Adafruit pIRkey

Created by lady ada

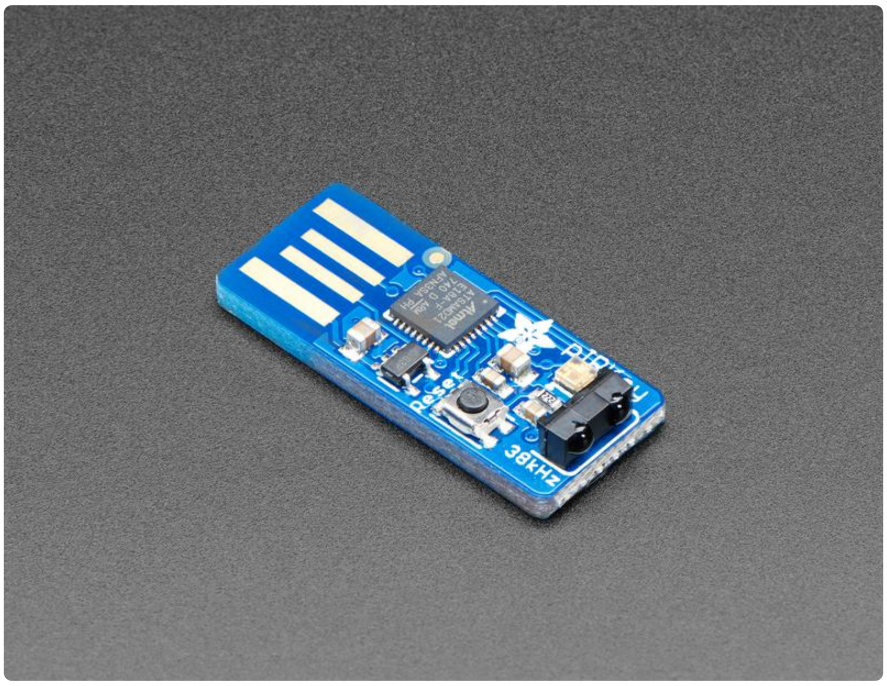
https://learn.adafruit.com/adafruit-pirkey-python-programmable-infrared-usb-adapter

