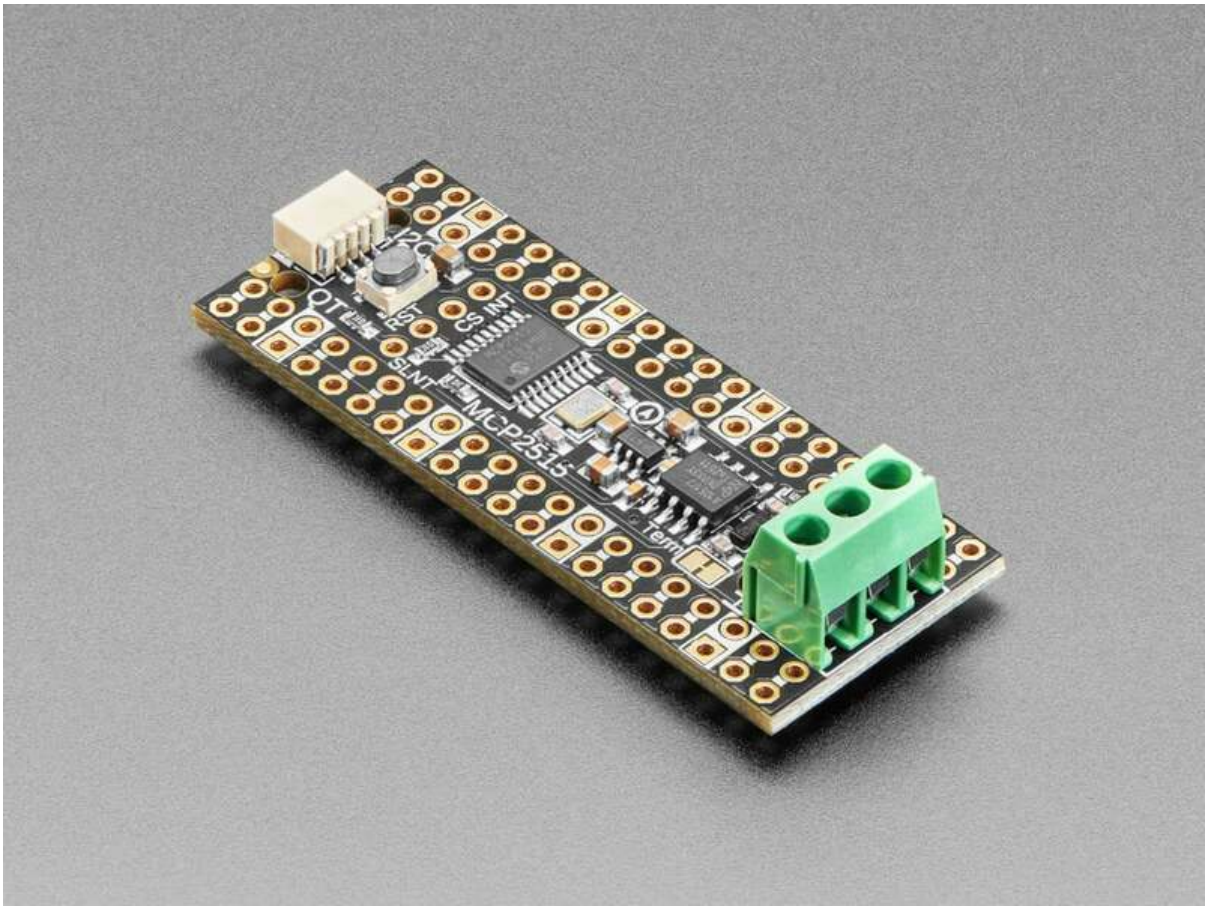
Adafruit PiCowbell CAN Bus for Pico

Created by Liz Clark

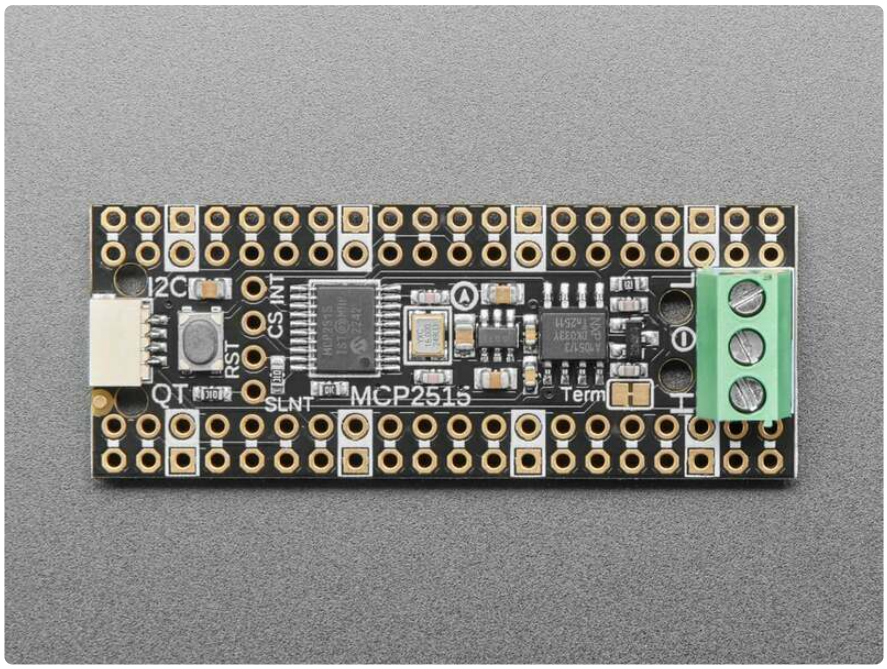


https://learn.adafruit.com/adafruit-picowbell-can-bus-for-pico

Last updated on 2024-06-03 03:50:26 PM EDT

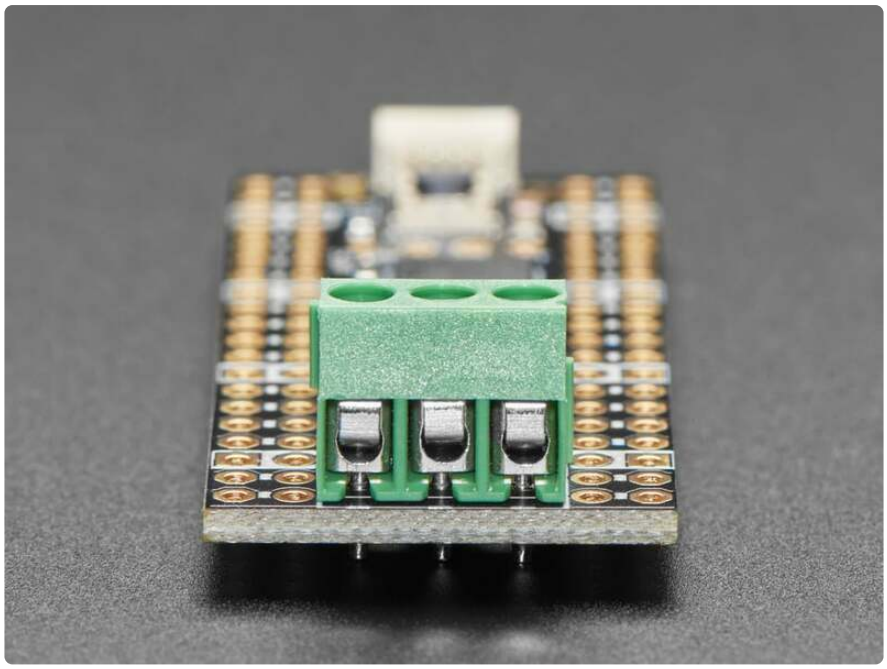
Table of Contents

Overview

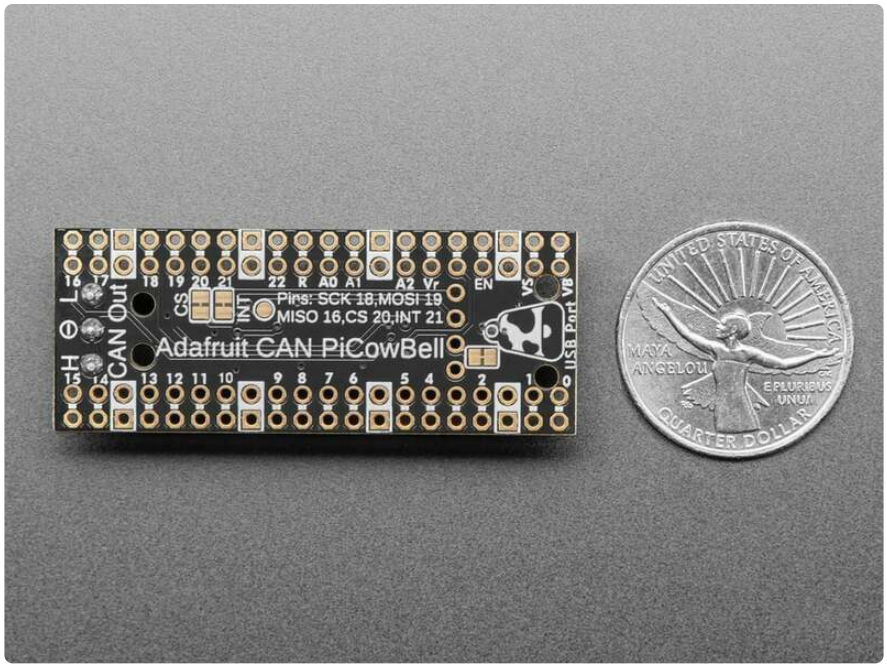


Ding dong! Hear that? It's the PiCowbell ringing, letting you know that the new **Adafruit PiCowbell CAN Bus** board is in stock and ready to assist your Raspberry Pi Pico (http://adafru.it/4864) and Pico W (http://adafru.it/5526) project in connecting to CAN bus networks for automotive or robotics projects.

