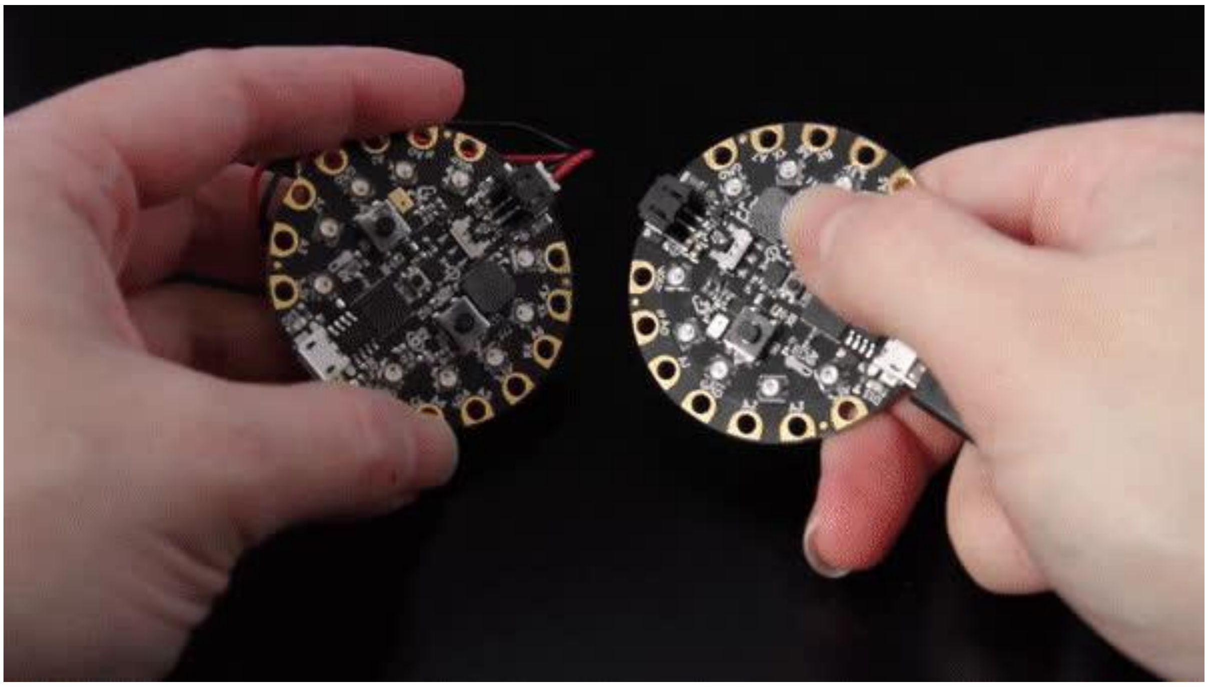
Infrared Receive and Transmit with Circuit Playground Express

Created by Kattni Rembor



https://learn.adafruit.com/infrared-ir-receive-transmit-circuit-playground-express-circuit-python

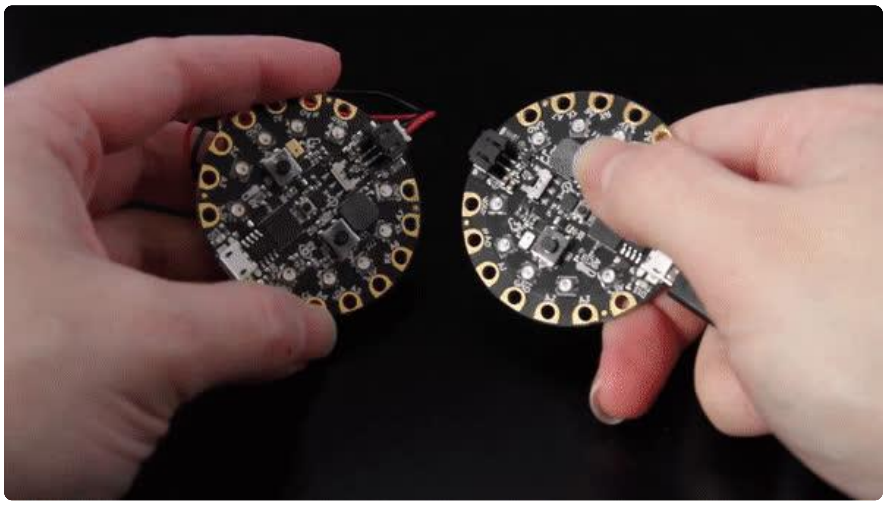
Last updated on 2023-08-29 03:48:31 PM EDT

