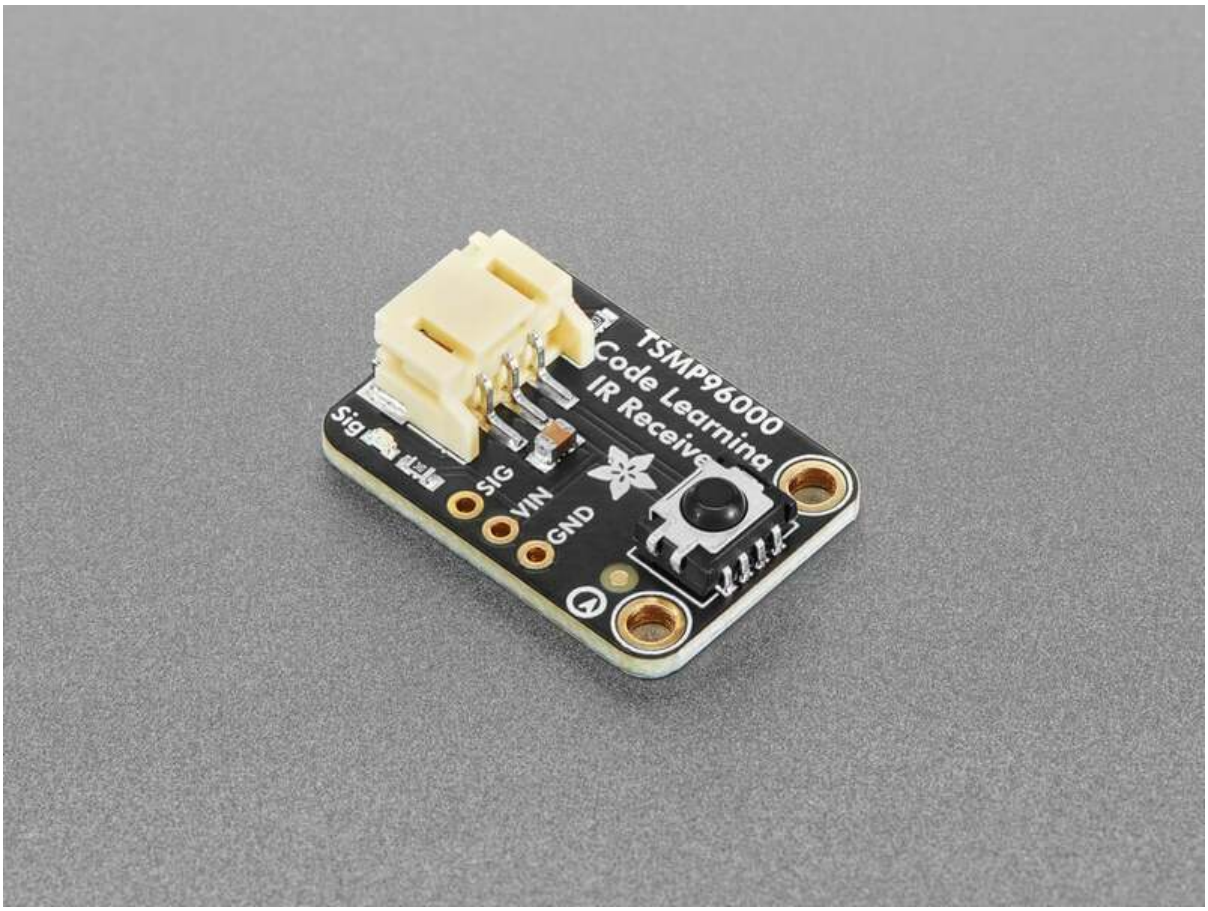
Adafruit TSMP96000 IR Receiver Breakout

Created by Liz Clark

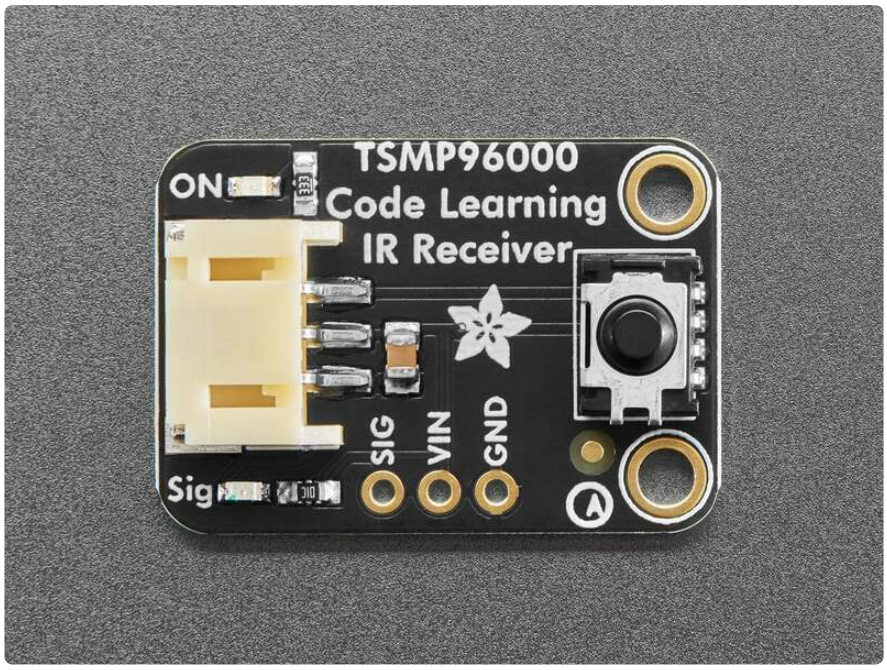


