an M0 board you must use the CircuitPython code on the next e of this guide, no Arduino IDE required!

IR Reader

The sketch for reading the IR codes for a new remote is below. We will be using Wiring Diagram#the The earlier Gemma v2 and Trinket Mini do not have a built-in hardware serial so we will be using Wiring Diagram #2 (https://adafru.it/CcR) Gemma v2 with the USB to TTL Serial Cable (http://adafru.it/954). Be sure the Arduino serial console (Putty or screen also work) to 9600 baud, use 8 bits, 1 stop bit, no parity.

Plug in Gemma v2 to your computer with a USB cable and load up the following:

```
// SPDX-FileCopyrightText: 2018 Limor Fried/ladyada for Adafruit Industries
//
// SPDX-License-Identifier: CC-BY-SA-3.0

/* Trinket/Gemma compatible Raw IR decoder sketch
This sketch/program uses an Adafruit Trinket or Gemma
ATTiny85 based mini microcontroller and a PNA4602 to
decode IR received. This can be used to make a IR receiver
(by looking for a particular code) or transmitter
(by pulsing an IR LED at ~38KHz for the durations pulse_index

Based on Adafruit tutorial http://learn.adafruit.com/ir-sensor/using-an-ir-sensor

and ATTiny program by TinyPCRemote Nathan Chantrell http://nathan.chantrell.net
```

```
under Creative Commons Attribution-ShareAlike 3.0 Unported (CC BY-SA 3.0) license

SendSoftwareSerial Lirary modification by Nick Gammon from NewSoftwareSerial code
GNU Lesser General Public License as published by the Free Software Foundation
version 2.1
at http://gammon.com.au/Arduino/SendOnlySoftwareSerial.zip
*/
#include <SoftwareSerial.h>       // use if you do not wish to use the lightweight
library

SoftwareSerial Serial(0,1);      // Receive, Transmit (Receive not used)

// We need to use the 'raw' pin reading methods because timing is very important
here
// and the digitalRead() procedure is slower!
#define IRpin_PIN PINB // ATTiny85 had Port B pins
#define IRpin 2

#define MAXPULSE    12000  // the maximum pulse we'll listen for - 5 milliseconds
#define NUMPULSES   34    // max IR pulse pairs to sample
#define RESOLUTION  2     // time between IR measurements
#define STORED_BUTTON_CODES 4 // remote control codes stored

// we will store up to 100 pulse pairs (this is -a lot-)
uint16_t pulses[NUMPULSES]; // high and low pulses
uint16_t pulse_index = 0;  // index for pulses we're storing
uint32_t irCode = 0;

void setup(void) {
  Serial.begin(9600);
  Serial.println();
  Serial.println("Ready to decode IR!");
  pinMode(IRpin, INPUT);   // Listen to IR receiver on Trinket/Gemma pin D2
}

void loop(void) {
  // Wait for an IR Code
  uint16_t numpulse=listenForIR();

  // Process the pulses to get a single number representing code
  for (int i = 0; i < NUMPULSES; i++) {
      Serial.print(pulses[i]);
      Serial.print(", ");
  }
    Serial.println("\n");
}

uint16_t listenForIR() {  // IR receive code
  pulse_index = 0;
  while (1) {
   unsigned int highpulse, lowpulse;  // temporary storage timing
   highpulse = lowpulse = 0; // start out with no pulse length

   while (IRpin_PIN & _BV(IRpin)) { // got a high pulse
      highpulse++;
      delayMicroseconds(RESOLUTION);
      if (((highpulse >= MAXPULSE) && (pulse_index != 0))|| pulse_index ==
NUMPULSES) {
        return pulse_index;
      }
   }
   pulses[pulse_index] = highpulse;

   while (! (IRpin_PIN & _BV(IRpin))) { // got a low pulse
      lowpulse++;
      delayMicroseconds(RESOLUTION);
      if (((lowpulse >= MAXPULSE) && (pulse_index != 0))|| pulse_index == NUMPULSES)
{
        return pulse_index;
```

```
      }
    }
    pulses[pulse_index] = lowpulse;
    pulse_index++;

  }

}
```

Now that we have a way capture IR data, you can write your own program which uses these codes to do great things.

IR Remote Speaker

The demonstration uses a piezo speaker to make different beep tones depending based on four captured codes. This code is based on Wiring Diagram #3 - Trinket 5v with FTDI Friend (https://adafru.it/CcR).