Last updated on 2023-08-29 03:43:24 PM EDT

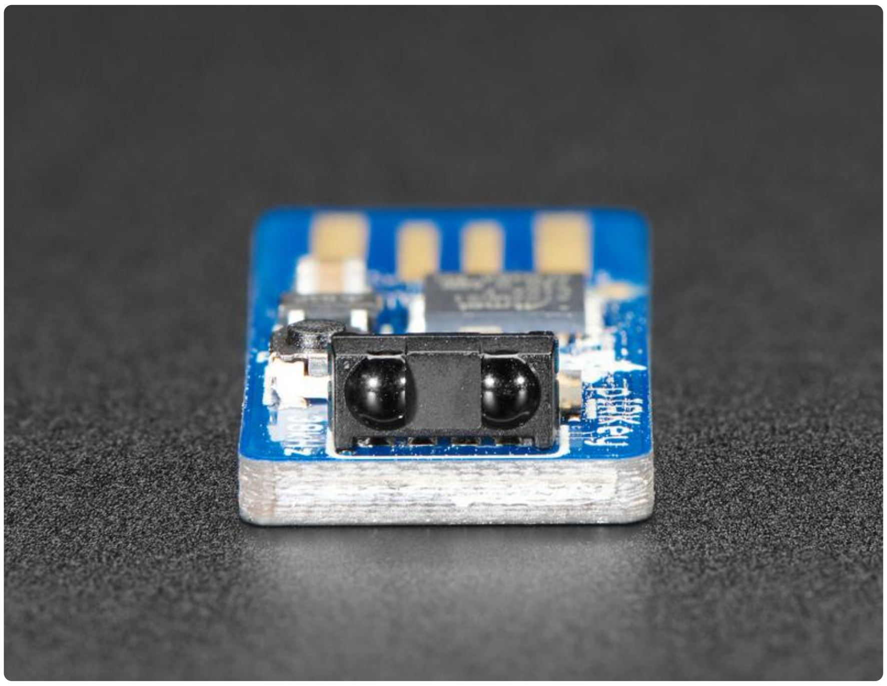
Table of Contents

Overview

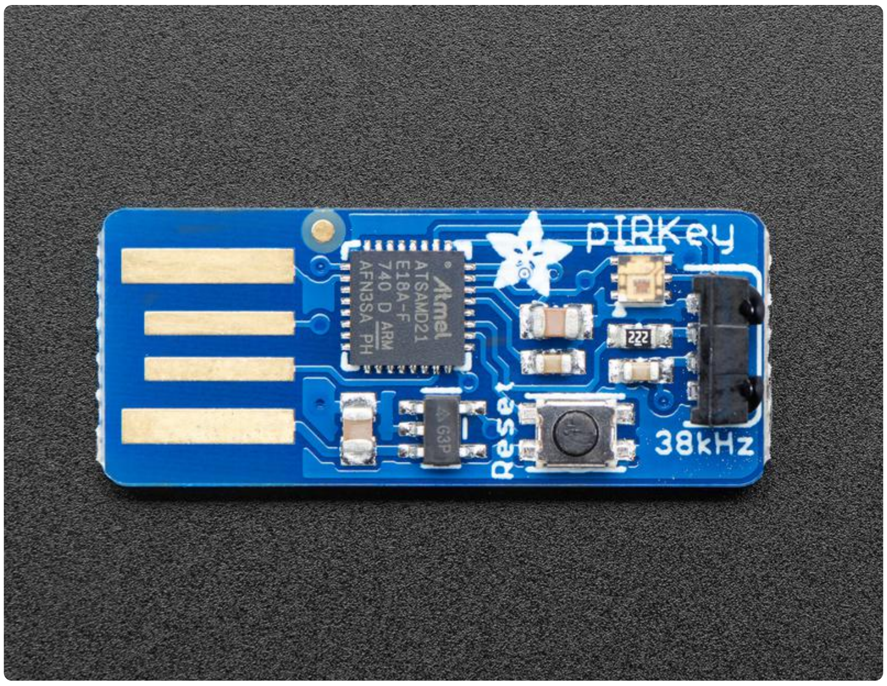
The pIRkey adds an IR remote receiver to any computer, laptop, tablet...any computer or device with a USB port that can use a keyboard. This little board slides into any USB A port, and shows up as an every-day USB keyboard. The onboard ATSAMD21 microcontroller listens for IR remote signals and converts them to keypresses, mouse movements, or even USB serial output.



Infrared is our favorite wireless protocol - no antennas, certifications, pairings, passwords, or special tools required. Works everywhere in the world and very intuitive - everyone's got an IR remote in their home! [Our original IRkey ()]() was a small USB-pluggable microcontroller board with an IR receiver, an Attiny85 microcontroller and indicator LED. When certain remote control commands were received, the IRkey would send corresponding keyboard presses. It was great, but was not easy to customize - you had to use the remote we sold it work.

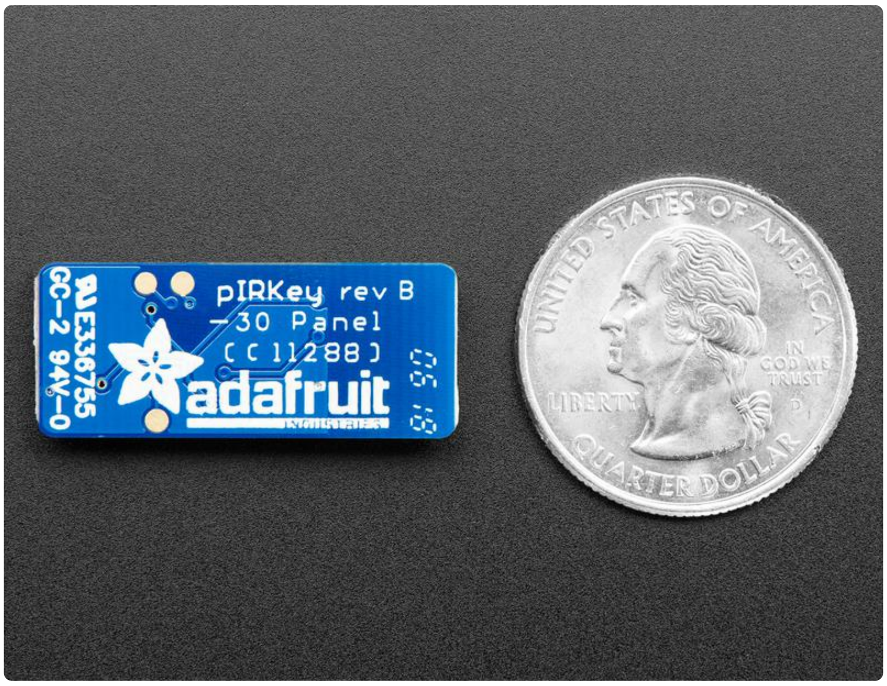


The pIRkey is an improvement on our original IRkey product, by adding a p for python. Now that we have CircuitPython available for the tiny ATSAMD21E processor, we swapped it for the ATtiny85, giving a huge boost in power and a working Python interpreter on board as well. This means its super easy to reprogram, customize or adapt it to whatever Infrared-reading needs you may have