https://learn.adafruit.com/adafruit-tsmp96000-ir-receiver-breakout

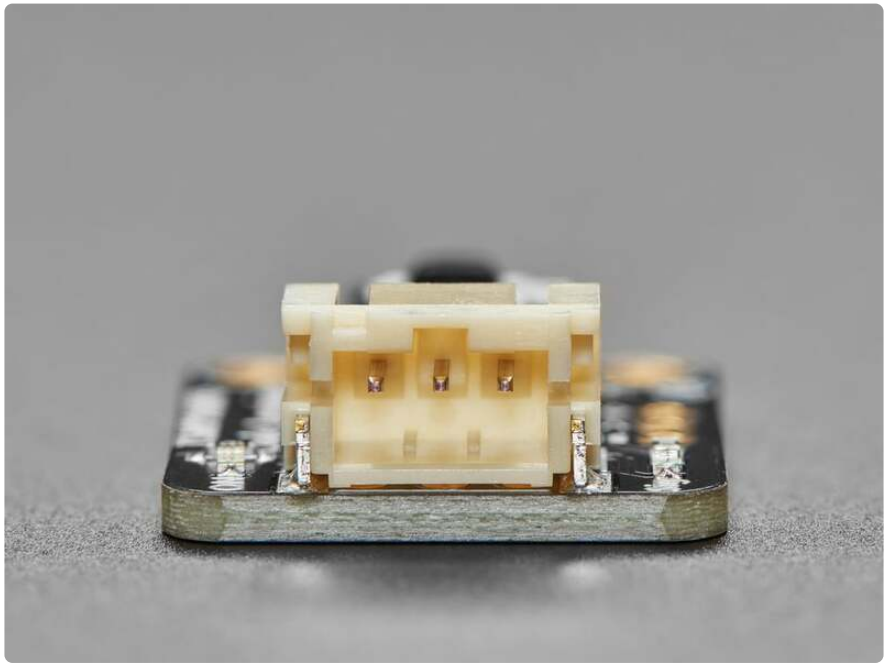
Last updated on 2024-06-05 10:09:53 AM EDT