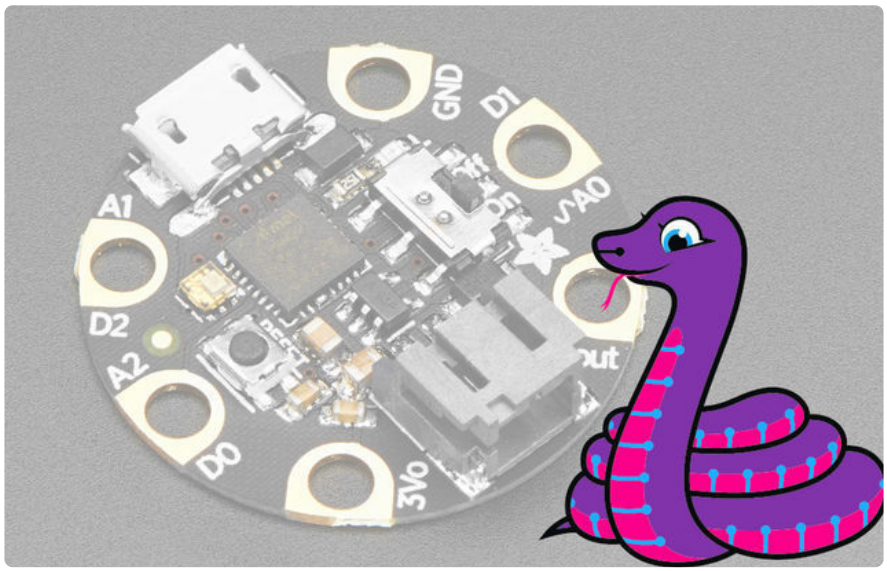
IR Codes in the code below were for a test remote and not the Adafruit remote. Use the codes you obtained in the Decoding IR Codes section your own project.

```
// SPDX-FileCopyrightText: 2018 Anne Barela and Limor Fried for Adafruit Industries
//
// SPDX-License-Identifier: MIT

/* Trinket/Gemma compatible IR read sketch
This sketch/program uses an Adafruit Trinket or Gemma
ATTiny85 based mini microcontroller and a PNA4602 or TSOP38238 to
read an IR code and perform a function.  In this
test program, tones are generated to a piezo speaker
but you can use codes to trigger any function you wish.
Based on Adafruit tutorial http://learn.adafruit.com/ir-sensor/using-an-ir-sensor
and http://learn.adafruit.com/trinket-gemma-mini-theramin-music-maker
*/

// We need to use the 'raw' pin reading methods because timing is very important
here
// and the digitalRead() procedure is slower!
#define IRpin_PIN  PINB // ATTiny85 had Port B pins
#define IRpin      2    // IR sensor - TSOP38238 on Pin GPIO #2 / D2
#define SPEAKERPIN 1    // Piezo speaker on Trinket/Gemma Pin GPIO #1/D1

#define MAXPULSE    5000  // the maximum pulse we'll listen for - 5 milliseconds
#define NUMPULSES    100  // max IR pulse pairs to sample
#define RESOLUTION     2  // // time between IR measurements

// we will store up to 100 pulse pairs (this is -a lot-, reduce if needed)
uint16_t pulses[100][2];   // pair is high and low pulse
```

```
uint16_t currentpulse = 0; // index for pulses we're storing
uint32_t irCode = 0;

void setup() {
  pinMode(IRpin, INPUT);       // Listen to IR receiver on Trinket/Gemma pin D2
  pinMode(SPEAKERPIN, OUTPUT); // Output tones on Trinket/Gemma pin D1
}

void loop() {

  irCode=listenForIR(); // Wait for an IR Code

  // Process the pulses to get our code
  for (int i = 0; i < 32; i++) {
    irCode=irCode<<1;
    if((pulses[i][0] * RESOLUTION)>0&&(pulses[i][0] * RESOLUTION)<500) {
      irCode|=0;
    } else {
      irCode|=1;
    }
  }
  if(irCode==0xe0c8A2DD)  {      // "1" on my remote, USE YOUR CODE HERE
    beep(SPEAKERPIN,400,500); }  // Make a 400 Hz beep
  else if (irCode==0xe0c8a3dc) { // "2", USE YOUR OWN HEX CODE HERE
    beep(SPEAKERPIN,500,500); }  // Make a 500 Hz beep
  else if (irCode==0xe0c8a41b) { // "3", USE YOUR OWN HEX CODE HERE
    beep(SPEAKERPIN,600, 500); } // Make a 600 Hz beep
  else if (irCode==0xe0c8a29d) { // "4", USE YOUR OWN HEX CODE HERE
    beep(SPEAKERPIN, 700, 500);  // Make a 700 Hz beep
  }

} // end loop


uint16_t listenForIR() {  // IR receive code
  currentpulse = 0;
  while (1) {
   unsigned int highpulse, lowpulse;  // temporary storage timing
   highpulse = lowpulse = 0; // start out with no pulse length

   while (IRpin_PIN & _BV(IRpin)) { // got a high pulse
      highpulse++;
      delayMicroseconds(RESOLUTION);
      if (((highpulse >= MAXPULSE) && (currentpulse != 0))|| currentpulse ==
NUMPULSES) {
        return currentpulse;
      }
   }
   pulses[currentpulse][0] = highpulse;

   while (! (IRpin_PIN & _BV(IRpin))) { // got a low pulse
      lowpulse++;
      delayMicroseconds(RESOLUTION);
      if (((lowpulse >= MAXPULSE) && (currentpulse != 0))|| currentpulse ==
NUMPULSES) {
        return currentpulse;
      }
   }
   pulses[currentpulse][1] = lowpulse;
   currentpulse++;
  }
}

// Generate a tone on speakerPin - Trinket/Gemma/ATTiny85 compatible
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);
            }
}
```