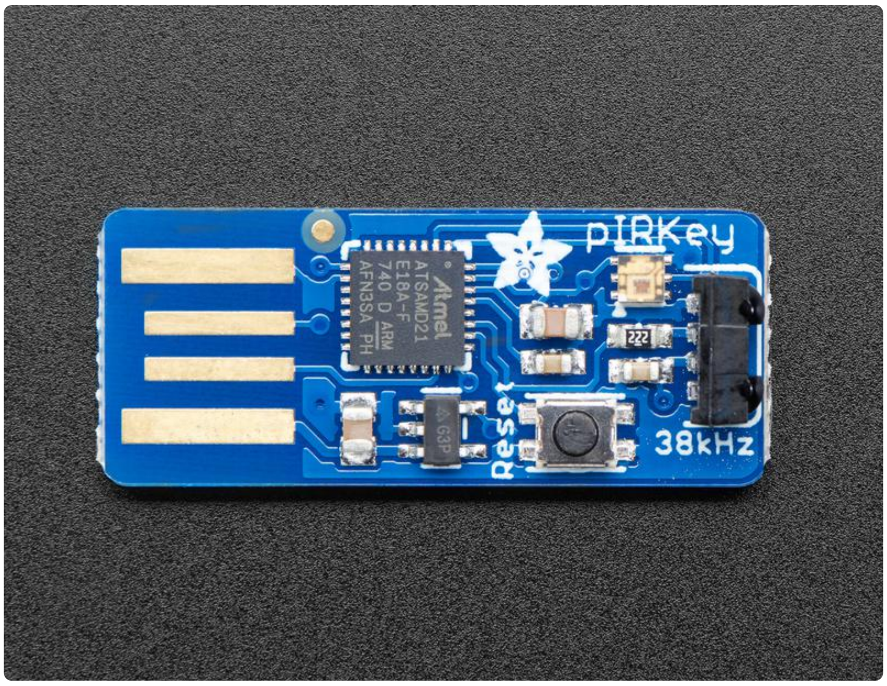


When you plug it in, the pIRkey shows up as a triple device: USB disk drive to store code, USB serial for debugging and Python interactive command line, and USB keyboard/mouse that can transmit keypresses or mouse movements.

By default we ship with some very simple example code to read NEC remotes but you can use any remote that has about 38KHz output frequency which is 99% of remote controls. Here's some ideas: you could use pIRkey to remotely start/stop a program, shut down a computer, control a smart phone or tablet mounted far away, make adaptive controls, etc.



Pinouts



pIRKey is pretty simple so you don't actually have a lot of pinout requirements!

USB Connector

On the left is a PCB-mount USB connector, just plug it right into any USB-A port. the 4 gold plated pads have Ground, D+, D- and 5V power. The 5V power is regulated down and used to power the pIRKey. The D+/D- is what the onboard chip uses to send/receive data

Microcontroller

The chip used here is the ATSAMD21E18 - the same chip in our Trinket M0 and Gemma M0. It has 256KB of flash, 32 KB of RAM and runs at 48MHz. We pre-load CircuitPython but you could also use Arduino if you like, just select Trinket M0 as the board type.

Reset Button

You can reset the board or put it into bootloader mode using the Reset button. One click resets. Double-click puts into bootloader mode. In bootloader mode, the small DotStar LED will turn green on successful USB enumeration, or red on failure

DotStar LED

We put a small RGB LED on board. This is great for helping the user know if the IR command was read properly, what the status is, or changing modes.

In CircuitPython you can communicate with the DotStar over the `board.APA102_MOSI` and `board.APA102_SCK` pins

Infrared Receiver

At the end is a lensed IR receiver module, it will read IR light, amplify if necessary and filter out the 38 KHz sub-carrier so you just get pulses when light is detected, making it a lot easier on the pIRkey!

Note that even though it is tuned to 38 KHz, you can use about 30 KHz to 46 KHz without too much difficulty, its not a very precise filter on purpose since low cost IR remotes have a lot of drift.

In CircuitPython you can read data over the `board.REMOTEIN` pin

---

pIRKey & CircuitPython

The pIRKey comes with CircuitPython installed on it.

tl;dr? It's extremely fast and easy to program and customize your pIRkey behavior using CircuitPython - much easier and faster than trying to use Arduino as we do not have to compile every time. All you have to do is edit the file on the little disk drive. Also, CircuitPython has native HID keyboard/mouse support so its perfect for pIRkeys simple IR in -> Keyboard out setup.

What is CircuitPython?

CircuitPython is a programming language designed to simplify experimenting and learning to program on low-cost microcontroller boards. It makes getting started easier than ever with no upfront desktop downloads needed. Once you get your board set up, open any text editor, and get started editing code. It's that simple.

Why Does This Product Use CircuitPython?

CircuitPython is designed to run on microcontroller boards. A microcontroller board is a board with a microcontroller chip that's essentially an itty-bitty all-in-one computer. The board you're holding is a microcontroller board! CircuitPython is easy to use because all you need is that little board, a USB cable, and a computer with a USB connection. But that's only the beginning.