Table of Contents

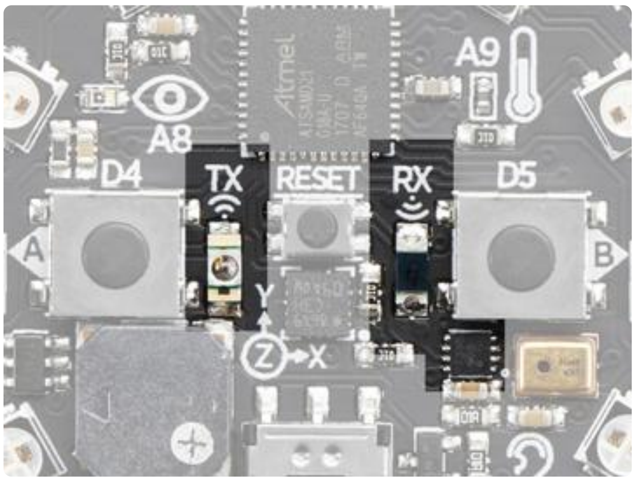
Overview

The Circuit Playground Express is an amazing little board with tons of sensors and things built in, including an infrared transmitter and an infrared receiver. This guide will show you how to use CircuitPython to send simple messages between two Circuit Playground Expresses using infrared!

This guide expects that you have two Circuit Playground Expresses, as we will be using IR to communicate between them. You can get wireless communications without antennas, pairing or passwords.



Infrared (IR) is invisible to the naked eye which makes it great for wireless communication. IR communication is line-of-sight (the sensor must be pointed towards the receiver) and has about a 10-20 meter range (optimally). It's good for sending short amounts of data. IR remotes use infrared for communicating with their targets, for example your television. For more information about IR, check out this article ().



The IR transmitter and receiver on the Circuit Playground Express can be found near the center of the board.

The transmitter is labeled TX and is on the left side of the reset button, to the right of button A. The receiver is labeled RX and is on the right side of the reset button, to the left of button B.

IR Test with Remote

The first thing we're going to do is test IR receive with a NEC remote. NEC is a electronics manufacturer, one of several, that defined their own IR coding scheme which has also been used by other folks in products. You can try any remotes you have sitting around the house (although they might use an encoding other than NEC). We have this handy little one () available in the store which we're going to use for our test.

Mini Remote Control
This little remote control would be handy for controlling a robot or other project from across the room. It has 21 buttons and a layout we thought was handy: directional buttons and...
https://www.adafruit.com/product/389

Copy the following code to your code.py:

```
# SPDX-FileCopyrightText: 2018 Kattni Rembor for Adafruit Industries
#
# SPDX-License-Identifier: MIT

import pulseio
import board
import adafruit_irremote

# Create a 'pulseio' input, to listen to infrared signals on the IR receiver
pulsein = pulseio.PulseIn(board.IR_RX, maxlen=120, idle_state=True)
# Create a decoder that will take pulses and turn them into numbers
decoder = adafruit_irremote.GenericDecode()

while True:
    pulses = decoder.read_pulses(pulsein)
    try:
        # Attempt to convert received pulses into numbers
        received_code = decoder.decode_bits(pulses)
    except adafruit_irremote.IRNECRepeatException:
        # We got an unusual short code, probably a 'repeat' signal
        # print("NEC repeat!")
        continue
    except adafruit_irremote.IRDecodeException as e:
        # Something got distorted or maybe its not an NEC-type remote?
        # print("Failed to decode: ", e.args)
        continue

    print("NEC Infrared code received: ", received_code)
    if received_code == [255, 2, 255, 0]:
        print("Received NEC Vol-")
    if received_code == [255, 2, 127, 128]:
        print("Received NEC Play/Pause")
    if received_code == [255, 2, 191, 64]:
        print("Received NEC Vol+")
```