Table of Contents

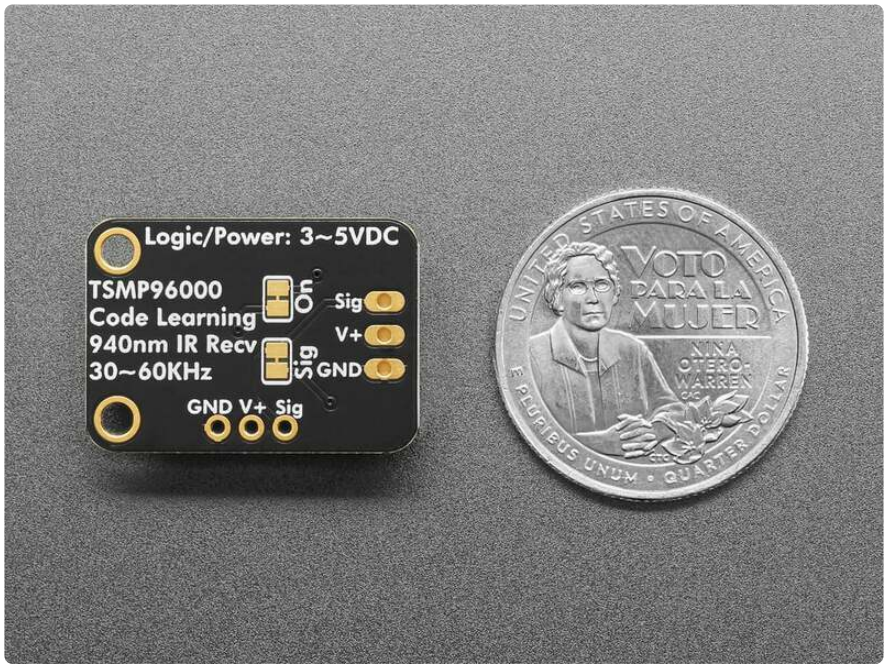
Overview



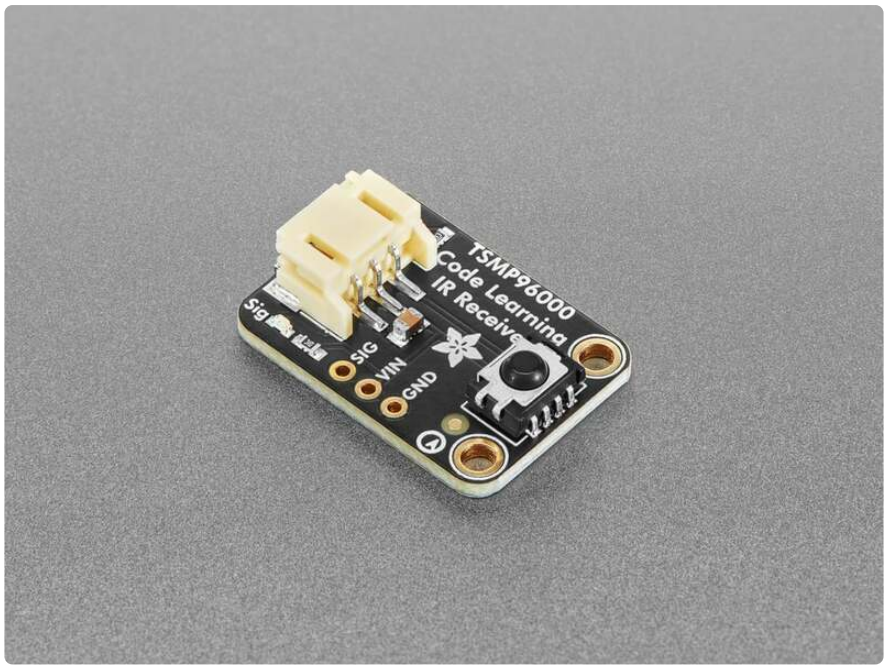
For classic 38KHz infrared remote signal reading, we've got a lovely STEMMA IR Receiver (http://adafru.it/5939). But if you want to read infrared signals from remotes with different carrier signals, especially when you don't know the frequency, this **Adafruit TSMP96000 "Code Learning" Infrared IR Receiver Breakout** has the ability to detect IR signals from 20 to 60KHz and provide the carrier signal for analysis. This is used for "code learning" situations where you want your device to work with any IR remote control.



Usage is simple: Power the board by connecting V+ and ground to 3~5VDC, point a 20~60KHz IR remote control at the sensors, and press some buttons. The modulated IR signal, with the carrier signal intact, is piped out the Signal pin into your microcontroller which will then need to decode it. To make usage really easy, we have a green 'power good' LED and a red 'signal' LED. When IR remote signals are read by the onboard sensors, the red LED will blink to let you know.