Other reasons to use CircuitPython include:

- You want to get up and running quickly. Create a file, edit your code, save the file, and it runs immediately. There is no compiling, no downloading and no uploading needed.
- You're new to programming. CircuitPython is designed with education in mind. It's easy to start learning how to program and you get immediate feedback from the board.
- Easily update your code. Since your code lives on the disk drive, you can edit it whenever you like, you can also keep multiple files around for easy experimentation.
- The serial console and REPL. These allow for live feedback from your code and interactive programming.
- File storage. The internal storage for CircuitPython makes it great for data-logging, playing audio clips, and otherwise interacting with files.
- Strong hardware support. There are many libraries and drivers for sensors, breakout boards and other external components.
- It's Python! Python is the fastest-growing programming language. It's taught in schools and universities. CircuitPython is almost-completely compatible with Python. It simply adds hardware support.

I'm Intrigued! Please Sign Me Up For Your Newsletter

Since pIRKey is a tool not a full dev-board, we're not going to put a full CircuitPython tutorial here. Instead we recommend you read our excellent guides:

- Adafruit's Welcome To CircuitPython https://learn.adafruit.com/welcome-to-circuitpython/ ()
- Adafruit's CircuitPython Essentials Guide https://learn.adafruit.com/circuitpython-essentials ()

Getting Started

The biggest challenge with making a USB device that reads IR commands is that there's a ton of different IR remote types and encodings. Even if we pre-programmed it to recognize the most popular brands, we could miss support for a remote you have.

For that reason, we decided to have decoding support for NEC remotes (the most popular encoding we've encountered) and then for other remotes, show you how to set up custom decoded types

Let's get started!

Step 0. Install Windows 7 Drivers

If you're using Windows 7, use the link below to download the driver package. You will not need to install drivers on Mac, Linux or Windows 10.

Download Adafruit Windows 7 Driver Installer

Step 1. Install Mu

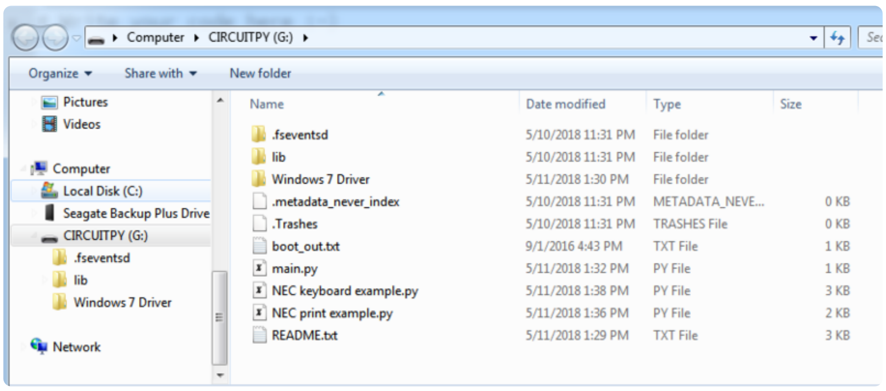
Mu is a simple code editor that works with the Adafruit CircuitPython boards. It's written in Python and works on Windows, MacOS, Linux and Raspberry Pi. The serial console is built right in so you get immediate feedback from your board's serial output!

Even if you plan on using your pIRKey on a tablet or phone, you still need to do the programming/customization part on a desktop computer

Follow our guide to install Mu on Windows/Mac/Linux

Step 2. Plug in pIRKey and open Mu REPL

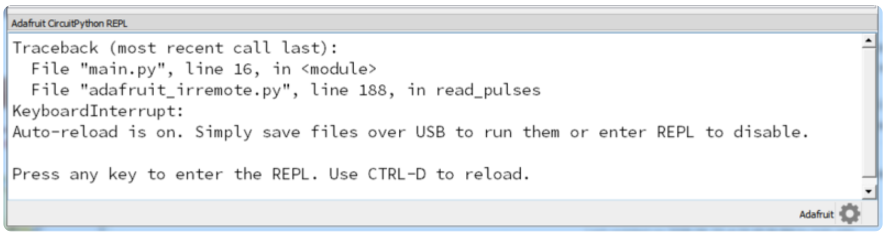
Now you're ready! Plug in the pIRKey into your computer. If you are using Windows 7, make sure you install the drivers above. After the pIRKey is plugged in, you'll get a disk drive with the code and some examples & documentation.



Name	Date modified	Type	Size
.fseventsd	5/10/2018 11:31 PM	File folder	
lib	5/10/2018 11:31 PM	File folder	
Windows 7 Driver	5/11/2018 1:30 PM	File folder	
.metadata_never_index	5/10/2018 11:31 PM	METADATA_NEVE...	0 KB
.Trashes	5/10/2018 11:31 PM	TRASHES File	0 KB
boot_out.txt	9/1/2016 4:43 PM	TXT File	1 KB
main.py	5/11/2018 1:32 PM	PY File	1 KB
NEC keyboard example.py	5/11/2018 1:38 PM	PY File	3 KB
NEC print example.py	5/11/2018 1:36 PM	PY File	2 KB
README.txt	5/11/2018 1:29 PM	TXT File	3 KB

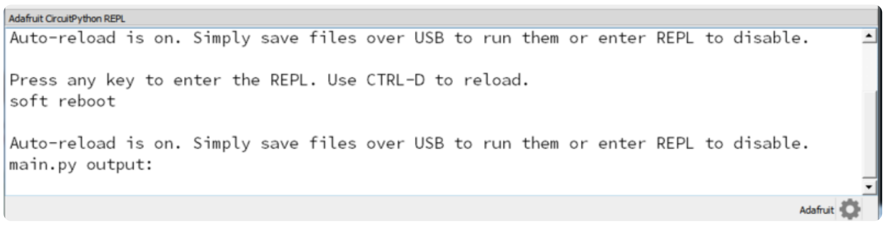
Now start Mu.