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** (https://adafru.it/z1B).

se directions are specific to the Gemma M0 and Trinket M0 boards. The inal GEMMA with an 8-bit AVR microcontroller doesn't run CircuitPython... hose boards, use the Arduino sketch on the "Arduino code" page of this e.

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

IR Reader

In this example we will read IR codes sent by the Adafruit Mini Remote Control (http://adafru.it/389) and output them to the python console in list form so they can be pasted into our next speaker example. This example is based on Wiring Diagram #1 - Gemma M0 or Trinket M0. (https://adafru.it/CcR)A step-by-step explanation of using IR codes with CircuitPython can be found in the IR Sensor tutorial (https://adafru.it/CcU).

This examples requires that an additional library be copied onto the Gemma M0 or Trinket M0 (under the 'lib' folder):

1. adafruit_irremote - for decoding the IR signals

The libraries can be obtained from the link below.

https://adafru.it/uap

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

import time

import adafruit_irremote
import board
import pulseio

IR_PIN = board.D2  # Pin connected to IR receiver.

print('IR listener')
print()
# Create pulse input and IR decoder.
pulses = pulseio.PulseIn(IR_PIN, maxlen=200, idle_state=True)
decoder = adafruit_irremote.GenericDecode()

# Loop waiting to receive pulses.
while True:
    # make sure pulses is empty
    # small delay for cleaner results
    pulses.clear()
```

```
    pulses.resume()
    time.sleep(.1)

    # Wait for a pulse to be detected.
    detected = decoder.read_pulses(pulses)

    # print the number of pulses detected
    # note: pulse count is an excellent indicator as to the quality of IR code
    # received.
    #
    # If you are expecting 67 each time (Adafruit Mini Remote Control #389)
    # and only receive 57 this will result in a incomplete listener

    print("pulse count: ", len(detected))

    # print in list form of the pulse duration in microseconds
    # typically starts with ~9,000 microseconds followed by a ~4,000
    # microseconds which is standard IR preamble