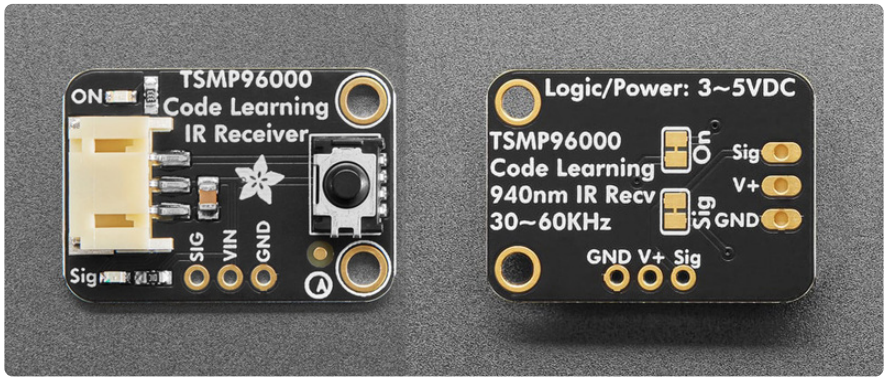


This board will work nicely for advanced IR remote receiving projects, because you don't get the demodulated output it's not good for most IR decoding firmware - make sure you've got code specifically designed for "code learning"! With mounting holes and a cable, it's easy to mount in enclosures and on devices. Using a 2mm pitch STEMMA JST PH cable with headers or alligator clips on the end, you can easily wire this board without any soldering.



Each STEMMA board is a fully assembled and tested PCB, but no cable. No soldering is required to use it, but you will need to pick up a 2mm pitch, 3-pin STEMMA JST PH cable (https://adafru.it/18cS). Alternatively, if you do want to solder, there's a 0.1" spaced header for power/ground/signal.

Pinouts



Power Pins

- **VIN/V+** - this is the power pin. To power the board, give it the same power as the logic level of your microcontroller - e.g. for a 5V microcontroller like Arduino, use 5V.
- **GND** - common ground for power and logic.

Signal Output

- **SIG/Sig** - this is the output signal from the infrared receiver. When an IR signal is received, the signal is piped out the Signal pin into your microcontroller. The TSMP96000 can detect IR signals from 20 to 60KHz and provide the carrier signal for analysis.

STEMMA JST PH

- **STEMMA JST PH** (https://adafru.it/Ft4) - 2mm pitch STEMMA JST port for use with 3-pin STEMMA JST PH cables (https://adafru.it/JRA). It has connections for:
  - **GND** - common ground for power and data. It is the black wire on the JST PH cable.
  - **V+** - power input for the infrared receiver. It is the red wire on the JST PH cable.
  - **Sig** - signal to your microcontroller. It is the white wire on the JST PH cable.

Signal LED and Jumper

- **Signal LED** - to the left of the JST PH connector is the signal LED, labeled **Sig**. It is the red LED. It will light up when an IR signal is read by the TSMP96000.
- **LED jumper** - in the center of the back of the board is a jumper for the signal LED. It is labeled **Sig** on the board silk. If you want to disable the signal LED, cut the trace on this jumper.

Power LED and Jumper

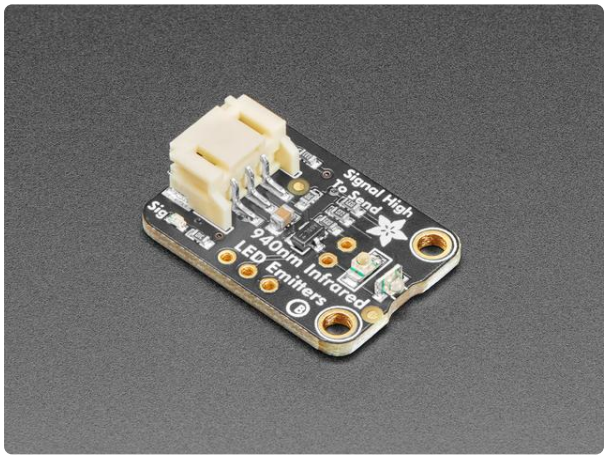
- **Power LED** - to the right of the JST PH connector is the power LED, labeled **ON**. It is the green LED.
- **LED jumper** - in the center of the back of the board is a jumper for the power LED. It is labeled **On** on the board silk. If you want to disable the power LED, cut the trace on this jumper.

Arduino

Using the TSMP96000 IR Receiver Breakout with Arduino involves wiring up the breakout to your Arduino-compatible microcontroller and running the provided

example code. The code is compatible with AVR (ATmega328, ATmega32u4, etc) and RP2040 boards.