CAN Bus is a small-scale networking standard (https://adafru.it/18Bb), originally designed for cars and, yes, busses, but is now used for many robotics or sensor networks that need better range and addressing than I2C and don't have the pins or computational ability to talk on Ethernet. CAN is a 2 wire differential standard, which means it's good for long distances and noisy environments.

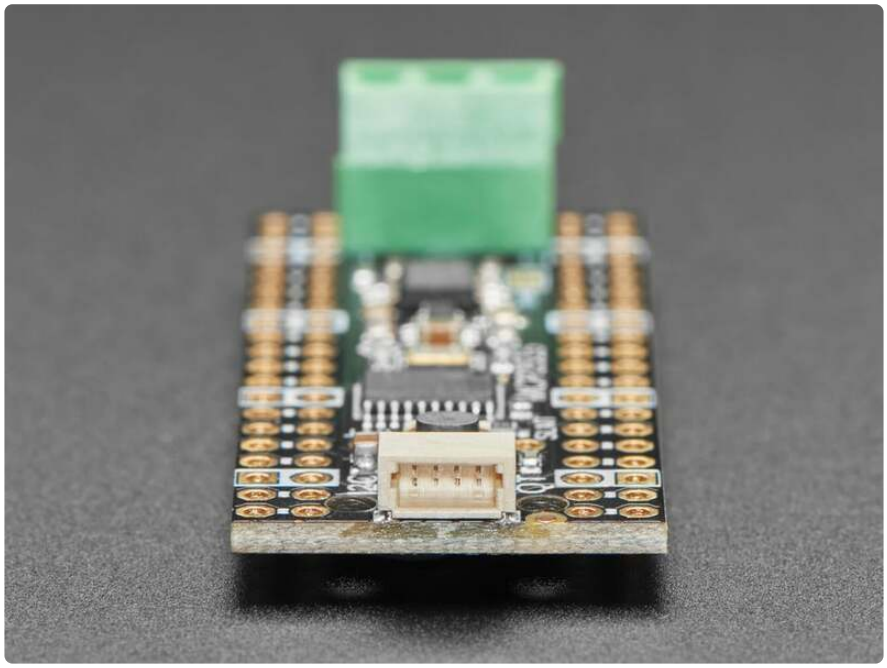


Messages are sent at about a 1 Mbps rate - you set the frequency for the bus and then all 'joiners' must match it, and have an address before the packet so that each node can listen in to messages just for it. New nodes can be attached easily because they just need to connect to the two data lines anywhere in the shared net. Each CAN device sends messages whenever it wants, and thanks to some clever data encoding, can detect if there's a message collision and retransmit later.



If you'd like to connect your Raspberry Pi Pico to a CAN Bus, the **Adafruit PiCowbell CAN Bus board** has a MCP2515 controller and TJA1051/3 transceiver! The controller used is the MCP2515, an extremely popular and well-supported chipset (https://adafru.it/18Be) that has drivers in Arduino and CircuitPython (https://adafru.it/18Bm)

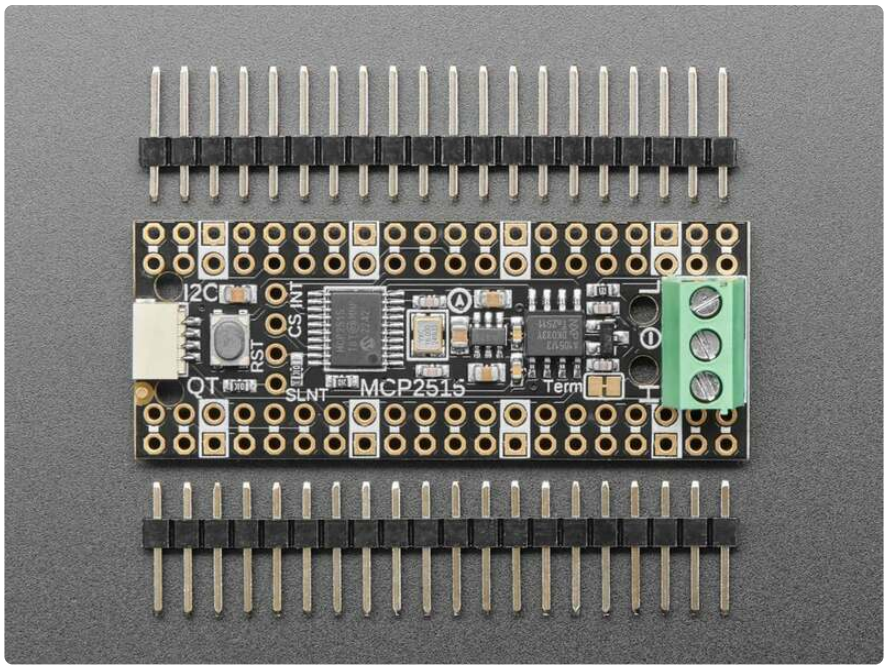
and only requires an SPI port and two pins for chip-select and IRQ. Use it to send and receive messages in either standard or extended format at up to 1 Mbps.



Adafruit has added a few nice extras to this PiCowBell to make it useful in many common CAN scenarios:

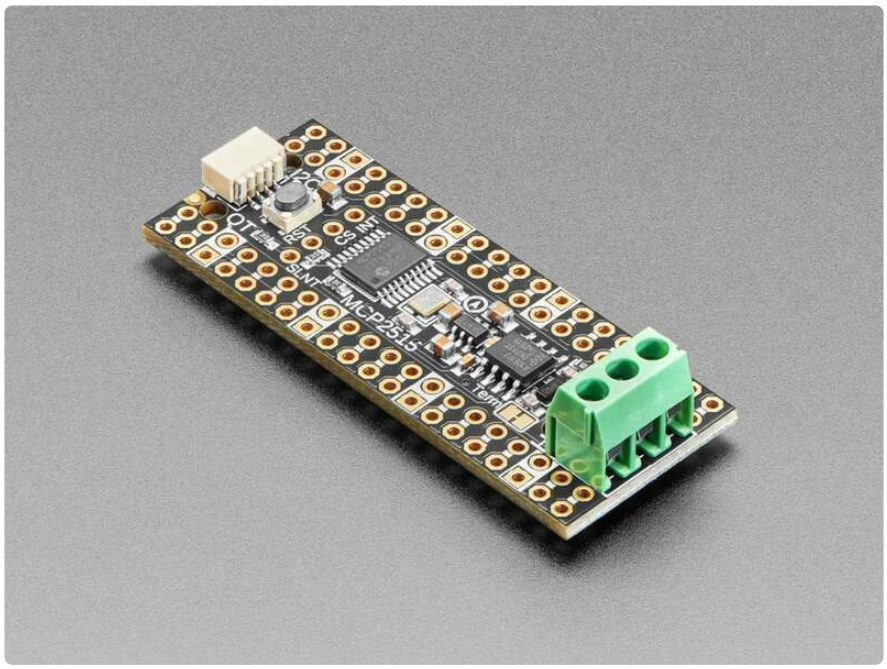
- **5V charge-pump voltage generator** (http://adafru.it/3661), so even though you are running 3.3V on a Pico board, it will generate a nice clean 5V as required by the transceiver.
- 3.5mm terminal block **(http://adafru.it/725)** pre-soldered on the board to get quick access to the High and Low data lines as well as a ground pin.
- **120-ohm termination resistor on board**, you can remove the termination easily by cutting the jumper marked **Term** on the top of the board.
- **Pre-connected CS and INT pins** on Pico GPIO #20 and #21. You can cut the bottom solder jumpers and use the breakout pads to connect to any two IO pins you like.

Each order comes with an assembled PCB and header. You will need to solder in the header yourself, but it's a quick task.



Please Note! There are a lot of possible configurations, and we stock various headers depending on how you want to solder and attach. Especially if you want the Pico on top so that the **BOOTSEL** button and LED are accessible.

1. Use the Pico Stacking Headers (http://adafru.it/5582) if you want to be able to plug into a breadboard or other accessory with sockets.
2. Use the Pico Socket Headers (http://adafru.it/5583) if you want to plug directly in and have a nice solid connection that doesn't have any poking-out-bits.

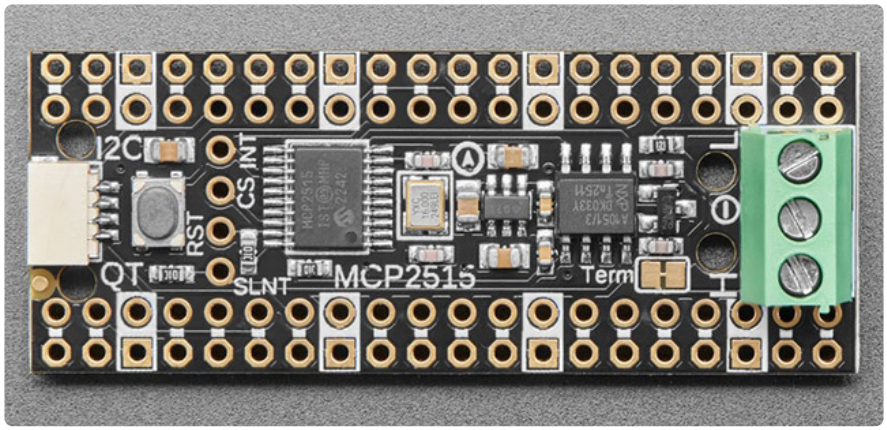


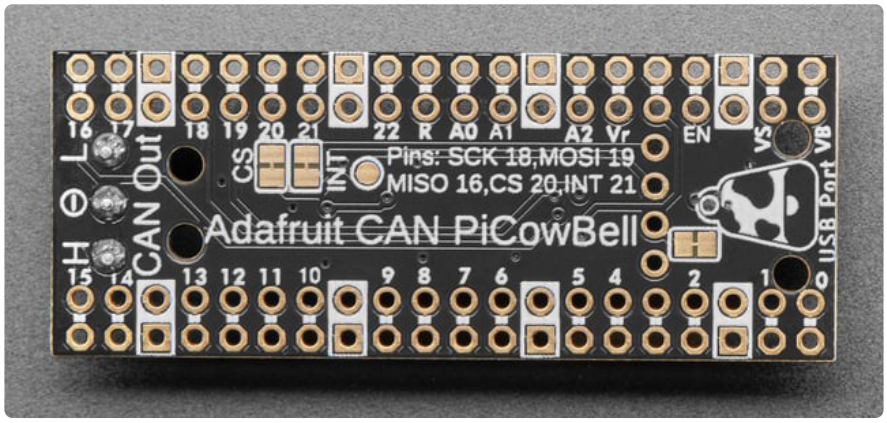
The PiCowbell CAN Bus provides you with:

- **Right angle JST SH connector for I2C / Stemma QT / Qwiic** connection. Provides 3V, GND, IO4 (SDA), and IO5 (SCL).
- **Reset button** - Press to restart your program.
- **Every pad on the 'bell has a duplicate hole pad** next to them for solder-jumpering
- **The ground pads have white silkscreen rectangles** for easy identification, plus one long ground strip near the reset button.
- **Gold-plated pads for easy soldering.**

If you are using the Philhower Arduino core, the Wire peripheral is already set up to use **GPIO4** and **GPIO5**, and SPI is default on **GPIO16**, **GPIO18** and **GPIO19**. If you use CircuitPython or MicroPython, you'll need to let the code know to look at **4**+**5** for SDA+SCL pins, and configure the SPI port for SCK=**18**, MOSI=**19** and MISO=**16**.

