UF UNIVERSITY of FLORIDA
Herbert Wertheim College of Engineering

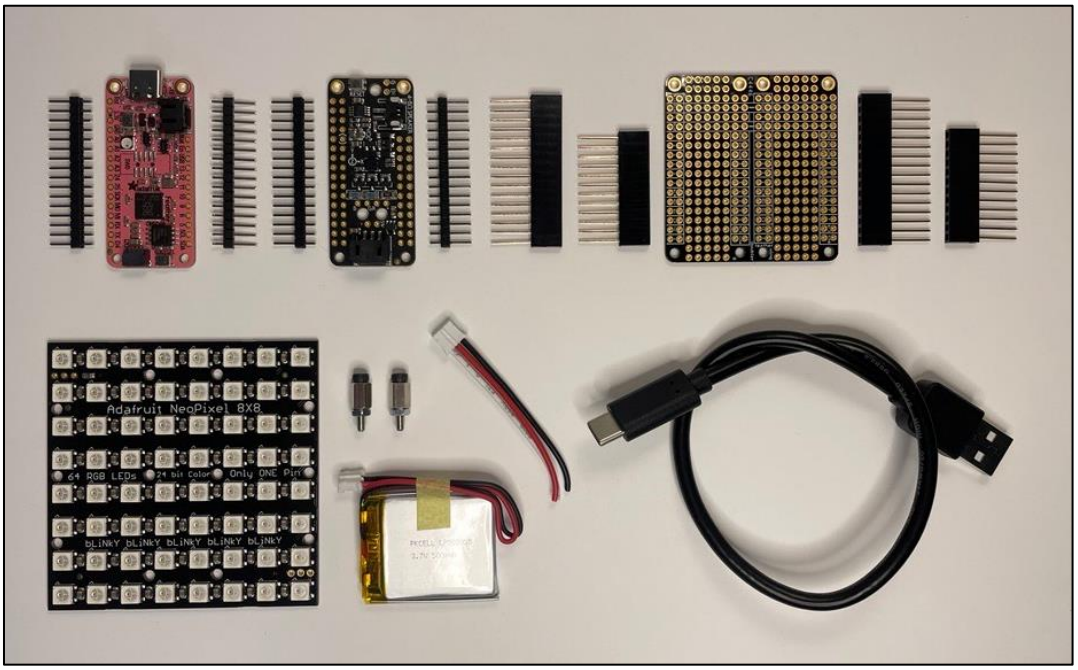
Computer Engineering
CEN3907C

Hardware Introduction

Overview

This term, you will be provided with a small hardware kit for exploring embedded firmware development. The components in this kit were specifically selected to enable the creation of fun, extensible, modular, and advanced projects. We hope that you will enjoy working with these devices and find them inspiring and suitable for a wide range of current and future projects.

What's Included in Your Kit



The provided kit includes the following components:

- Adafruit Feather RP2040 Development Board
- Adafruit Propmaker FeatherWing
- Adafruit FeatherWing Doubler
- Adafruit 8x8 NeoMatrix (addressable RGB LED matrix)
- Two M2.5 standoffs with screw and nut
- Three-position JST PH Connector Pigtail
- 500mAh single-cell LiPo Battery
- USB Cable (type-A to type-C)

The main and peripheral boards in this kit follow the Feather Specification, designed by Adafruit Industries in New York City. This specification defines a common board shape and pinout to create a modular, cross-vendor ecosystem of development boards. This is beneficial, as it allows makers and engineers to mix-and-match a variety of boards that serve different purposes. For instance, if you are interested in IoT devices, you could easily swap the Feather RP2040 for a Particle Boron to give your project LTE connectivity – while still being able to use the LED Matrix and Propmaker FeatherWing.

The Feather Specification also dictates that all main boards have built-in single-cell LiPo charging (the battery must connect using a two-position Molex PH connector), so that all Feather-based projects can be mobile.

Assembling Your Kit

Your kit requires some soldering and mechanical assembly, as described below.

1. Solder the male headers onto the Feather RP2040 and the Propmaker. Make sure that the long portion of the pins are on the bottom side of the boards. A breadboard can be helpful to keep the pins positioned nicely while soldering. One of the headers for each board will have to be shortened from 16 positions to 12 positions