You'll need an infrared emitter to use this example with the breakout, such as an IR remote or IR LED:



Adafruit High Power Infrared IR LED Emitter - STEMMA JST PH 2mm
pew *pew*! This board is like a little ray gun for infrared light, with two high powered LED outputs. When controlled with the onboard N-Channel FET driver,...
https://www.adafruit.com/product/5639

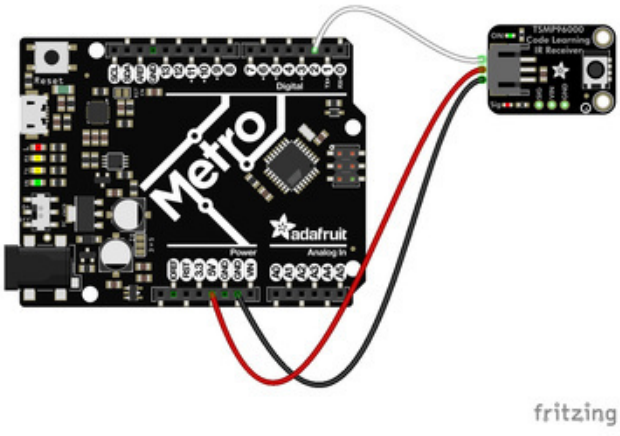
Mini Remote Control
This little remote control would be handy for controlling a robot or other project from across the room. It has 21 buttons and a layout we thought was handy: directional buttons and...
https://www.adafruit.com/product/389

This breakout is for advanced IR remote receiving projects. You don't get the demodulated output, so it is not good for most IR decoding firmware.

Wiring

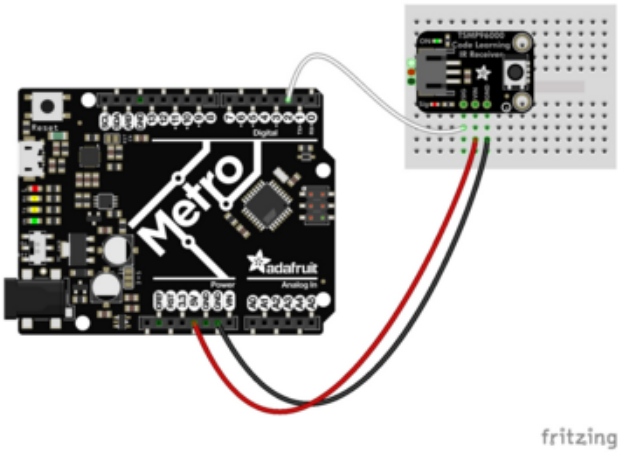
Wire as shown for a **5V** board like an Uno. If you are using a **3V** board, like an Adafruit Feather, wire the board's 3V pin to the breakout VIN.

Here is an Adafruit Metro wired up to the demodulator using a JST PH cable.



Board 5V to **breakout JST PH V+ (red wire)**
Board GND to **breakout JST PH GND (black wire)**
Board pin 2 to **breakout JST PH Sig (white wire)**

Here is an Adafruit Metro wired up using a solderless breadboard:



Board 5V to **breakout VIN (red wire)**
Board GND to **breakout GND (black wire)**
Board pin 2 to **breakout SIG (white wire)**

No additional libraries are needed for this example code.