    print(detected)
    print()
```

IR Remote Speaker

This demonstration uses a piezo speaker to make different beep tones depending based on the four captured codes we acquired in the IR Reader example above. This code is based on Wiring Diagram #1 - Gemma M0 or Trinket M0 (https://adafru.it/CcR). There is a 20% 'fuzziness' factor used in matching the IR pulse codes.

This example requires that two additional libraries be copied onto the Gemma M0 or Trinket M0 (under the 'lib' folder):

1. adafruit_irremote - for decoding the IR signals
2. simpleio - used to send tones to the piezo speaker

Both libraries can be obtained here:

https://adafru.it/uap

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

"""
IR codes for button 0, 1, 2, 3 these were acquired from the
Adafruit Mini Remote Control #389
https://www.adafruit.com/product/389
stored as a 2D list
these codes were collected from running IR_reader.py
and watching the output on the REPL console
"""

import time
```

```
import adafruit_irremote
import board
import pulseio
import simpleio

speaker_pin = board.D0  # speaker connected to digital pin
ir_pin = board.D2  # pin connected to IR receiver.

fuzzyness = 0.2  # IR remote timing must be within 20% tolerance

button_presses = [

    [
        9073, 4504, 575, 589, 557, 581, 546, 592, 553, 585, 542, 595,
        550, 588, 552, 588, 544, 591, 549, 1671, 579, 1669, 572, 1675,
        577, 1671, 570, 1678, 574, 1673, 579, 559, 572, 1676, 575, 589,
        542, 596, 546, 1674, 577, 1671, 571, 567, 574, 590, 552, 586,
        545, 593, 548, 1672, 569, 1678, 574, 591, 550, 588, 550, 1670,
        575, 1673, 579, 1671, 570, 1675, 586
    ],

    [
        9075, 4498, 571, 566, 575, 562, 569, 569, 573, 564, 577, 587, 544,
        567, 577, 587, 551, 586, 545, 1675, 577, 1671, 575, 1672, 574, 1674,
        580, 1668, 572, 1675, 576, 589, 542, 1678, 574, 564, 577, 588, 543,
        594, 547, 591, 551, 1669, 572, 592, 553, 585, 542, 595, 550, 1671,
        577, 1670, 571, 1677, 575, 1673, 569, 576, 565, 1675, 577, 1670, 572,
        1676, 575
    ],

    [
        9070, 4505, 574, 563, 578, 559, 572, 566, 575, 562, 569, 569, 573, 564,
        577, 561, 570, 567, 575, 1674, 578, 1669, 577, 1670, 577, 1670, 571,
        1677, 575, 1672, 569, 569, 573, 1674, 577, 1671, 571, 566, 575, 562,
        569, 575, 566, 1675, 577, 560, 571, 567, 574, 563, 568, 569, 573, 1675,
        576, 1671, 571, 1681, 571, 562, 569, 1679, 575, 1672, 578, 1670, 570
    ],

    [
        9080, 4500, 569, 568, 573, 564, 577, 561, 570, 567, 574, 564, 578, 559,
        577, 561, 575, 562, 579, 1669, 572, 1675, 577, 1671, 570, 1677, 575,
        1673, 578, 1672, 570, 570, 571, 1671, 574, 564, 574, 1678, 573, 560,
        571, 568, 574, 1671, 570, 568, 573, 564, 577, 561, 570, 1677, 575, 563,
        578, 1669, 572, 1676, 576, 561, 570, 1677, 574, 1674, 578, 1669, 572
    ]

]


def fuzzy_pulse_compare(received):
    # Did we receive a full IR code?
    # Should be 67 timings for this remote
    if len(received) == len(button_presses[0]):

        # compare received IR code with our stored button_press list
        # remote control button codes for : [0-3]
        for b_index, button_press in enumerate(button_presses):

            # compare individual timings for each IR code
            # confirm that every entry is within fuzzyness 20% accuracy
            for i, press in enumerate(button_press):

                threshold = int(press * fuzzyness)

                if abs(press - received[i]) < threshold:
                    match_count[b_index] += 1
```

```
def play_tone():
    """ half second tones based on button selection [0-3] """
    if remote_control_press == 0:
        simpleio.tone(speaker_pin, 400, .5)  # 400Hz beep, 1/2 sec

    elif remote_control_press == 1:
        simpleio.tone(speaker_pin, 500, .5)  # 500Hz beep, 1/2 sec

    elif remote_control_press == 2:
        simpleio.tone(speaker_pin, 600, .5)  # 600Hz beep, 1/2 sec

    elif remote_control_press == 3:
        simpleio.tone(speaker_pin, 700, .5)  # 700Hz beep, 1/2 sec


# Create pulse input and IR decoder.
pulses = pulseio.PulseIn(ir_pin, maxlen=200, idle_state=True)
decoder = adafruit_irremote.GenericDecode()

# Loop waiting to receive pulses.
while True:
    # total count of IR code matches for each button {0, 1, 2, 3}
    match_count = [0] * len(button_presses)

    # make sure pulses is empty
    pulses.clear()
    pulses.resume()
    time.sleep(.1)

    # Wait for a pulse to be detected.
    detected = decoder.read_pulses(pulses)

    fuzzy_pulse_compare(detected)

    # confirm that we know this button
    # received IR code compared with saved button_presses
    # 100% match (+/- fuzziness)
    # otherwise we don't know this button pressed
    if max(match_count) == len(button_presses[0]):
        remote_control_press = match_count.index(max(match_count))
        play_tone()
        print(match_count.index(max(match_count)))
    else:
        print("unknown button")
```

Use and Going Further

Video

Here's a short clip demonstrating the project

Going Further

The IR sensor only requires one digital input pin. On a Trinket, this leaves you 4 pins for other functions, on the Gemma, two pins. This gives you maximum flexibility for using an IR signal to trigger other devices like different Neopixel effects. See the Adafruit NeoPixel Überguide (https://adafru.it/cEz) on how to connect Neopixels.

An IR sensor makes an inexpensive, compact (and nearly magical) input device to

trigger actions in any project.