Select Adafruit Mode if you're asked. Then click the REPL button, you should see something like the following:



```
Traceback (most recent call last):
  File "main.py", line 16, in <module>
  File "adafruit_irremote.py", line 188, in read_pulses
KeyboardInterrupt:
Auto-reload is on. Simply save files over USB to run them or enter REPL to disable.

Press any key to enter the REPL. Use CTRL-D to reload.
```

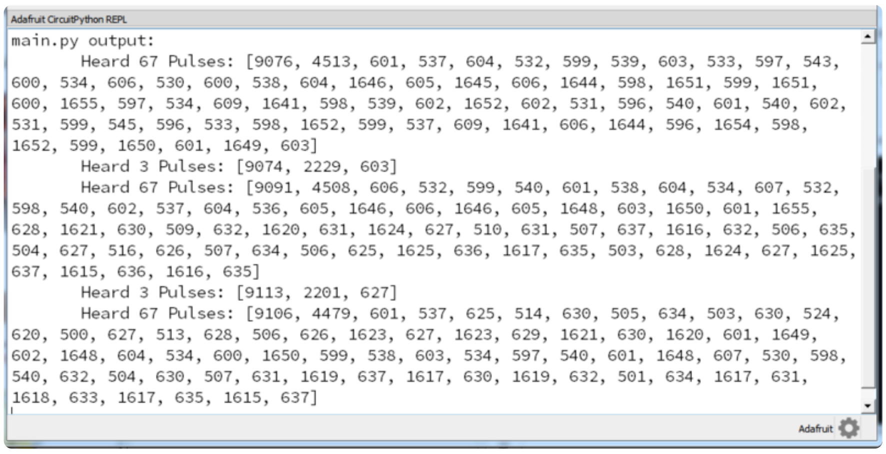
Press Control D to restart the program



```
Auto-reload is on. Simply save files over USB to run them or enter REPL to disable.

Press any key to enter the REPL. Use CTRL-D to reload.
soft reboot

Auto-reload is on. Simply save files over USB to run them or enter REPL to disable.
main.py output:
```

Now find your IR remote control and click some buttons. You'll see the received data printed out!

These are the raw data pulses that correspond to each button. The number of pulses and the quantity in each array may vary



```
main.py output:
        Heard 67 Pulses: [9076, 4513, 601, 537, 604, 532, 599, 539, 603, 533, 597, 543,
600, 534, 606, 530, 600, 538, 604, 1646, 605, 1645, 606, 1644, 598, 1651, 599, 1651,
600, 1655, 597, 534, 609, 1641, 598, 539, 602, 1652, 602, 531, 596, 540, 601, 540, 602,
531, 599, 545, 596, 533, 598, 1652, 599, 537, 609, 1641, 606, 1644, 596, 1654, 598,
1652, 599, 1650, 601, 1649, 603]
        Heard 3 Pulses: [9074, 2229, 603]
        Heard 67 Pulses: [9091, 4508, 606, 532, 599, 540, 601, 538, 604, 534, 607, 532,
598, 540, 602, 537, 604, 536, 605, 1646, 606, 1646, 605, 1648, 603, 1650, 601, 1655,
628, 1621, 630, 509, 632, 1620, 631, 1624, 627, 510, 631, 507, 637, 1616, 632, 506, 635,
504, 627, 516, 626, 507, 634, 506, 625, 1625, 636, 1617, 635, 503, 628, 1624, 627, 1625,
637, 1615, 636, 1616, 635]
        Heard 3 Pulses: [9113, 2201, 627]
        Heard 67 Pulses: [9106, 4479, 601, 537, 625, 514, 630, 505, 634, 503, 630, 524,
620, 500, 627, 513, 628, 506, 626, 1623, 627, 1623, 629, 1621, 630, 1620, 601, 1649,
602, 1648, 604, 534, 600, 1650, 599, 538, 603, 534, 597, 540, 601, 1648, 607, 530, 598,
540, 632, 504, 630, 507, 631, 1619, 637, 1617, 630, 1619, 632, 501, 634, 1617, 631,
1618, 633, 1617, 635, 1615, 637]
```