Pinouts





Power

- **VB (VBUS)** - This is the micro-USB input voltage, connected to the micro-USB port on the Raspberry Pi Pico. It is nominally 5V.
- **VS (VSYS)** - This is the main system input voltage. It can range from 1.8V to 5.5V and is used to generate the 3.3V needed for the RP2040 and the GPIO pins.
- **EN (3V3_EN)** - This connects to the enable pin on the Raspberry Pi Pico, and is pulled high (to VSYS) via a 100kΩ resistor.
- **3V** - This is the 3.3V output from the Raspberry Pi Pico.
- **VR (ADC_VREF)** - This is the ADC power supply and reference voltage. It is generated on the Raspberry Pi Pico by filtering the 3.3V supply. It can be used with an external reference when ADC performance is required.
- **G** - This is the common ground for power and logic. All **GND pins are highlighted in white on the silk**, with the exception of the ground pin for the CAN Bus terminal block. It is labeled **- (minus sign).**

I2C Logic

- **SCL** - I2C clock pin on the PiCowbell. It is connected to your microcontroller I2C clock line, which is **GPIO5** on the Pico. This connection is shared with the STEMMA QT port on the end of the board.
- **SDA** - I2C data pin on the PiCowbell. It is connected to your microcontroller I2C data line, which is **GPIO4** on the Pico. This connection is shared with the STEMMA QT port on the end of the board.
- STEMMA QT **(https://adafru.it/Ft4) -** These connectors allow you to connect to dev boards with STEMMA QT connectors or to other things with various associated accessories (https://adafru.it/JRA). The port is located on the end of the PiCowbell.

Duplicate GPIO Hole Pads

The following pads on the PiCowbell CAN Bus have a duplicate hole pad next to it for solder-jumpering:

- **GP0-GP15**, **GP16-GP22**, **Reset**, **A0-A2**, **VR**, **3V**, **EN**, **VS** and **VB**. Ground pins that have a duplicate hole pad are highlighted in white on the board silkscreen.
    - Note that **GP20** and **GP21** are connected by default to CAN Bus **CS** and **INT**. They can be disconnected from these GPIO pins by cutting the CS and INT jumpers described below.

Reset Button

In the center of the board, to the right of the STEMMA QT connector, is the reset button. It is routed to the **reset pin on the PiCowbell** and is labeled **RST** on the board silk. You can press it to restart your program.

Terminal Block Pins

On the front of the board is the terminal block with the three pins for communicating with the CAN Bus standard.

- **L** - the CAN low signal for the CAN Bus standard.
- **-** - common ground shared between the two CAN connections
- **H** - the CAN high signal for the CAN Bus standard.

Both **L** and **H** are connected to a **5V charge-pump voltage generator** (http://adafru.it/3661). Even if you are using 3.3V logic on a Pico board, it will generate a nice clean 5V as required by the CAN Bus transceiver.

Terminator Jumper

The terminator jumper is located on the front of the board, to the left of the terminal block. It is labeled **Term** and is outlined in white on the board silk.

- **Term** - The board has a 120 ohm termination resistor connected between **H** and **L**. You can disable (remove) the resistor by **cutting** the **Term** jumper, if your bus is already terminated.

MCP2515 Reset Pin and Jumper

- **Reset Jumper** - located on the back of the board, below the PiCowbell icon on the board silk, is the reset jumper. It is outlined in white on the board silk. You can cut this jumper to disconnect the **MCP2515 reset pin** from the **Pico reset pin**.
- **Reset Pin** - located on the front of the board to the right of the **reset button**. It is labeled **RST** on the silk. If you cut the **reset jumper**, you can use this pin to connect the **MCP2515 reset pin** to a different IO pin.

Chip Select Pin and Jumper

- **CS Jumper** - located on the back of the board, next to **PiCowbell pin 20**, is the chip select jumper. It is labeled **CS** and is outlined in white on the board silk. By default, the **MCP2515 chip select pin** is connected to **GPIO20** on the PiCowbell. You can cut this jumper to disconnect the **MCP2515 CS pin** from **PiCowbell pin 20**.
- **CS pin** - located on the front of the board to the right of the **reset button**. It is labeled **CS** on the silk. If you cut the **CS jumper**, you can use this pin to connect the **MCP2515 CS pin** to a different IO pin.

Interrupt Pin and Jumper

- **INT Jumper** - located on the back of the board, next to **PiCowbell pin 21**, is the interrupt jumper. It is labeled **INT** and is outlined in white on the board silk. By default, the **MCP2515 interrupt pin** is connected to **GPIO21** on the PiCowbell. You can cut this jumper to disconnect the **MCP2515 INT pin** from **PiCowbell pin 21**.

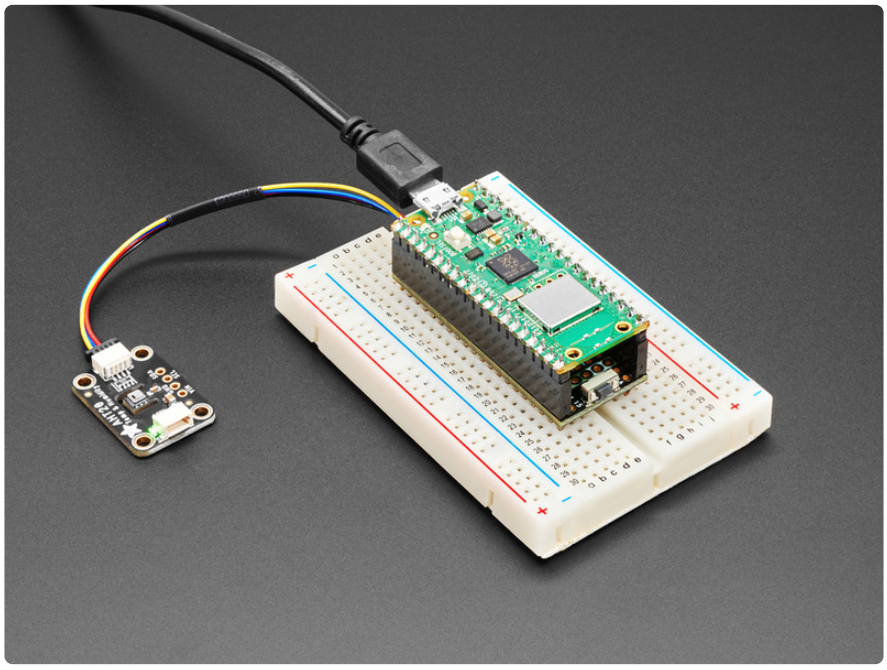
- **INT pin** - located on the front of the board to the right of the **reset button**. It is labeled **INT** on the silk. If you cut the **INT jumper**, you can use this pin to connect the **MCP2515 INT pin** to a different IO pin.

Silent Mode Pin

- **SLNT pin** - located on the front of the board to the right of the **reset button**. It is labeled **SLNT** on the silk. This pin can be pulled high to put the TJA1051/3 transceiver into silent mode. From the datasheet:

  In Silent mode, the transmitter is disabled, releasing the bus pins to recessive state. All other IC functions, including the receiver, continue to operate as in Normal mode. Silent mode can be used to prevent a faulty CAN controller from disrupting all network communications.

Assembly



Although these pages show the PiCowbell Proto, the soldering instructions are applicable for all PiCowbell boards.

There are four ways to get your PiCowbell board working with your Pico. To keep things flexible, PiCowbells do not come with headers: there's a lot of possible configurations and we stock various headers depending on how you want to solder and attach. Especially since you want the Pico on top, so that the BOOTSEL button and LED are accessible.

The options are as follows.

1. Use the Pico Stacking Headers (http://adafru.it/5582) if you want to be able to plug into a breadboard or other accessory with sockets.
2. Use the Pico Socket Headers (http://adafru.it/5583) if you want to plug directly into the Pico and have a nice solid connection that doesn't have any poking-out-bits.
3. For some PiCowbells: Use the Short Socket Headers (http://adafru.it/5585) for a very slim but pluggable design, note that you'll want to trim down the Pico's headers or use the short plug headers on the Pico (http://adafru.it/5584) to have a skinny sandwich.
4. For some PiCowbells: Solder the PiCowbell directly to the standard headers already soldered to your Pico. Of course this is very compact and inexpensive but you won't be able to remove the PiCowbell. However, this method is not possible for some PiCowbell variants depending on the clearance of the components on the PiCowbell (i.e. the PiCowbell Adalogger and its coin cell battery holder).

The next page shows how to solder standard headers onto a Pico board. The following four pages walk you through each type of PiCowbell assembly so you can choose the one that will work best for you!

You MUST solder all of the pins for the PiCowbell to work! Soldering only a few pins, or not soldering at all are not sufficient!