Figure 2: Shorten one header to 12-position

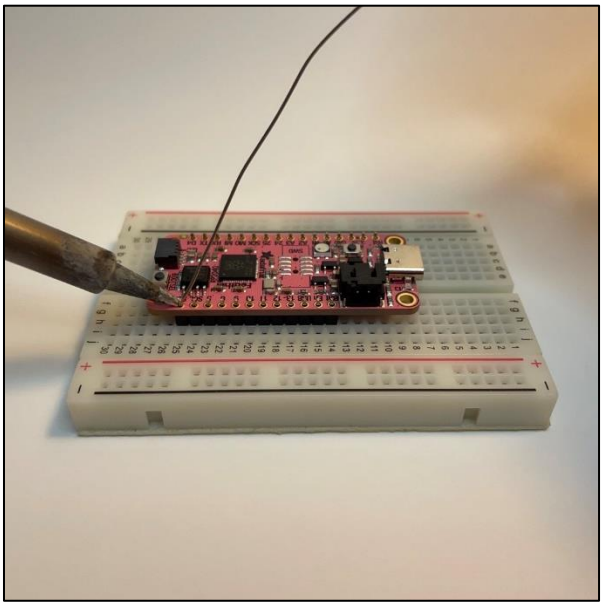


Figure 1: Solder all 28 pins on each board

2. Solder the female headers onto the FeatherWing Doubler. Make sure that the plastic portion of the headers is on the silkscreen-side of the board, and that the 16-pos headers are on the side with the larger silkscreen area. I like to solder one pin on each header, then lift the board and make sure the alignment is nice before I solder the rest. When finished, clip the excess metal leads off the bottom of the board.

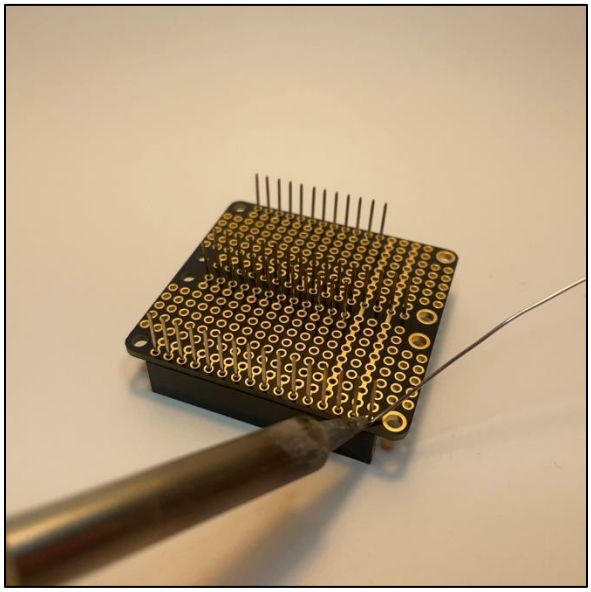


Figure 4: Soldering the female headers to the Doubler

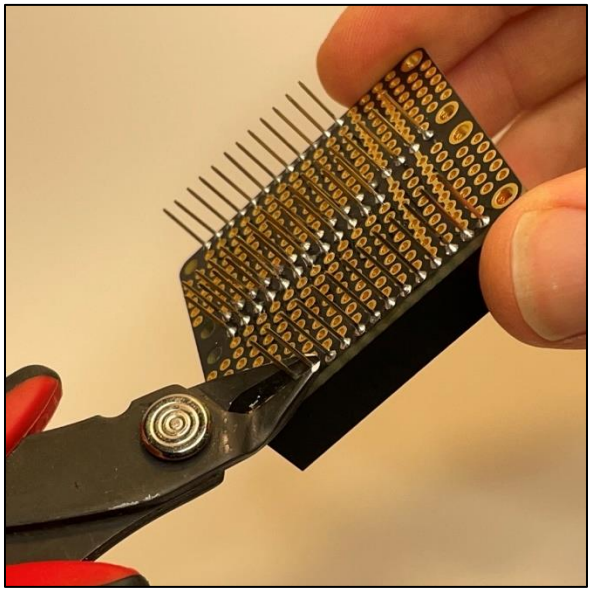


Figure 3: Clipping the excess leads

3. Use the nuts to secure the standoffs to the bottom side of the Doubler.
4. Use a piece of double-sided sticky foam to secure the battery to the bottom side of the Doubler.