For example, on our NEC remote, the Play Pause button will send the following

`[9072, 4512, 627, 511, 647, 490, 620, 516, 626, 511, 630, 507, 623, 515, 627, 509, 601, 537, 625, 1624, 627, 1623, 628, 1626, 629, 1618, 629, 1621, 630, 1620, 631, 506, 625, 1625, 638, 1612, 658, 479, 714, 423, 625, 513, 628, 508, 633, 504, 627, 510, 631, 506, 656, 481, 629, 1621, 630, 1620, 651, 1601, 630, 1618, 633, 1617, 634, 1616, 626, 1624, 625]`

And the Volume Up button will send

`[9068, 4510, 601, 540, 601, 531, 600, 537, 605, 532, 599, 538, 603, 533, 599, 538, 602, 535, 596, 1652, 599, 1650, 601, 1649, 602, 1648, 608, 1642, 604, 1649, 602, 532, 599, 1650, 601, 541, 621, 1626, 594, 542, 631, 505, 630, 508, 629, 511, 630, 502, 628, 508, 633, 1621, 603, 533, 625, 1620, 632, 1618, 633, 1617, 639, 1610, 637, 1613, 628, 1622, 629]`

Note these look the same but they're not the exact same codes

Detecting & Matching Codes

Now you have a selection of codes you want to match. We'll be using the two codes above but use whatever your remote output!

Save this sketch to your pIRkey disk drive, named code.py

```
# SPDX-FileCopyrightText: 2018 Limor Fried for Adafruit Industries
#
# SPDX-License-Identifier: MIT

import board
import pulseio
import adafruit_dotstar
import adafruit_irremote

led = adafruit_dotstar.DotStar(board.APA102_SCK, board.APA102_MOSI, 1)
decoder = adafruit_irremote.GenericDecode()
pulsein = pulseio.PulseIn(board.REMOTEIN, maxlen=200, idle_state=True)

# Expected pulse, pasted in from previous recording REPL session:
key1_pulses = [0]  # PUT YOUR PULSECODES HERE!
key2_pulses = [1]  # PUT YOUR PULSECODES HERE!

print('IR listener')
# Fuzzy pulse comparison function:
def fuzzy_pulse_compare(pulse1, pulse2, fuzzyness=0.2):
    if len(pulse1) != len(pulse2):
        return False
    for i in range(len(pulse1)):
        threshold = int(pulse1[i] * fuzzyness)
        if abs(pulse1[i] - pulse2[i]) > threshold:
            return False
    return True

# Create pulse input and IR decoder.
pulsein.clear()
pulsein.resume()

# Loop waiting to receive pulses.
while True:
    led[0] = (0, 0, 0)   # LED off
    # Wait for a pulse to be detected.
    pulses = decoder.read_pulses(pulsein)
    led[0] = (0, 0, 100) # flash blue

    print("\tHeard", len(pulses), "Pulses:", pulses)

    # Got a pulse set, now compare.
    if fuzzy_pulse_compare(key1_pulses, pulses):
        print("****** KEY 1 DETECTED! ******")

    if fuzzy_pulse_compare(key2_pulses, pulses):
        print("****** KEY 2 DETECTED! ******")
```

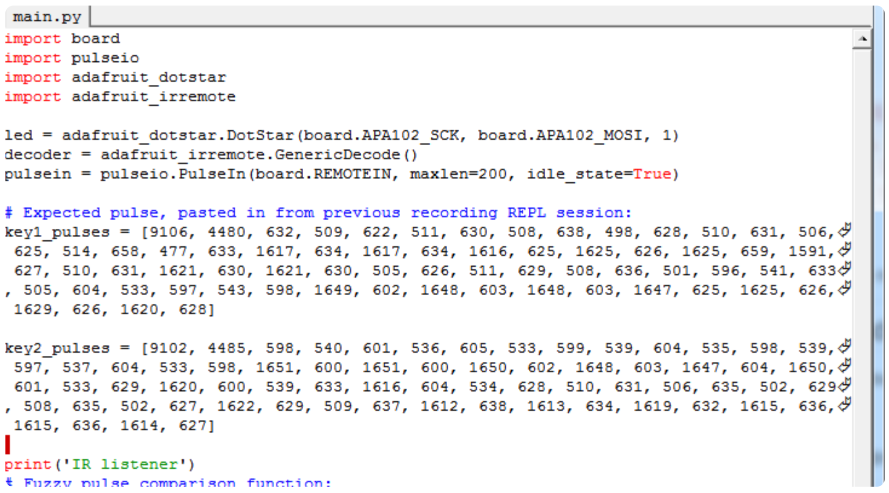
When you save the new python example, the REPL will automatically reload this example.

Now find these two lines:

```
key1_pulses = [0] # PUT YOUR PULSECODES HERE!
key2_pulses = [1] # PUT YOUR PULSECODES HERE!
```

And replace the `[0]` and `[1]` with the two pulse sets you detected before. Or you can use the REPL to press other keys until you get the ones you want.