If you're unsure about soldering up the Pico and PiCowbell, check out our FAQ on soldering (https://adafru.it/18b9).

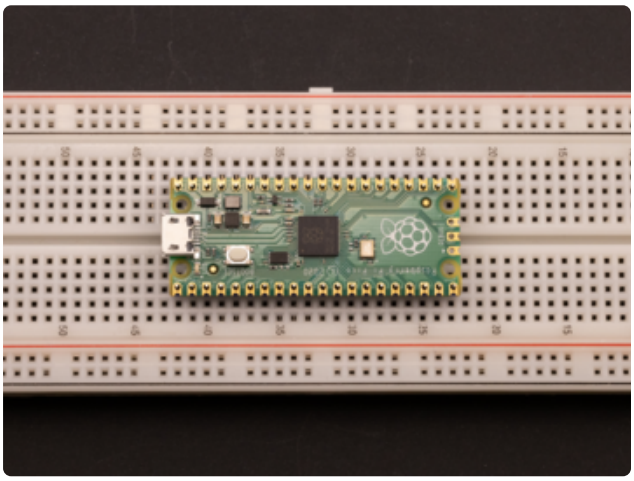
Pico

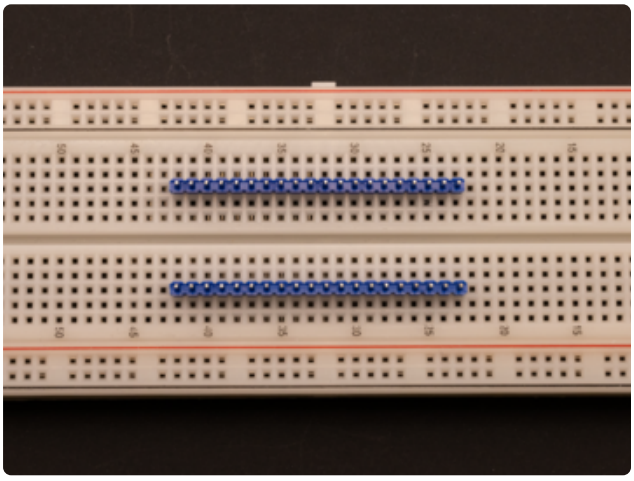
Three out of four of the assembly methods included in this guide assume you have a Raspberry Pi Pico soldered up with standard male headers in preparation for using it with the PiCowbell Proto. This page will show you how to solder a set of standard headers to a Pico.

(The shorty header assembly method uses short male headers on the Pico. The soldering concept is exactly the same, but use the shorty male headers on the Pico instead of standard ones. You can follow these instructions with the shorty headers and you'll be set for that.)

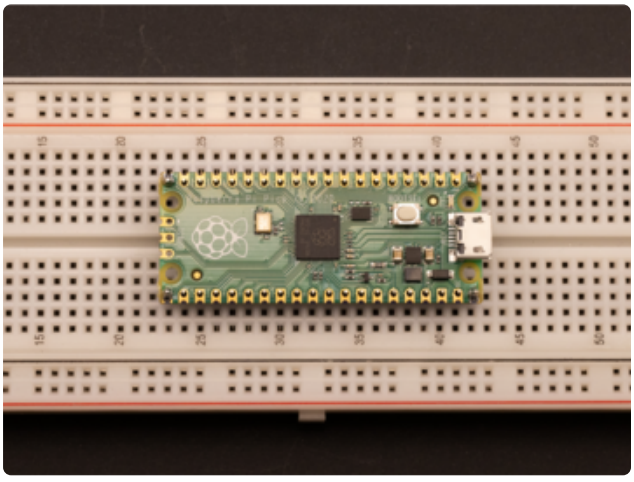
Follow the steps below to solder the standard male headers to a Pico. The process is the same for all flavors of Pico, such as Pico W.

Assembly Steps

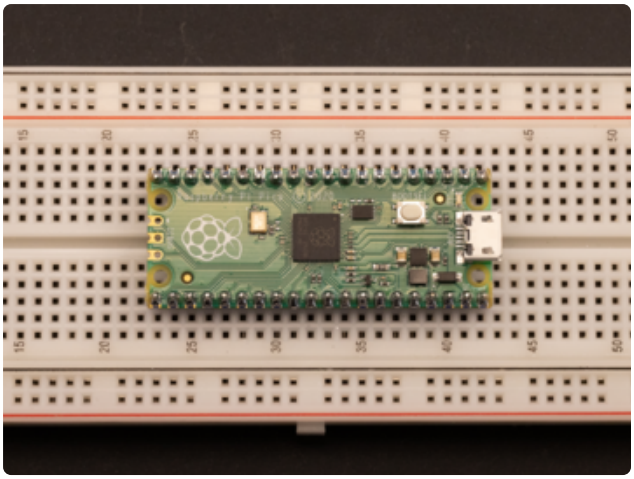




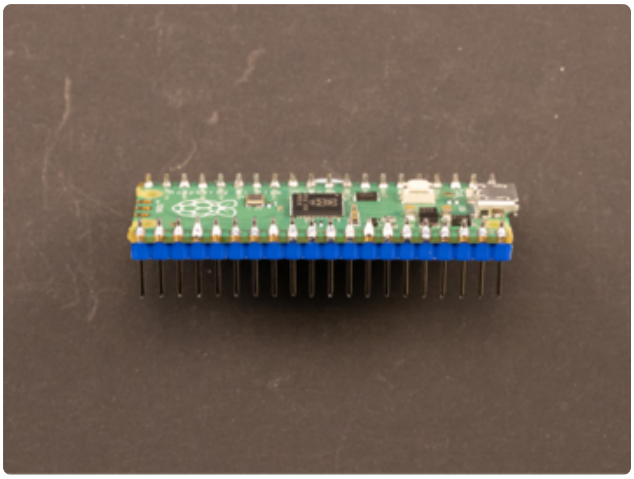
Use the Pico to line up the headers on a breadboard. This is the easiest way to ensure the headers are soldered on straight.



Solder the pins on each end of the two header strips, so the four corners of the Pico are soldered. This ensures the Pico and headers are attached properly while you continue to solder the rest of the pins.



Solder the rest of the pins.



Remove it from the breadboard. You're done!

For a bit more detail on the process of soldering standard male headers to a board, check out the How to Solder Headers' Male Headers page (https://adafru.it/188C).

Stacking Headers

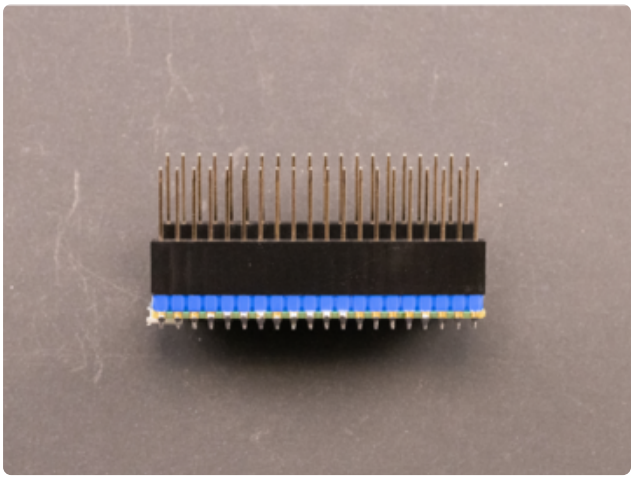
The first PiCowbell assembly method uses stacking headers, which allows you to use a breadboard with your PiCowbell-Pico sandwich. This is super helpful when you're still prototyping other parts of your project, or simply want jumper-wire access to the Pico pins in addition to the PiCowbell.

This page assumes you have already soldered standard male headers to your Pico. If you have not, please return to the Pico assembly page (https://adafru.it/18bQ) and follow the steps there.

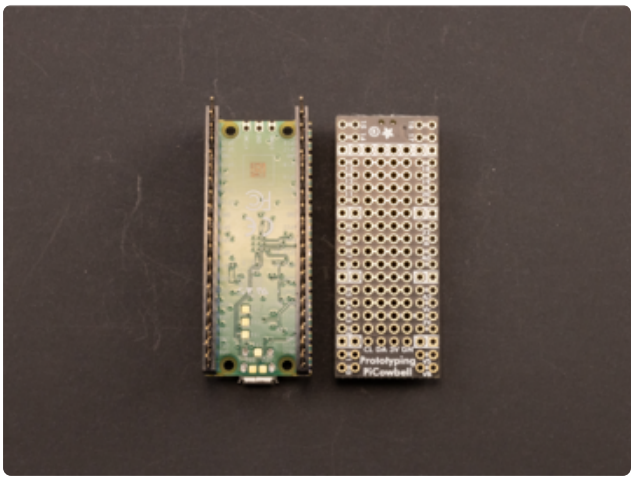
Follow the steps below to solder stacking headers to your PiCowbell.

Although these pages show the PiCowbell Proto, the soldering instructions are applicable for all PiCowbell boards.

Assembly Steps



Place a standard-header-soldered Pico upside down on the table, so the long side of the header pins are facing up. Press the female sockets of each stacking header onto one of the rows of standard headers attached to the Pico, until they are fully attached.

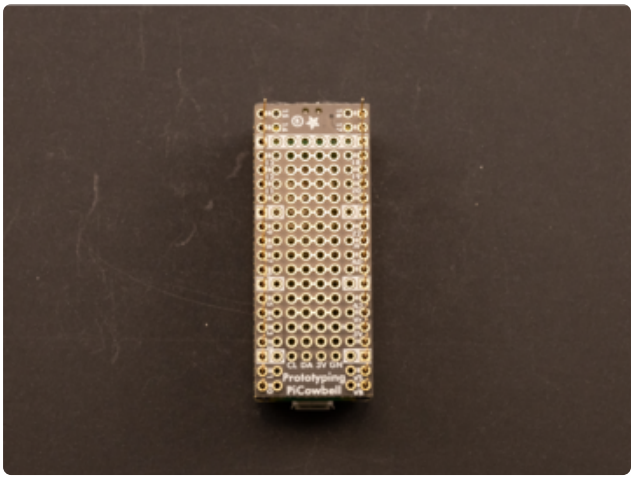


Ensure the PiCowbell is oriented correctly before beginning assembly. The PiCowbell should be top-down, so that you are looking at the bottom of the PiCowbell. **The STEMMA QT connector should be on the same end as the Pico USB connector, and the reset button should be on the opposite end with the Pico debug pins.**

The PiCowbell pins must match the pinout on the Pico.