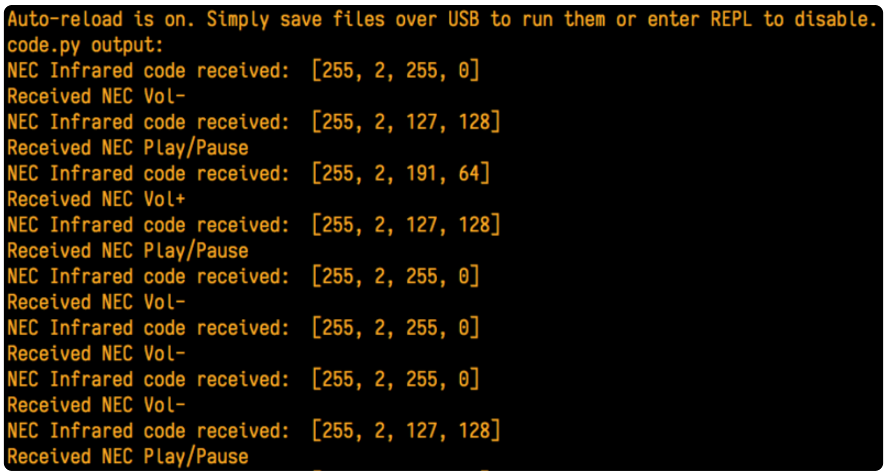
We create the `pulsein` object to listen for infrared signals on the IR receiver. Then we create the `decoder` object to take the pulses and turn them into numbers.

Then we take the `decoder` object and attempt to convert the received pulses into numbers. There's two errors we check for and tell the code to continue running if they're encountered.

One possible decoding error is an unusually short code, which is probably an NEC repeat signal. If you hold down a remote button, the remote control may 'save effort and time' by sending a short code that means "keep doing that". For example, holding down the volume button to quickly increase or decrease the volume on a TV. We don't handle those repeat codes in this project, we're only looking for unique button presses

The second possible decoding error is when it fails to decode, which can mean the signal got distorted or you're not using a NEC remote.

Then we print the code we receive from the remote. If we receive the codes from the first three buttons on the remote, we print which button was pressed.



```
Auto-reload is on. Simply save files over USB to run them or enter REPL to disable.
code.py output:
NEC Infrared code received:  [255, 2, 255, 0]
Received NEC Vol-
NEC Infrared code received:  [255, 2, 127, 128]
Received NEC Play/Pause
NEC Infrared code received:  [255, 2, 191, 64]
Received NEC Vol+
NEC Infrared code received:  [255, 2, 127, 128]
Received NEC Play/Pause
NEC Infrared code received:  [255, 2, 255, 0]
Received NEC Vol-
NEC Infrared code received:  [255, 2, 255, 0]
Received NEC Vol-
NEC Infrared code received:  [255, 2, 255, 0]
Received NEC Vol-
NEC Infrared code received:  [255, 2, 127, 128]
Received NEC Play/Pause
```

If you're using your own remote, you can check the serial console to find the codes you're receiving. They'll be printed after `NEC Infrared code received:`. Then, you can change `code.py` to reflect the specific button codes for your remote.

IR from CPX to CPX

Your Circuit Playground Express can both transmit and receive IR signals! There is an example where each button on the CPX sends a different signal. We've emulated the volume up and volume down buttons on the Adafruit NEC remote since the receive code is already looking for those signals. So, the program will be sending four bytes of data with each button press for the other CPX to receive and decode.

Why send four bytes instead of one? IR communication is messy, and often gets mixed up with flickering lights in the room, or maybe something blocking the photons. If you send only one byte, you run the risk of a signal being mis-interpreted as some other signal. By requiring and checking for four digits, your chance of getting a mistaken message is less likely. Why not more than four? If the message is too long, it would take a long time to send, and use more power. Four bytes are defined for the NEC standard - it's a nice trade-off between too-short and too-long.

You'll need two Circuit Playground Expresses for this example.

You should still have the code from the previous example () on the first CPX.