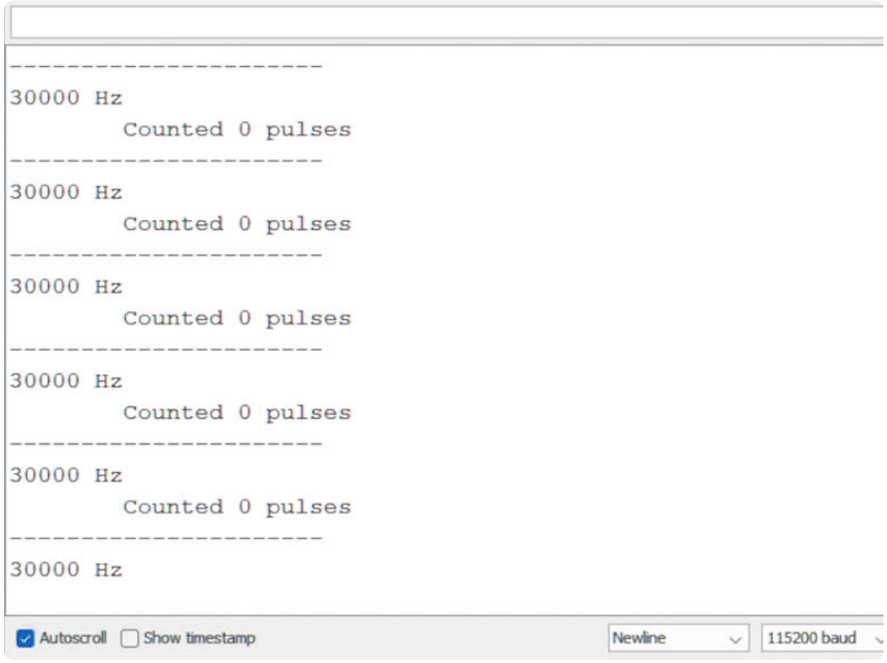
Example Code

```
// SPDX-FileCopyrightText: 2024 ladyada for Adafruit Industries
//
// SPDX-License-Identifier: MIT

#include <Arduino.h>

#define IROUT 9
#define IRIN  2

volatile unsigned long pulseCount = 0;

#if defined(ARDUINO_RASPBERRY_PI_PICO)
#include "hardware/pwm.h"
void countPulse() {
  pulseCount++;
}
```

```
#else
void countPulse() {
  pulseCount++;
}
#endif

void setup() {
  Serial.begin(115200);
  pinMode(IRIN, INPUT); // Assuming the input signal is connected to pin 2

  attachInterrupt(digitalPinToInterrupt(IRIN), countPulse, RISING);
  pinMode(IROUT, OUTPUT);

}

bool testFreq(uint32_t freq) {
  uint32_t temp = 0;

  Serial.print(freq); Serial.println(" Hz");
  setFrequency(freq); // Set the initial frequency to the specified value
  pulseCount = 0;
  delay(100);
  temp = pulseCount;
  Serial.print("\tCounted "); Serial.print(temp);
  Serial.println(" pulses");
  if ((temp > (freq / 10) + (freq / 100)) || (temp < (freq / 10) - (freq / 100))) {
    return false;
  }
  return true;
}

void loop() {
  Serial.println("----------------------");
  if (!testFreq(30000)) return;
  if (!testFreq(40000)) return;
  if (!testFreq(50000)) return;
  if (!testFreq(60000)) return;

}

void setFrequency(unsigned long frequency) {
#if defined(__AVR__)
  unsigned long ocrValue;
  byte csBits = 0;

  // Disable interrupts
  noInterrupts();

  // Reset Timer1 Control Registers
  TCCR1A = 0;
  TCCR1B = 0;
  TCNT1 = 0; // Reset counter

  // Set Timer1 to CTC mode (Clear Timer on Compare Match)
  TCCR1B |= (1 << WGM12);

  // Determine best prescaler and OCR1A value for the desired frequency
  if (frequency > 4000) { // Can use no prescaler if frequency is high enough
    csBits = (1 << CS10); // No prescaler
    ocrValue = 16000000 / (2 * frequency) - 1;
  } else if (frequency > 500) {
    csBits = (1 << CS11); // Prescaler 8
    ocrValue = 2000000 / (2 * frequency) - 1;
  } else if (frequency > 60) {
    csBits = (1 << CS11) | (1 << CS10); // Prescaler 64
    ocrValue = 250000 / (2 * frequency) - 1;
  } else {
    csBits = (1 << CS12); // Prescaler 256
    ocrValue = 62500 / (2 * frequency) - 1;
```