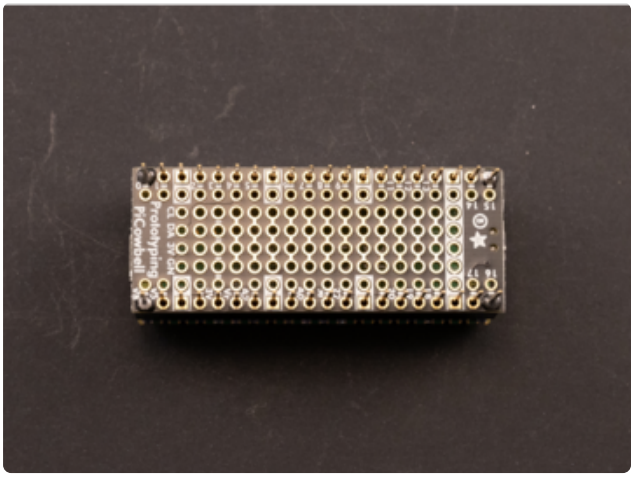
Remember, the pins are labeled on the bottom of the Pico. In this case, that works well because they are labeled on both sides of the PiCowbell, allowing for direct comparison before attaching the PiCowbell to the stacking header assembly.

Ensure the PiCowbell is oriented properly before beginning soldering! If you solder it on upside down or backwards, it will not function properly!

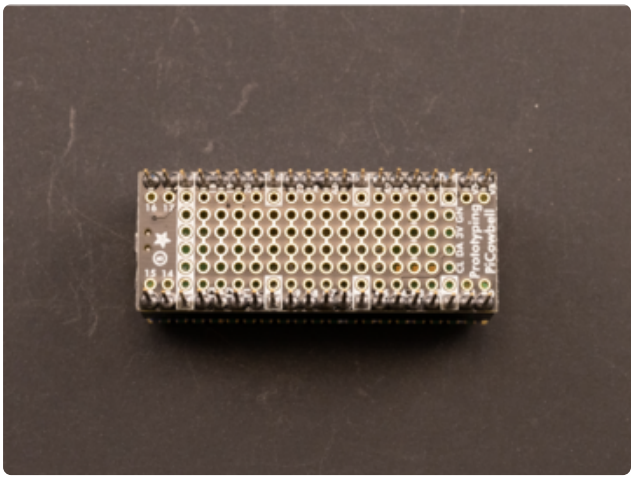


Press the PiCowbell onto the male pins sticking up from the stacking headers. You may need to push the stacking header pins in or out a bit to get the PiCowbell attached.

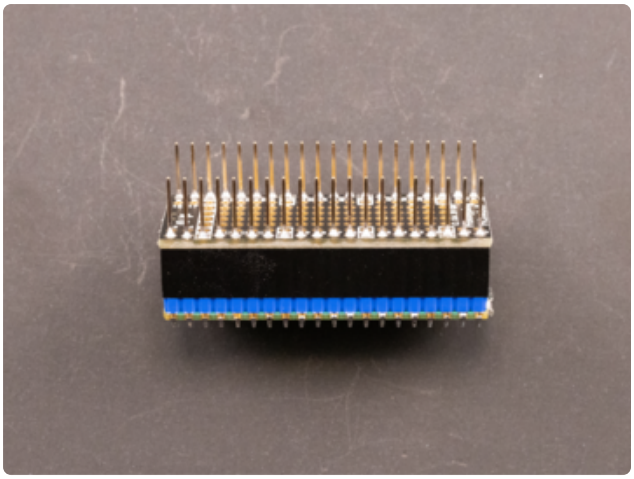
With the stacking header male pins sticking up, the bottom of the PiCowbell should be facing up as well.

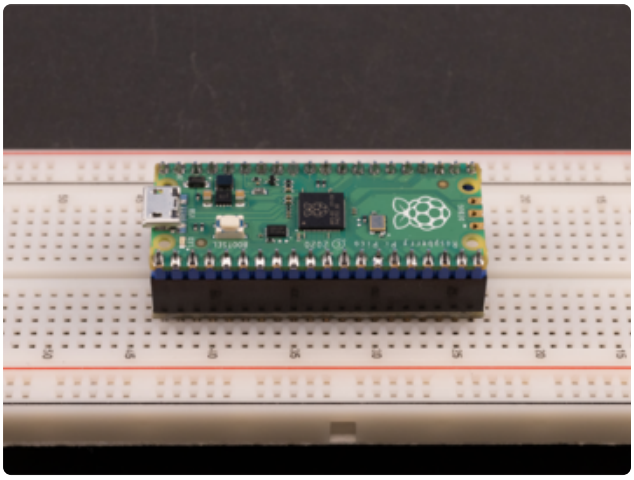


Solder the pins on each end of each stacking header, so that the opposite four corners of the PiCowbell are soldered on.



Solder the rest of the pins onto the PiCowbell.





You're done! Now you can attach the whole sandwich to a breadboard, have access to the pins via the breadboard, and still be able to use the PiCowbell as well.

Socket Headers

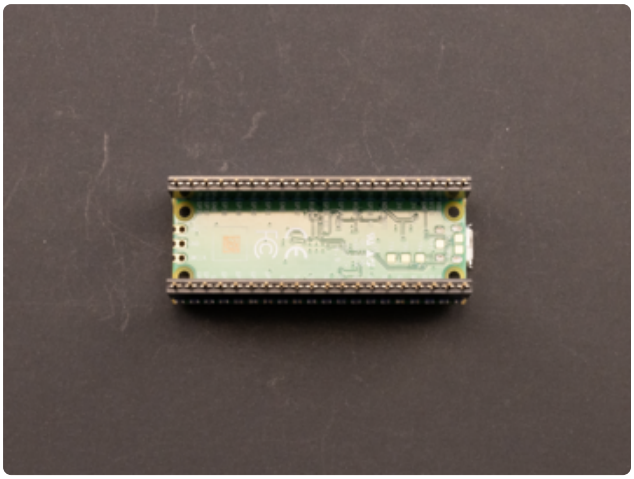
This PiCowbell assembly method uses female socket headers on the PiCowbell to create a standalone sandwich when attached to a Pico with standard male headers.

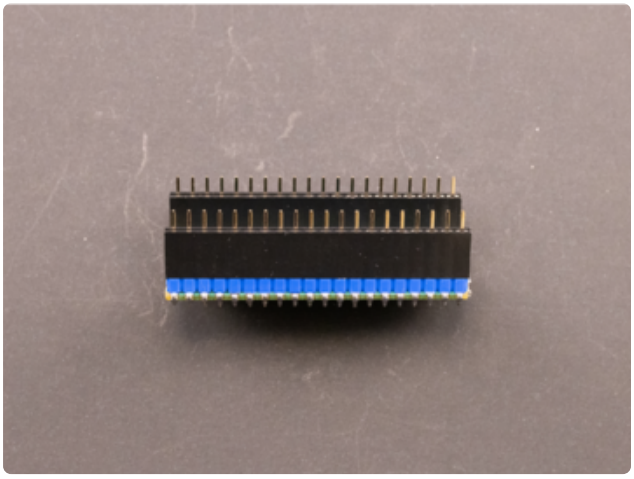
This page assumes you have already soldered standard male headers to your Pico. If you have not, please return to the Pico assembly page (https://adafru.it/18bQ) and follow the steps there.

Follow the steps below to solder socket headers to your PiCowbell.

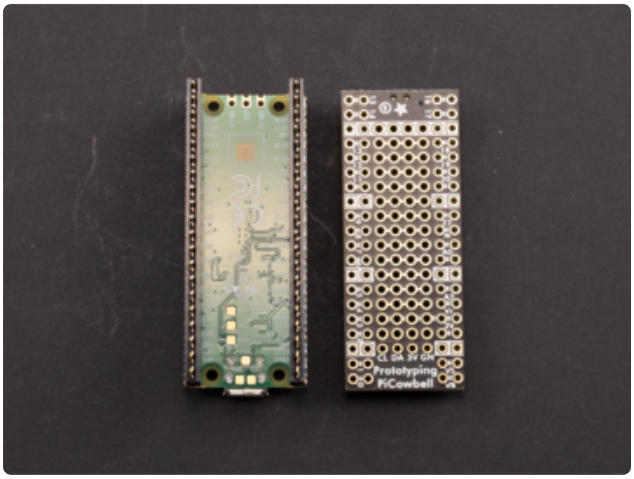
Although these pages show the PiCowbell Proto, the soldering instructions are applicable for all PiCowbell boards.

Assembly Steps





Place a standard-header-soldered Pico upside down on the table, so the long side of the header pins are facing up. Press the female sockets onto one of the rows of standard headers attached to the Pico, until both are fully attached.