Copy the following code into your `code.py` on the second CPX:

```
# SPDX-FileCopyrightText: 2018 Kattni Rembor for Adafruit Industries
#
# SPDX-License-Identifier: MIT

import time
from adafruit_circuitplayground.express import cpx
import adafruit_irremote
import pulseio
import board

# Create a 'pulseio' output, to send infrared signals on the IR transmitter @ 38KHz
pulseout = pulseio.PulseOut(board.IR_TX, frequency=38000, duty_cycle=2 ** 15)
# Create an encoder that will take numbers and turn them into NEC IR pulses
encoder = adafruit_irremote.GenericTransmit(header=[9500, 4500], one=[550, 550],
                                            zero=[550, 1700], trail=0)

while True:
    if cpx.button_a:
        print("Button A pressed! \n")
        cpx.red_led = True
        encoder.transmit(pulseout, [255, 2, 255, 0])
        cpx.red_led = False
        # wait so the receiver can get the full message
        time.sleep(0.2)
    if cpx.button_b:
        print("Button B pressed! \n")
        cpx.red_led = True
        encoder.transmit(pulseout, [255, 2, 191, 64])
        cpx.red_led = False
        time.sleep(0.2)
```

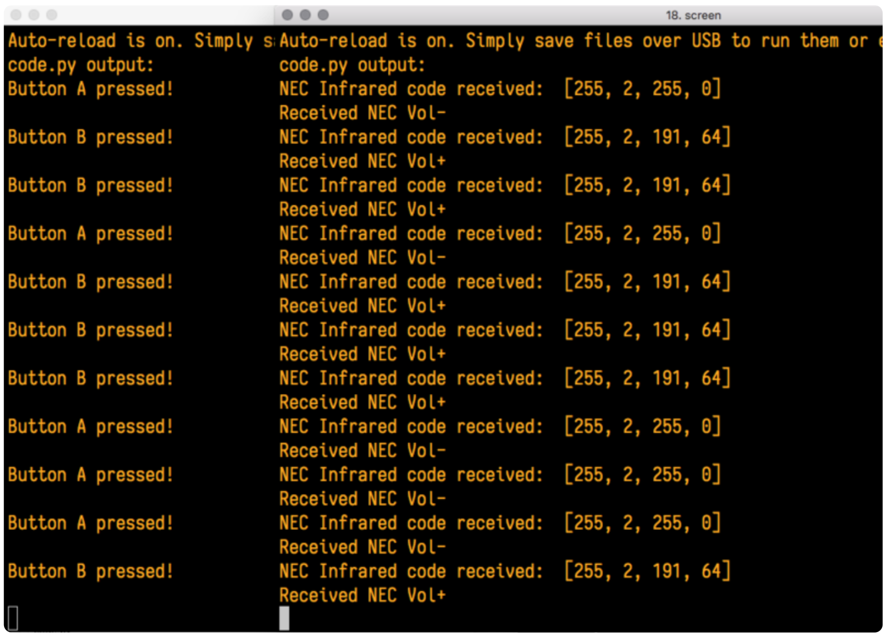
We create a `pulseio` output to send infrared signals from the IR transmitter at 38KHz. Then we create an `encoder` that will take the numbers we're sending and turn them into NEC IR pulses.

The arguments we pass into the `adafruit_irremote.GenericTransmit` line is what lets the irremote library know how to encode the 4 bytes into NEC remote data. There's a 9.5ms pulse high followed by a 4.5ms pulse low to let the receiver know data is coming. Then 550us+550us pulse pairs for a '1' value and 550us+1700us pulse pair for a '0' value.

header=[9500, 4500], one=[550, 550], zero=[550, 1700], trail=0

Inside our loop, we check to see if each button is pressed. When a button is pressed, we `print` a message to the serial console. Then, we turn on the red LED, send our 4 byte data signal, and turn the red LED off. Then we wait 0.2 seconds to give the receiver a chance to receive the full message.