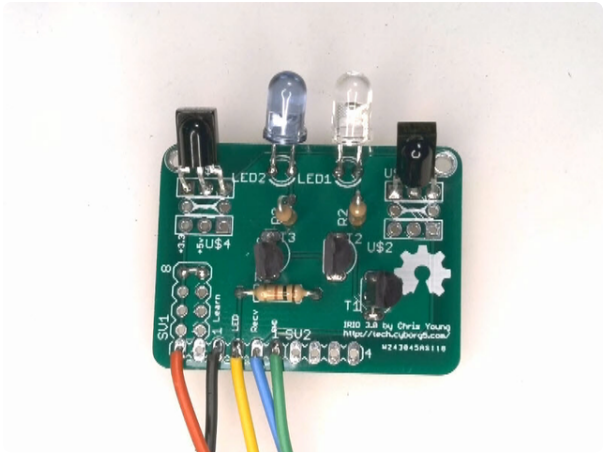
```
  }

  // Handle boundary conditions for OCR1A
  if (ocrValue > 65535) ocrValue = 65535; // Cap at maximum for 16-bit timer
  if (ocrValue < 1) ocrValue = 1; // Ensure OCR1A is at least 1

  OCR1A = ocrValue;
  TCCR1B |= csBits; // Set the prescaler
  TCCR1A |= (1 << COM1A0); // Toggle OC1A on Compare Match

  // Re-enable interrupts
  interrupts();

#elif defined(ARDUINO_RASPBERRY_PI_PICO)
  // Set PWM frequency for the RP2040
  gpio_set_function(IROUT, GPIO_FUNC_PWM);
  uint slice_num = pwm_gpio_to_slice_num(IROUT);
  pwm_set_wrap(slice_num, 125000000 / frequency);
  pwm_set_chan_level(slice_num, PWM_CHAN_A, 125000000 / (2 * frequency));
  pwm_set_enabled(slice_num, true);

#endif
}
```

Upload the sketch to your board and open up the Serial Monitor (**Tools -> Serial Monitor**) at 115200 baud. As you send IR signals to the breakout, you'll see the frequency and pulse count print to the Serial Monitor.



Going Further

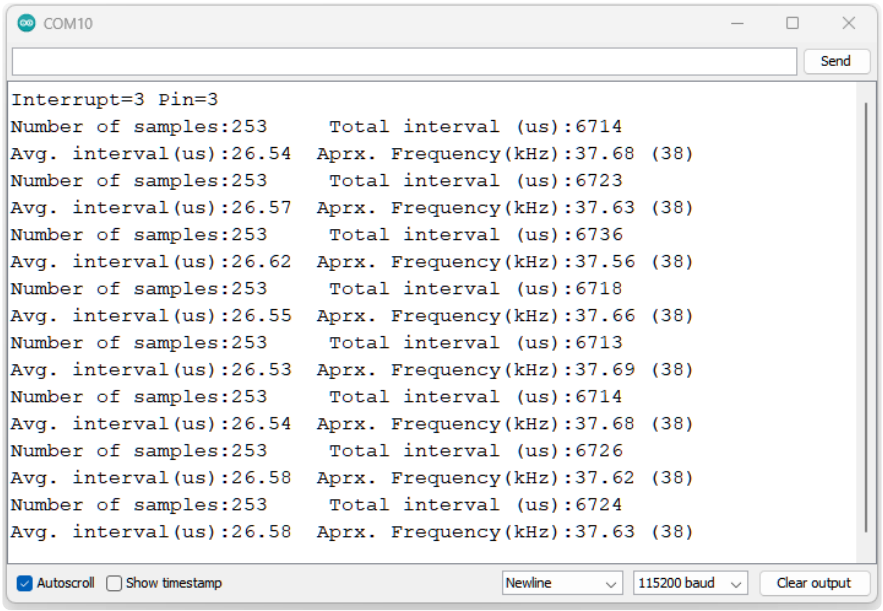
Chris Young (https://adafru.it/1a1U) has an excellent guide detailing how he built an IR Transmitter and Receiver that utilizes a similar code learning module (the TSMP58000 (https://adafru.it/1a1V)). This guide demonstrates how you would utilize one of these receivers to decode IR messages.



Building an Infrared Transmitter and Receiver Board
By Chris Young
Overview

https://learn.adafruit.com/building-an-infrared-transmitter-and-receiver-board/overview

Also noted in the guide is Chris' IRLib2 Arduino library (https://adafru.it/vwF). The frequency example (https://adafru.it/1a1W) measures frequency modulation from an IR signal and prints it to the Serial Monitor.



Downloads

Files

- TSMP96000 Datasheet (https://adafru.it/1a1X)
- EagleCAD PCB Files on GitHub (https://adafru.it/1a1Y)
- Fritzing object in the Adafruit Fritzing Library (https://adafru.it/1a1Z)

Schematic and Fab Print