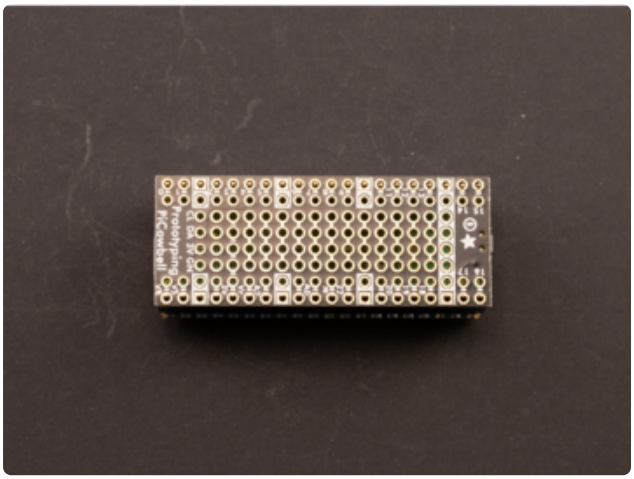


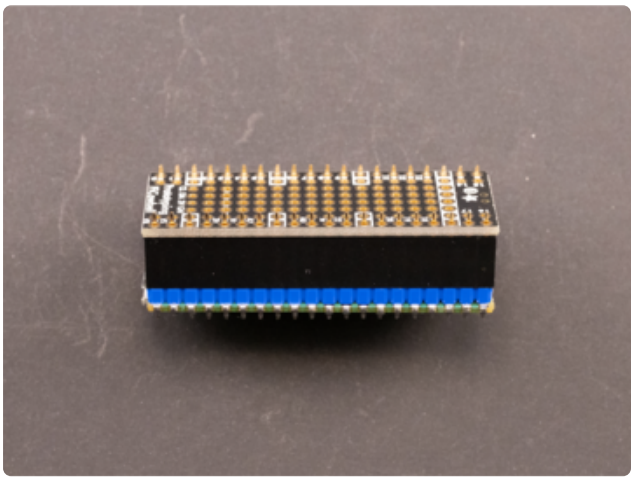
Ensure the PiCowbell is oriented correctly before beginning assembly. The PiCowbell should be top-down, so that you are looking at the bottom of the Cowbell. **The STEMMA QT connector should be on the same end as the Pico USB connector, and the reset button should be on the opposite end with the Pico debug pins.**

The PiCowbell pins must match the pinout on the Pico.

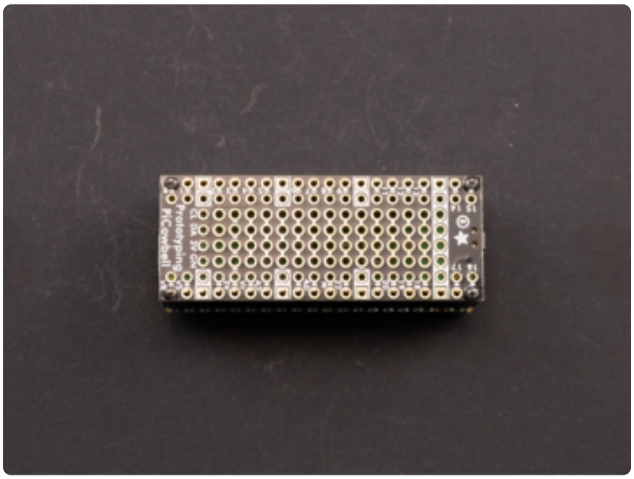
Remember, the pins are labeled on the bottom of the Pico. In this case, that works well because they are labeled on both sides of the PiCowbell, allowing for direct comparison before attaching the PiCowbell to the stacking header assembly.

Ensure the PiCowbell is oriented properly before beginning soldering! If you solder it on upside down or backwards, it will not function properly!

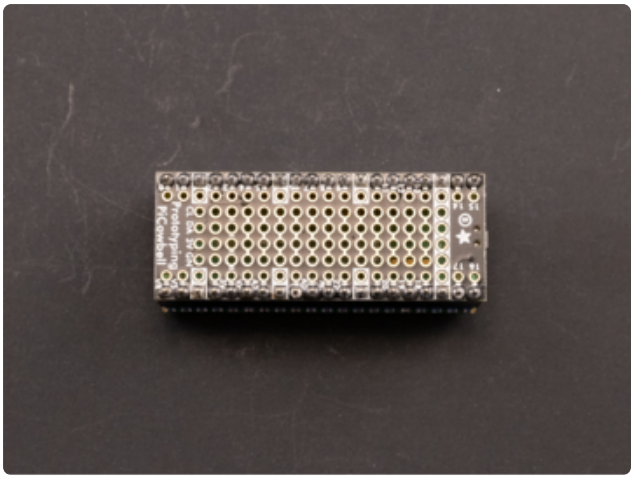




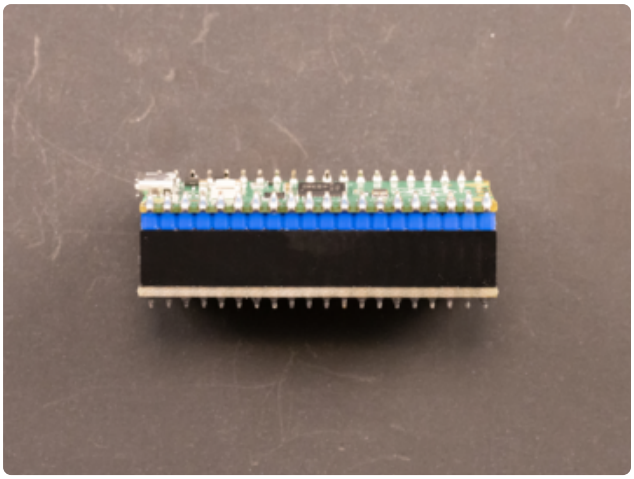
Press the PiCowbell onto the pins sticking up from the socket headers. You may need to push the stacking header pins in or out a bit to get the PiCowbell attached.



Solder the pins on each end of each socket header, so that the opposite four corners of the PiCowbell are soldered on.



Solder the rest of the pins onto the PiCowbell.



That's it, you're done!

CircuitPython

It's easy to use the **PiCowbell CAN Bus** with CircuitPython and the Adafruit_CircuitPython_MCP2515 (https://adafru.it/18Bm) module. This module allows you to easily write Python code that lets you utilize the MCP2515 CAN bus controller.

CircuitPython Usage

To use with CircuitPython, you need to first install the MCP2515 library, and its dependencies, into the **lib** folder onto your **CIRCUITPY** drive. Then you need to update **code.py** with the example script.

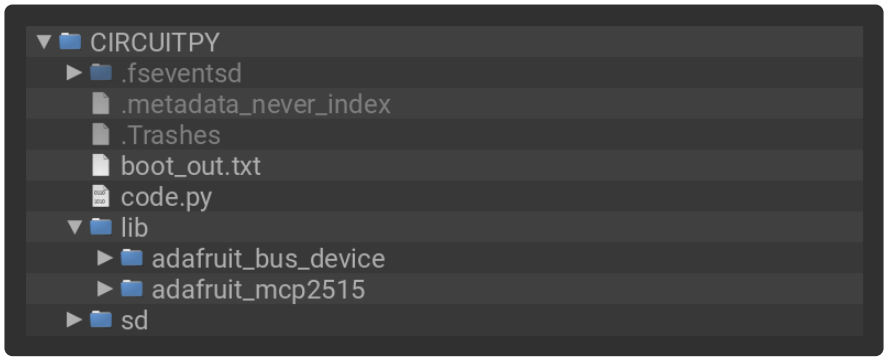
Thankfully, we can do this in one go. In the example below, click the **Download Project Bundle** button below to download the necessary libraries and the **code.py** file in a zip file.

Connect your Feather board to your computer via a known good data+power USB cable. The board should show up in your File Explorer/Finder (depending on your operating system) as a flash drive named **CIRCUITPY**.

Extract the contents of the zip file, and copy the **entire lib folder** and the **code.py** file to your **CIRCUITPY** drive.

Your **CIRCUITPY/lib** folder should contain the following folders:

- **adafruit_bus_device/**
- **adafruit_mcp2515/**



Simple Test Example

Once everything is saved to the **CIRCUITPY** drive, connect to the serial console (https://adafru.it/Bec) to see the messages printed out!

```
# SPDX-FileCopyrightText: Copyright (c) 2020 Bryan Siepert for Adafruit Industries
#
# SPDX-License-Identifier: MIT
'''Simple Test for the PiCowbell CAN Bus with Raspberry Pi Pico'''

from time import sleep
import board
import busio
from digitalio import DigitalInOut
from adafruit_mcp2515.canio import Message, RemoteTransmissionRequest
from adafruit_mcp2515 import MCP2515 as CAN


cs = DigitalInOut(board.GP20)
cs.switch_to_output()
spi = busio.SPI(board.GP18, board.GP19, board.GP16)

can_bus = CAN(
    spi, cs, loopback=True, silent=True
)  # use loopback to test without another device
while True:
    with can_bus.listen(timeout=1.0) as listener:

        message = Message(id=0x1234ABCD, data=b"adafruit", extended=True)
        send_success = can_bus.send(message)
```

```
        print("Send success:", send_success)
        message_count = listener.in_waiting()
        print(message_count, "messages available")
        for _i in range(message_count):
            msg = listener.receive()
            print("Message from ", hex(msg.id))
            if isinstance(msg, Message):
                print("message data:", msg.data)
            if isinstance(msg, RemoteTransmissionRequest):
                print("RTR length:", msg.length)
    sleep(1)
```

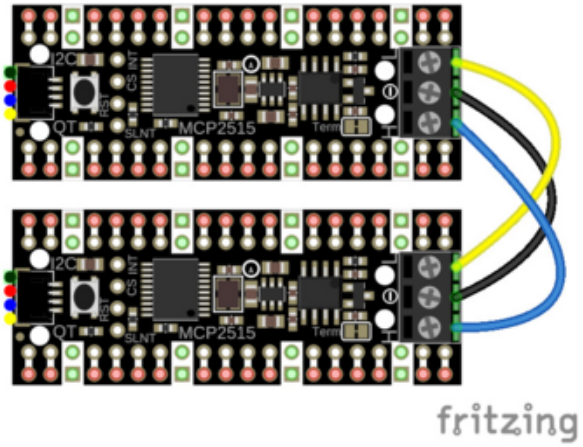
In the code, the instantiation of `can_bus` sets `loopback` and `silent` to `True`. This allows you to test the CAN Bus messaging without another CAN device.

```
can_bus = CAN(
    spi, cs, loopback=True, silent=True
)  # use loopback to test without another device
```

In the REPL, you'll see the byte array message be sent successfully every second.