Connect both Circuit Playground Expresses to the serial console to see the associated serial output. We've overlapped the windows side-by-side to show the serial output from both boards next to each other. You can see exactly what happens on the left when you press a button on the transmitting board, and on the right after the signal is received by the receiving board.



```
Auto-reload is on. Simply sAuto-reload is on. Simply save files over USB to run them or e
code.py output:            code.py output:
Button A pressed!          NEC Infrared code received:  [255, 2, 255, 0]
                           Received NEC Vol-
Button B pressed!          NEC Infrared code received:  [255, 2, 191, 64]
                           Received NEC Vol+
Button B pressed!          NEC Infrared code received:  [255, 2, 191, 64]
                           Received NEC Vol+
Button A pressed!          NEC Infrared code received:  [255, 2, 255, 0]
                           Received NEC Vol-
Button B pressed!          NEC Infrared code received:  [255, 2, 191, 64]
                           Received NEC Vol+
Button B pressed!          NEC Infrared code received:  [255, 2, 191, 64]
                           Received NEC Vol+
Button B pressed!          NEC Infrared code received:  [255, 2, 191, 64]
                           Received NEC Vol+
Button A pressed!          NEC Infrared code received:  [255, 2, 255, 0]
                           Received NEC Vol-
Button A pressed!          NEC Infrared code received:  [255, 2, 255, 0]
                           Received NEC Vol-
Button A pressed!          NEC Infrared code received:  [255, 2, 255, 0]
                           Received NEC Vol-
Button B pressed!          NEC Infrared code received:  [255, 2, 191, 64]
                           Received NEC Vol+
```

If you'd like to emulate your own remote, simply change the IR codes in the `code.py` above to match the changes you made in the last example. It works as long as you're transmitting the same code that the receiving board is expecting!

Using IR as an Input

We've shown how to send and receive IR signals from Circuit Playground Express to Circuit Playground Express. You can use those signals as an input to trigger events on the receiving CPX. So, let's do something fun with it. It's time to light it up and make some noise!

You'll need two Circuit Playground Expresses for this example.

Copy the following code to code.py on the CPX currently receiving signals. The transmission `code.py` will remain the same from the previous page

```
# SPDX-FileCopyrightText: 2018 Kattni Rembor for Adafruit Industries
#
# SPDX-License-Identifier: MIT

import pulseio
import board
import adafruit_irremote
from adafruit_circuitplayground.express import cpx

# Create a 'pulseio' input, to listen to infrared signals on the IR receiver
pulsein = pulseio.PulseIn(board.IR_RX, maxlen=120, idle_state=True)
# Create a decoder that will take pulses and turn them into numbers
decoder = adafruit_irremote.GenericDecode()

while True:
    pulses = decoder.read_pulses(pulsein)
    try:
        # Attempt to convert received pulses into numbers
        received_code = decoder.decode_bits(pulses)
    except adafruit_irremote.IRNECRepeatException:
        # We got an unusual short code, probably a 'repeat' signal
        # print("NEC repeat!")
        continue
    except adafruit_irremote.IRDecodeException as e:
        # Something got distorted or maybe its not an NEC-type remote?
        # print("Failed to decode: ", e.args)
        continue

    print("NEC Infrared code received: ", received_code)
    if received_code == [255, 2, 255, 0]:
        print("Button A signal")
        cpx.pixels.fill((130, 0, 100))
    if received_code == [255, 2, 191, 64]:
        print("Button B Signal")
        cpx.pixels.fill((0, 0, 0))
        cpx.play_tone(262, 1)
```

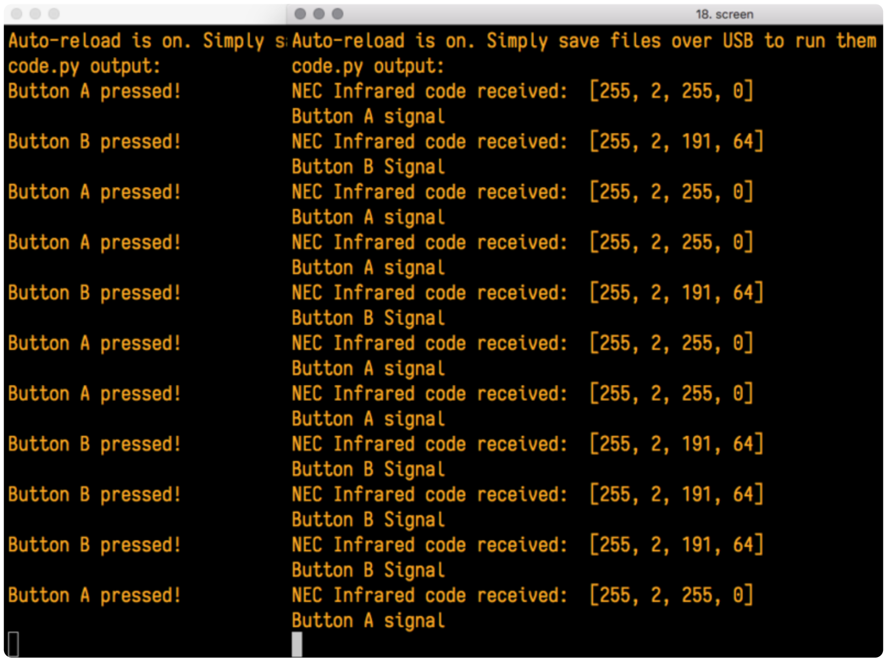
The beginning of this code is the same as the test with the remote. We create the `pulsesio` and `decoder` objects, wait to receive the signals, and attempt to decode them. Then we print the IR code received.