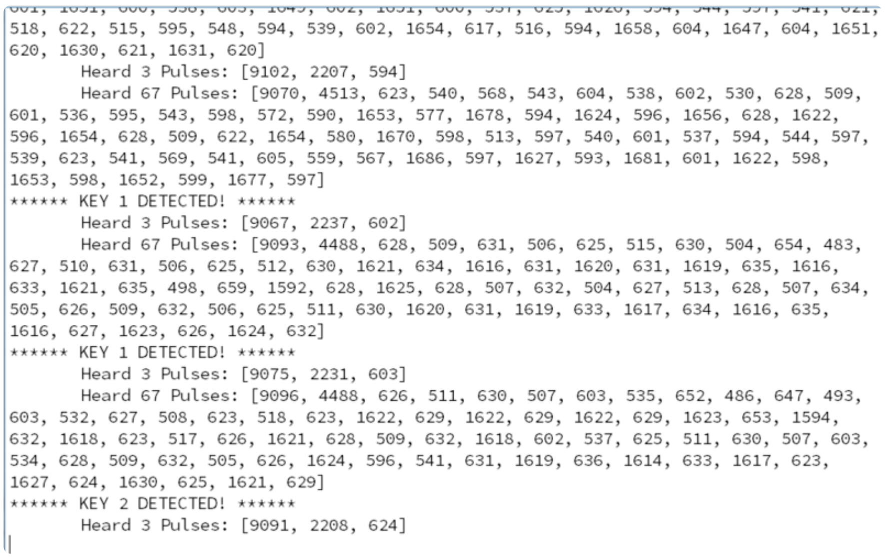
Ours looks like this when done:



```
main.py
import board
import pulseio
import adafruit_dotstar
import adafruit_irremote

led = adafruit_dotstar.DotStar(board.APA102_SCK, board.APA102_MOSI, 1)
decoder = adafruit_irremote.GenericDecode()
pulsein = pulseio.PulseIn(board.REMOTEIN, maxlen=200, idle_state=True)

# Expected pulse, pasted in from previous recording REPL session:
key1_pulses = [9106, 4480, 632, 509, 622, 511, 630, 508, 638, 498, 628, 510, 631, 506,
 625, 514, 658, 477, 633, 1617, 634, 1617, 634, 1616, 625, 1625, 626, 1625, 659, 1591,
 627, 510, 631, 1621, 630, 1621, 630, 505, 626, 511, 629, 508, 636, 501, 596, 541, 633
, 505, 604, 533, 597, 543, 598, 1649, 602, 1648, 603, 1648, 603, 1647, 625, 1625, 626,
 1629, 626, 1620, 628]

key2_pulses = [9102, 4485, 598, 540, 601, 536, 605, 533, 599, 539, 604, 535, 598, 539,
 597, 537, 604, 533, 598, 1651, 600, 1651, 600, 1650, 602, 1648, 603, 1647, 604, 1650,
 601, 533, 629, 1620, 600, 539, 633, 1616, 604, 534, 628, 510, 631, 506, 635, 502, 629
, 508, 635, 502, 627, 1622, 629, 509, 637, 1612, 638, 1613, 634, 1619, 632, 1615, 636,
 1615, 636, 1614, 627]

print('IR listener')
# Fuzzy pulse comparison function:
```

Save to cause the CircuitPython code to reload the new code.

Now when you press those same remote control keys you will get DETECTED displays!



```
518, 622, 515, 595, 548, 594, 539, 602, 1654, 617, 516, 594, 1658, 604, 1647, 604, 1651,
620, 1630, 621, 1631, 620]
        Heard 3 Pulses: [9102, 2207, 594]
        Heard 67 Pulses: [9070, 4513, 623, 540, 568, 543, 604, 538, 602, 530, 628, 509,
601, 536, 595, 543, 598, 572, 590, 1653, 577, 1678, 594, 1624, 596, 1656, 628, 1622,
596, 1654, 628, 509, 622, 1654, 580, 1670, 598, 513, 597, 540, 601, 537, 594, 544, 597,
539, 623, 541, 569, 541, 605, 559, 567, 1686, 597, 1627, 593, 1681, 601, 1622, 598,
1653, 598, 1652, 599, 1677, 597]
****** KEY 1 DETECTED! ******
        Heard 3 Pulses: [9067, 2237, 602]
        Heard 67 Pulses: [9093, 4488, 628, 509, 631, 506, 625, 515, 630, 504, 654, 483,
627, 510, 631, 506, 625, 512, 630, 1621, 634, 1616, 631, 1620, 631, 1619, 635, 1616,
633, 1621, 635, 498, 659, 1592, 628, 1625, 628, 507, 632, 504, 627, 513, 628, 507, 634,
505, 626, 509, 632, 506, 625, 511, 630, 1620, 631, 1619, 633, 1617, 634, 1616, 635,
1616, 627, 1623, 626, 1624, 632]
****** KEY 1 DETECTED! ******
        Heard 3 Pulses: [9075, 2231, 603]
        Heard 67 Pulses: [9096, 4488, 626, 511, 630, 507, 603, 535, 652, 486, 647, 493,
603, 532, 627, 508, 623, 518, 623, 1622, 629, 1622, 629, 1622, 629, 1623, 653, 1594,
632, 1618, 623, 517, 626, 1621, 628, 509, 632, 1618, 602, 537, 625, 511, 630, 507, 603,
534, 628, 509, 632, 505, 626, 1624, 596, 541, 631, 1619, 636, 1614, 633, 1617, 623,
1627, 624, 1630, 625, 1621, 629]
****** KEY 2 DETECTED! ******
        Heard 3 Pulses: [9091, 2208, 624]
```