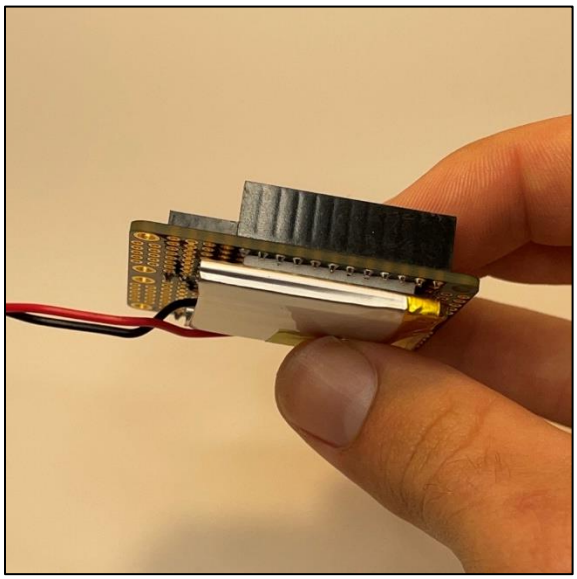


Figure 6: Securely attach the battery to the Doubler

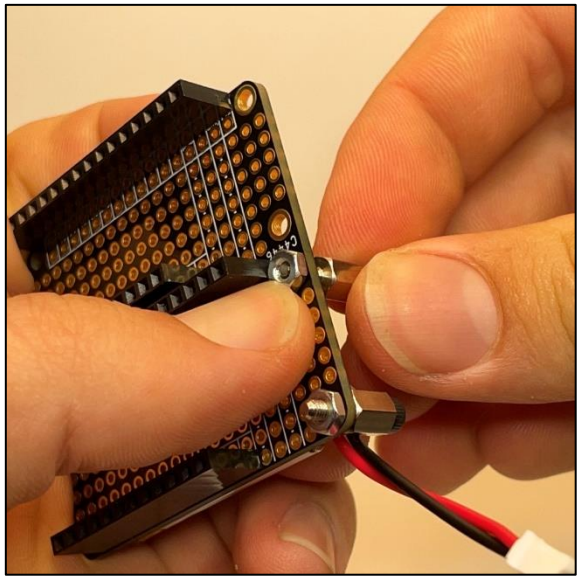
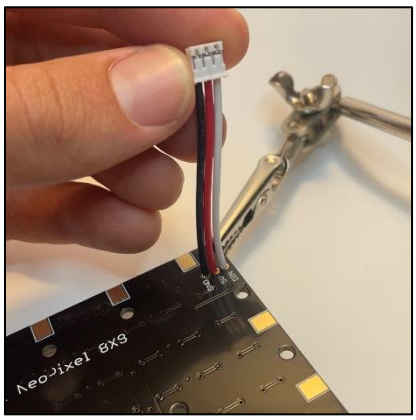
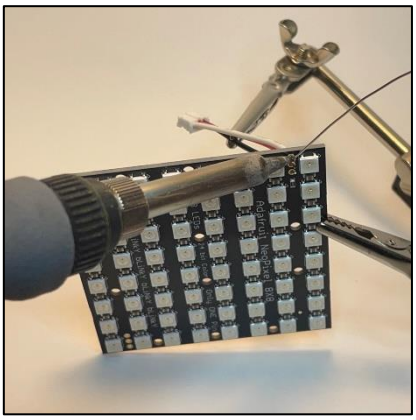


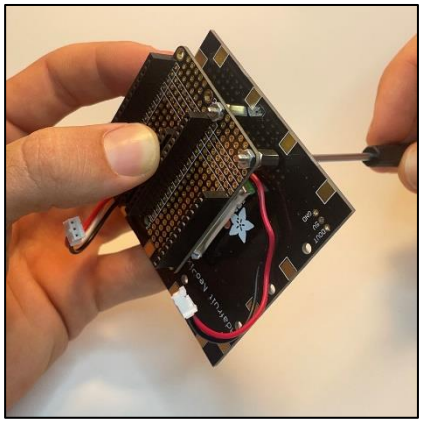
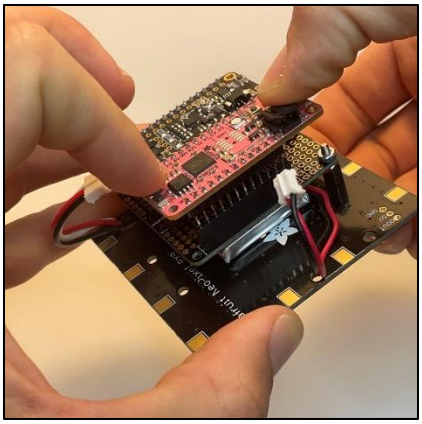
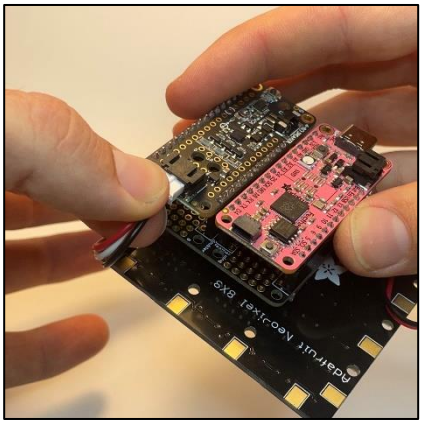
Figure 5: Attach the standoffs to the Doubler

5. Solder the 3-pos JST PH pigtail to the NeoMatrix. Strip approximately 3mm of insulation off of the wires, insert them into the holes on the NeoMatrix, and solder them in place. The black wire should be in the GND hole, the red in the 5V hole, and the white in the DIN hole. Using a pair of helping-hands here can ease the process.