We check for the button presses like before as well. However, instead of simply `print()ing` to the serial output, we've added something more fun!

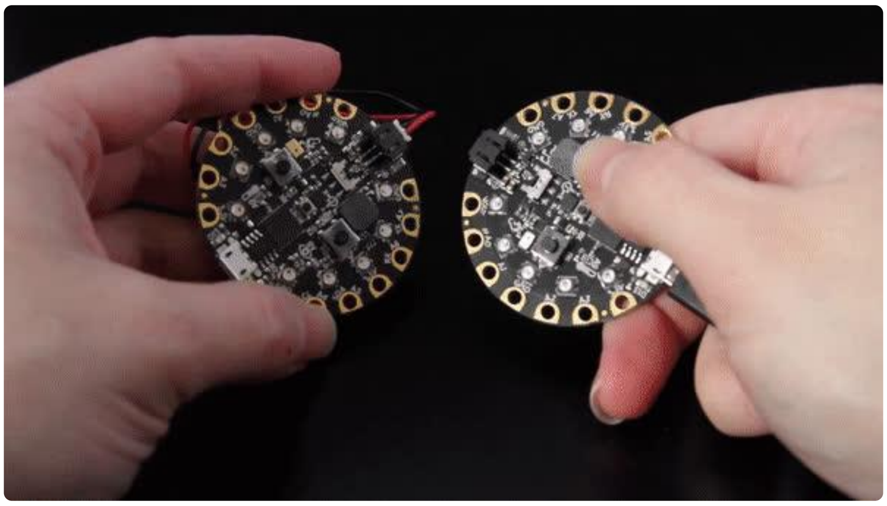
- If we receive the IR code now associated with button A, we `print` `Button A signal`, and turn all the NeoPixel LEDs purple!

- If we receive the IR code now associated with button B, we `print` `Button B signal`, turn all of the LEDs off, and play a 262Hz tone!

We've put the two windows together again to show the associated `print` statements from each board. The transmitting board output is on the left and the receiving output is on the right.



```
Auto-reload is on. Simply s Auto-reload is on. Simply save files over USB to run them
code.py output:             code.py output:
Button A pressed!           NEC Infrared code received:  [255, 2, 255, 0]
                            Button A signal
Button B pressed!           NEC Infrared code received:  [255, 2, 191, 64]
                            Button B Signal
Button A pressed!           NEC Infrared code received:  [255, 2, 255, 0]
                            Button A signal
Button A pressed!           NEC Infrared code received:  [255, 2, 255, 0]
                            Button A signal
Button B pressed!           NEC Infrared code received:  [255, 2, 191, 64]
                            Button B Signal
Button A pressed!           NEC Infrared code received:  [255, 2, 255, 0]
                            Button A signal
Button A pressed!           NEC Infrared code received:  [255, 2, 255, 0]
                            Button A signal
Button B pressed!           NEC Infrared code received:  [255, 2, 191, 64]
                            Button B Signal
Button B pressed!           NEC Infrared code received:  [255, 2, 191, 64]
                            Button B Signal
Button B pressed!           NEC Infrared code received:  [255, 2, 191, 64]
                            Button B Signal
Button A pressed!           NEC Infrared code received:  [255, 2, 255, 0]
                            Button A signal
```



You can do all kinds of things by adding code to these `if` blocks, such as, move a servo, play a wave file, change LED animations, or print a special message to the serial output. The possibilities are endless. Pick something and give it a try!