You can add just about as many codes as you like, from any kind of remote. Just keep adding `keyn_pulses = [....]` lines, make sure each key has a different '`n`' - we like numbering them but whatever makes them unique will work fine. Then at the bottom add another

```
if fuzzy_pulse_compare(keyn_pulses, pulses):
    print("****** KEY n DETECTED! ******")
```

For each key you want to detect, again match the 'n' to the pulses you defined at the top

Adding Keyboard Output

It's tougher to debug when you have the pIRkey typing stuff into your windows so get the basic detection script above working well and detecting all the keys you want, then you can add keyboard output.

For more details on HID keyboard support, check out the CircuitPython Essentials HID Keyboard & Mouse () guide.

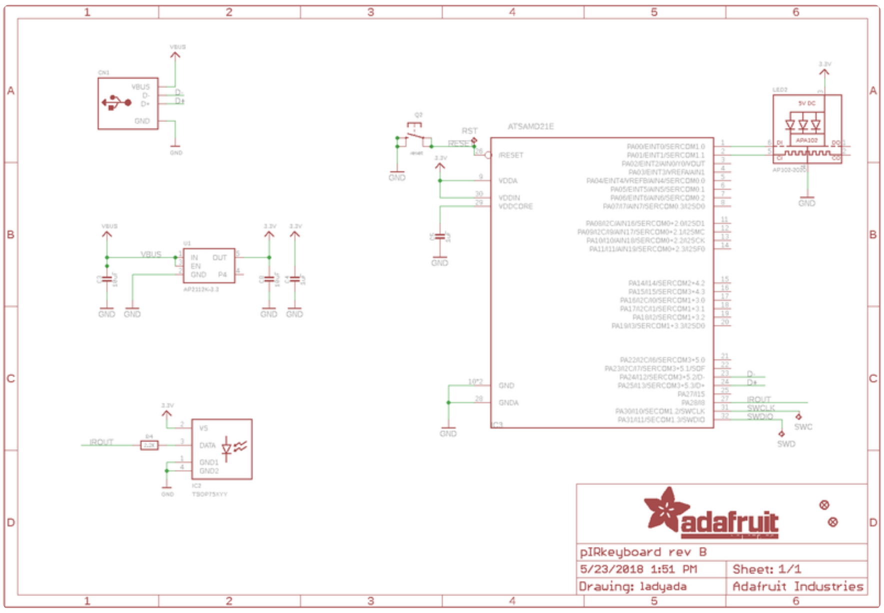
Downloads

Files

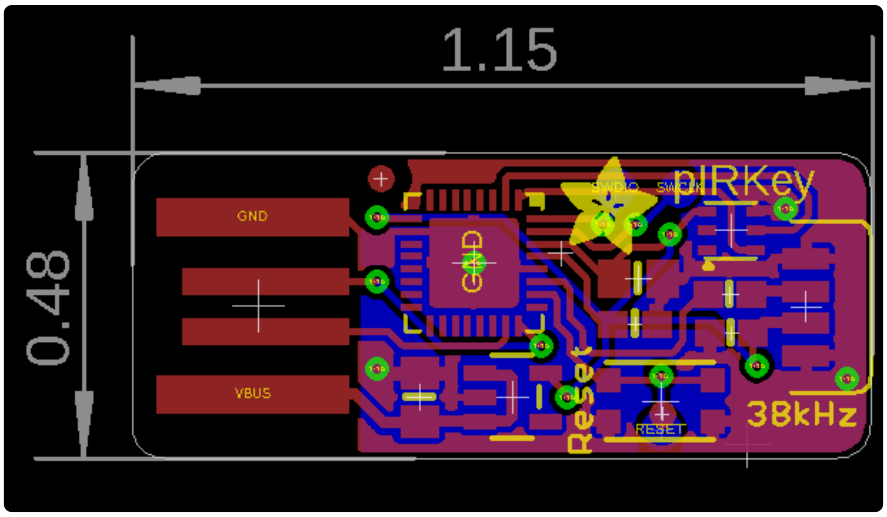
- EagleCAD PCB files on GitHub ()
- TSOP75238 Infrared receiver Datasheet ()
- Adafruit Windows 7 Driver bundle ()

Schematic

Click to embiggen



Fabrication Print


1.15
0.48
GND
VBUS
GND
SWDIO
SWCLK
pIRKey
Reset
RESET
38kHz