Testing with Two CAN Bus PiCowbells

You can test with two CAN Bus PiCowbells by preparing two Raspberry Pi Pico boards with two CAN Bus PiCowbells. Prepare two Raspberry Pi Pico boards with two PiCowbell CAN Bus boards using your preferred header method described in the Assembly pages (https://adafru.it/18DH).



Once you have two Pico boards connected to CAN Bus PiCowbells, you can connect the CAN signals:

PiCowbell A CANH terminal block to **PiCowbell B CANH** terminal block **(blue wire)**
PiCowbell A - (ground) terminal block to **PiCowbell B -** (ground) terminal block **(black wire)**
PiCowbell A CANL terminal block to **PiCowbell B CANL** terminal block **(yellow wire)**

Upload the Project Bundle to both **CIRCUITPY** drives. In the code, change the `can_bus` instantiation on both Pico boards to have `loopback` and `silent` be `False`.

```
can_bus = CAN(
    spi, cs, loopback=False, silent=False
)  # use loopback to test without another device
```

When you run the code on both Pico boards, you can check each serial monitor window and see messages being exchanged between the two boards.

```
CircuitPython REPL
Auto-reload is on. Simply save files over USB to run them or enter REPL to disable.
code.py output:
Send success: True
1 messages available
Message from  0x1234abcd
message data: bytearray(b'adafruit')
Send success: True
1 messages available
Message from  0x1234abcd
message data: bytearray(b'adafruit')
Send success: True
1 messages available
Message from  0x1234abcd
message data: bytearray(b'adafruit')
```

Python Docs

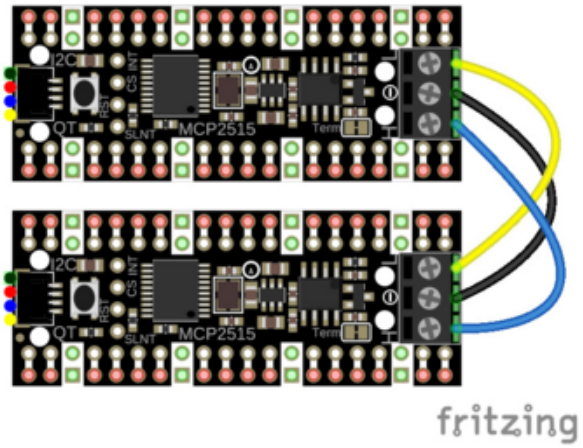
Python Docs (https://adafru.it/18Bp)

Arduino

Using the PiCowbell CAN Bus with Arduino involves wiring up the two PiCowbells to two Raspberry Pi Picos, installing the Adafruit_MCP2515 (https://adafru.it/18Bs) library and running the provided example code.

Wiring

Since CAN bus is a two-way communication protocol, you'll need two devices talking to each other over CAN. Prepare two Raspberry Pi Pico boards with two PiCowbell CAN Bus boards using your preferred header method described in the Assembly pages (https://adafru.it/18DH).

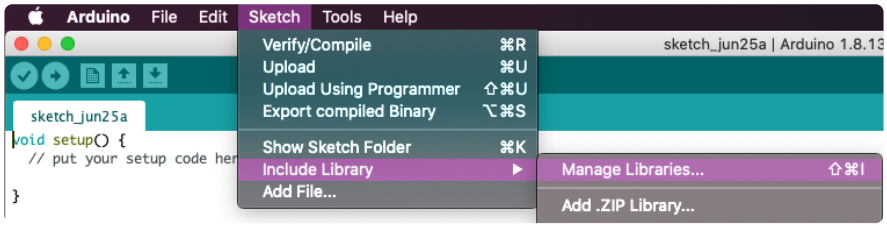


Once you have two Pico boards connected to CAN Bus PiCowbells, you can connect the CAN signals:

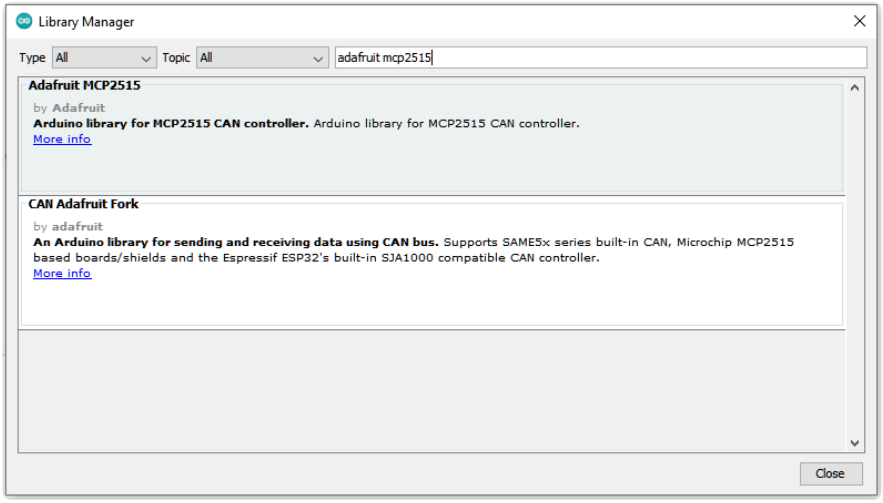
PiCowbell A CANH terminal block to **PiCowbell B CANH** terminal block **(blue wire)**
PiCowbell A - (ground) terminal block to **PiCowbell B -** (ground) terminal block **(black wire)**
PiCowbell A CANL terminal block to **PiCowbell B CANL** terminal block **(yellow wire)**

Library Installation

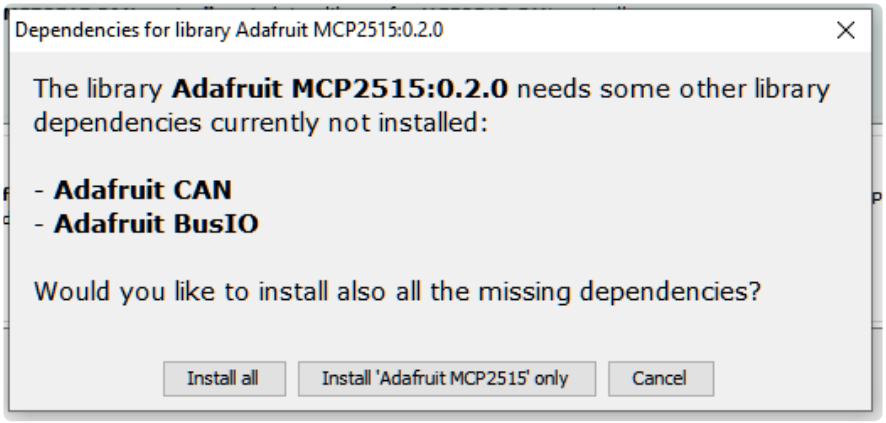
You can install the **Adafruit MCP2515** library for Arduino using the Library Manager in the Arduino IDE.



Click the **Manage Libraries ...** menu item, search for **Adafruit MCP2515** and select the **Adafruit MCP2515** library:



If asked about dependencies for any of the libraries, click "Install all".



If the "Dependencies" window does not come up, then you already have the dependencies installed.

If the dependencies are already installed, you must make sure you update them through the Arduino Library Manager before loading the example!

Example Send Code

```
/*
 * Adafruit MCP2515 FeatherWing CAN Sender Example
 */

#include <Adafruit_MCP2515.h>

#ifdef ESP8266
   #define CS_PIN    2
#elif defined(ESP32) && !defined(ARDUINO_ADAFRUIT_FEATHER_ESP32S2) && !
defined(ARDUINO_ADAFRUIT_FEATHER_ESP32S3)
   #define CS_PIN    14
#elif defined(TEENSYDUINO)
   #define CS_PIN    8
#elif defined(ARDUINO_STM32_FEATHER)
   #define CS_PIN    PC5
```

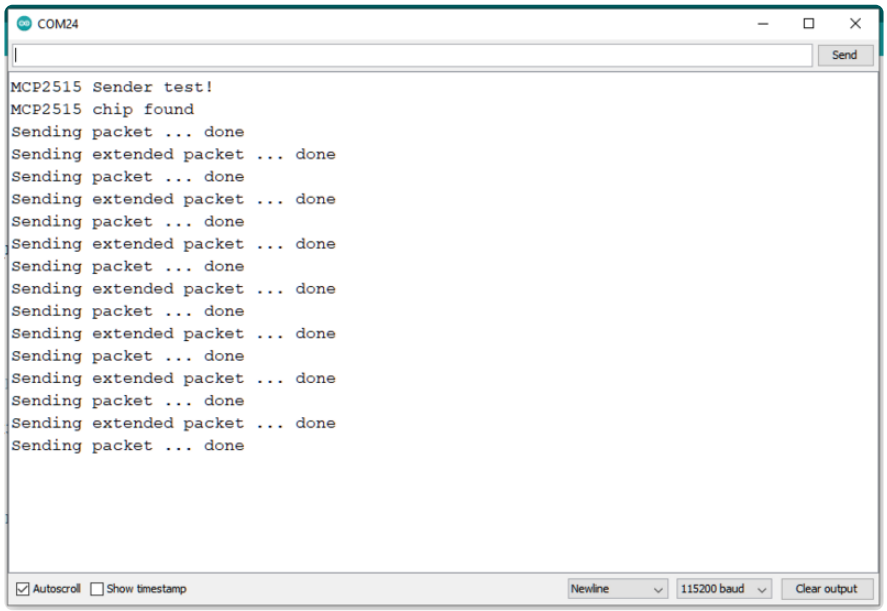
```
#elif defined(ARDUINO_NRF52832_FEATHER)  /* BSP 0.6.5 and higher! */
   #define CS_PIN    27
#elif defined(ARDUINO_MAX32620FTHR) || defined(ARDUINO_MAX32630FTHR)
   #define CS_PIN    P3_2
#elif defined(ARDUINO_ADAFRUIT_FEATHER_RP2040)
   #define CS_PIN    7
#elif defined(ARDUINO_ADAFRUIT_FEATHER_RP2040_CAN)
   #define CS_PIN    PIN_CAN_CS
#elif defined(ARDUINO_RASPBERRY_PI_PICO) || defined(ARDUINO_RASPBERRY_PI_PICO_W) //
PiCowbell CAN Bus
   #define CS_PIN    20
#else
    // Anything else, defaults!
   #define CS_PIN    5
#endif

// Set CAN bus baud rate
#define CAN_BAUDRATE (250000)

Adafruit_MCP2515 mcp(CS_PIN);

void setup() {
  Serial.begin(115200);
  while(!Serial) delay(10);

  Serial.println("MCP2515 Sender test!");

  if (!mcp.begin(CAN_BAUDRATE)) {
    Serial.println("Error initializing MCP2515.");
    while(1) delay(10);
  }
  Serial.println("MCP2515 chip found");
}

void loop() {
  // send packet: id is 11 bits, packet can contain up to 8 bytes of data
  Serial.print("Sending packet ... ");

  mcp.beginPacket(0x12);
  mcp.write('h');
  mcp.write('e');
  mcp.write('l');
  mcp.write('l');
  mcp.write('o');
  mcp.endPacket();

  Serial.println("done");

  delay(1000);

  // send extended packet: id is 29 bits, packet can contain up to 8 bytes of data
  Serial.print("Sending extended packet ... ");

  mcp.beginExtendedPacket(0xabcdef);
  mcp.write('w');
  mcp.write('o');
  mcp.write('r');
  mcp.write('l');
  mcp.write('d');
  mcp.endPacket();

  Serial.println("done");

  delay(1000);
}
```

Example Receive Code

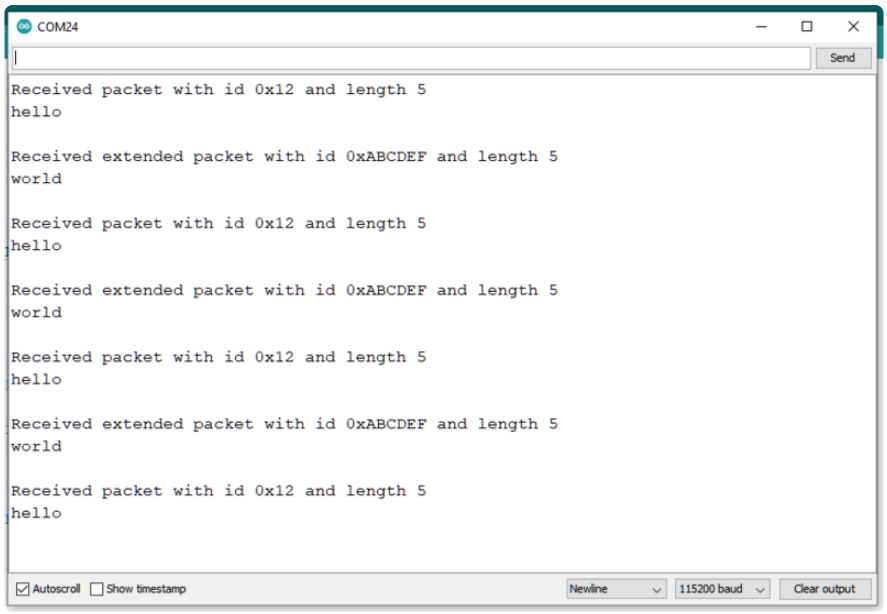
```
/*
 * Adafruit MCP2515 FeatherWing CAN Receiver Example
 */

#include <Adafruit_MCP2515.h>

#ifdef ESP8266
   #define CS_PIN    2
#elif defined(ESP32) && !defined(ARDUINO_ADAFRUIT_FEATHER_ESP32S2) && !
defined(ARDUINO_ADAFRUIT_FEATHER_ESP32S3)
   #define CS_PIN    14
#elif defined(TEENSYDUINO)
   #define CS_PIN    8
#elif defined(ARDUINO_STM32_FEATHER)
   #define CS_PIN    PC5
#elif defined(ARDUINO_NRF52832_FEATHER)  /* BSP 0.6.5 and higher! */
   #define CS_PIN    27
#elif defined(ARDUINO_MAX32620FTHR) || defined(ARDUINO_MAX32630FTHR)
   #define CS_PIN    P3_2
#elif defined(ARDUINO_ADAFRUIT_FEATHER_RP2040)
   #define CS_PIN    7
#elif defined(ARDUINO_ADAFRUIT_FEATHER_RP2040_CAN)
   #define CS_PIN    PIN_CAN_CS
#elif defined(ARDUINO_RASPBERRY_PI_PICO) || defined(ARDUINO_RASPBERRY_PI_PICO_W) //
PiCowbell CAN Bus
   #define CS_PIN    20
#else
    // Anything else, defaults!
   #define CS_PIN    5
#endif

// Set CAN bus baud rate
#define CAN_BAUDRATE (250000)

Adafruit_MCP2515 mcp(CS_PIN);

void setup() {
  Serial.begin(115200);
  while(!Serial) delay(10);

  Serial.println("MCP2515 Receiver test!");

  if (!mcp.begin(CAN_BAUDRATE)) {
    Serial.println("Error initializing MCP2515.");
    while(1) delay(10);
  }
  Serial.println("MCP2515 chip found");
}

void loop() {
  // try to parse packet
  int packetSize = mcp.parsePacket();

  if (packetSize) {
    // received a packet
    Serial.print("Received ");

    if (mcp.packetExtended()) {
      Serial.print("extended ");
    }

    if (mcp.packetRtr()) {
      // Remote transmission request, packet contains no data
      Serial.print("RTR ");
```

```
    }

    Serial.print("packet with id 0x");
    Serial.print(mcp.packetId(), HEX);

    if (mcp.packetRtr()) {
      Serial.print(" and requested length ");
      Serial.println(mcp.packetDlc());
    } else {
      Serial.print(" and length ");
      Serial.println(packetSize);

      // only print packet data for non-RTR packets
      while (mcp.available()) {
        Serial.print((char)mcp.read());
      }
      Serial.println();
    }

    Serial.println();
  }
}
```

Upload the Example Send Code to the first Pico and then upload the Example Receive Code to the second Pico. When you open the Serial Monitor for the Pico running the Send code, you'll see confirmations that packets have been sent.



When you open the Serial Monitor for the Pico running the Receive code, you'll see the messages coming in via CAN.



Arduino Docs

Arduino Docs (https://adafru.it/18Bs)

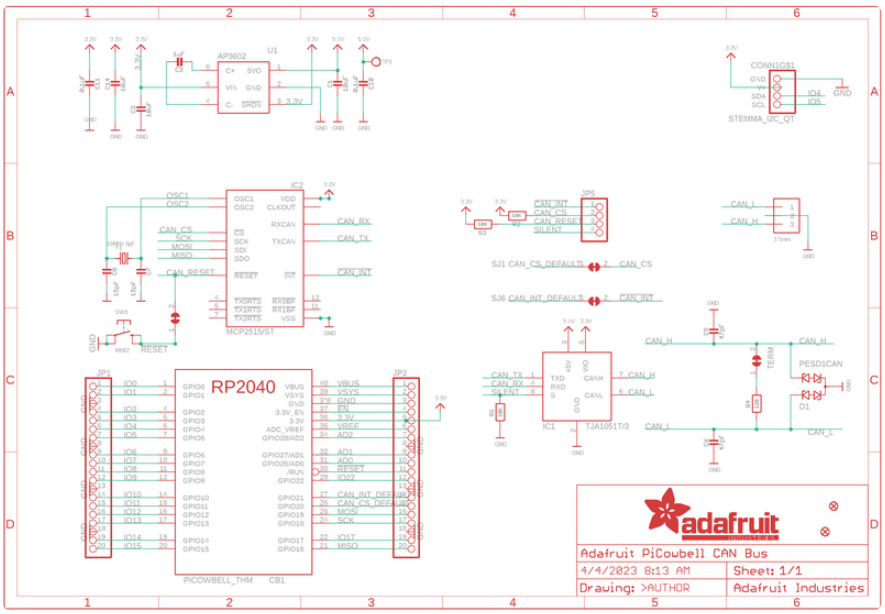
Downloads

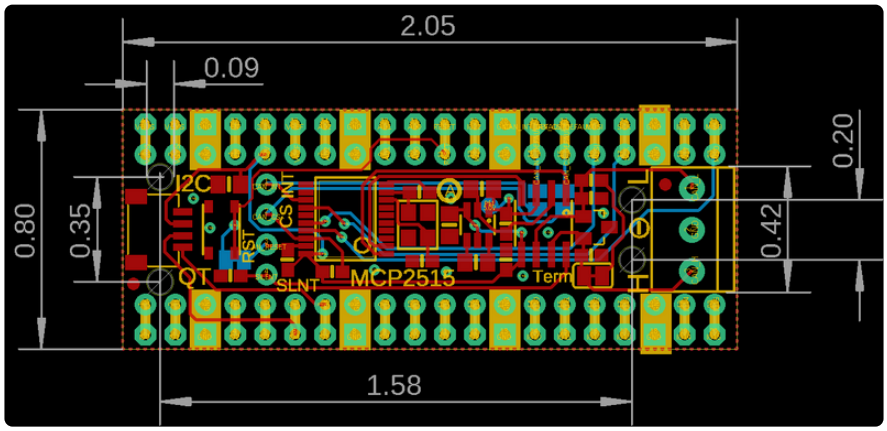
Files

- MCP2515 Datasheet (https://adafru.it/18Bv)
- TJA1051/3 Datasheet (https://adafru.it/18Bw)
- EagleCAD PCB files on GitHub (https://adafru.it/18DI)
- Fritzing object in the Adafruit Fritzing Library (https://adafru.it/18DJ)

Schematic and Fab Print

Dimensions are in inches.


RP2040
MCP2515/ST
TJA1051T/3
adafruit
Adafruit PiCowbell CAN Bus
4/4/2023 8:13 AM
Sheet: 1/1
Drawing: >AUTHOR
Adafruit Industries


2.05
0.09
0.80
0.35
0.20
0.42
1.58
I2C
CS
INT
RST
QT
SLNT
MCP2515
Term