6. Place the Doubler as shown onto the NeoMatrix and use the screws to secure.
7. Insert the Feather RP2040 and the Propmaker into the headers of the Doubler.
8. Attach the 3-pos PH connect into the Propmaker.

9. To power on the system, insert the battery connector into the Feather RP2040.

Options for Programming

This hardware has many options for programming, three of which are outlined below. Flashing the board with compiled code will be the same for all methods, via the **build-in permanent ROM UF2 bootloader**. To enter this bootloader, simply hold down the *BOOTSEL* button while powering-on or resetting the board.

C/C++

To program the RP2040 in C or C++, the easiest way to get started is the official Pico SDK. Chapter 2 of the getting started guide has detailed info about how to install and configure the build system; however, this guide is specific to the Raspberry Pi Pico, the official development board by RPi. There is a small change that needs to be made to build code for the Feather RP2040. The development computer should be running a native Linux distro or WSL2 on a Windows machine. It might be possible to setup the toolchain on a native Windows installation, but that will not be supported in this class.

- After cloning both the SDK and the pico-examples repository (and properly initializing the submodules), as well as installing the toolchain components, navigate into the pico-examples repo. We must change the *blink* example to target the LED pin on our board. On line 13 of the example, change *PICO_DEFAULT_LED_PIN* to the immediate value *13*. *Note*: You can see which GPIO the led is connected to on the Feather RP2040 by looking at the schematic!
- In the root of the pico-examples repo, create and *cd* into a *build* directory. Then, export an environment variable called *PICO_SDK_PATH* that points to the location where you cloned the SDK. If the examples and SDK repos are siblings, that command will look like the following:

```
carstentb@cise-dsk-5952cc /m/c/U/c/D/P/r/p/build (master)> export PICO_SDK_PATH=../../pico-sdk
```

- Now run the following command, which will configure the build system for the Feather RP2040 board. *Note*: I have experienced this *cmake* command taking a long time to complete on WSL2, whereas it takes a very short time on native linux. I'm not sure why this happens, but it is important to let the process finish.

```
carstentb@cise-dsk-5952cc /m/c/U/c/D/P/r/p/build (master)> cmake -DPICO_BOARD=adafruit_feather_rp2040 ..
```

- To build all examples, simply type *make* once the above command has completed. Alternatively, to build only a specific example, *cd* into that example's directory (within the newly created *build* directory), and then type *make*.
- To load one of the executables, simply reboot the board into the UF2 bootloader, and drag-and-drop the corresponding *.uf2* executable onto the filesystem.

Rust

The embedded Rust ecosystem is rapidly maturing into a viable option for programming microcontrollers. The Embedded Rust Book is a great place to start learning about how and why you might want to program an embedded system with Rust.

To program the RP2040 in embedded Rust, follow the below steps:

- Install the *rustup* tool to install and configure the Rust toolchain. This can be accomplished by following the instructions available here. Some things to be aware of:
    - **On Windows**: The Rust toolchain relies on the Microsoft C++ Build Tools. Install those first if you don't already have them on your system. *Note – if you have visual studio or have done other C++ development on Windows, you might already have these!*
    - **On WSL2**: The Rust toolchain can be installed on WSL2 by using the curl command found on the link above. This option does not require the Microsoft C++ Build Tools, as it will instead make use of the GNU compiler collection. However, there are some additional Ubuntu packages needed to run the *elf2uf2* cargo runner, namely *libudev-dev* and *pkg-config*.
    - **On native Linux**: Alternatively, if available, use your system's package manager (Arch Linux repos contain an official package for *rustup*).
- Execute the following commands to configure the Rust toolchain. *Note*: the commands shown are run from Windows Powershell, but identical commands should work on Unix-like systems.

```
PS C:\Users\carstentb\Documents\Projects\rp2040-dev> rustup default stable
```

```
PS C:\Users\carstentb\Documents\Projects\rp2040-dev> rustup target add thumbv6m-none-eabi
```

```
PS C:\Users\carstentb\Documents\Projects\rp2040-dev> cargo install elf2uf2-rs
```

- The last command installs the *cargo runner* we will use, which is responsible for converting the *.elf* executable into a *.uf2* file for use with the RP2040 Bootloader.
    - **On Windows and native Linux**, after we put our Feather into UF2 mode, invoking *cargo run* will automatically generate and load our code onto the board.
    - **On WSL2** we need to change the runner in the *.config/cargo* file of the below repository. Near the bottom of the file, remove the *-d* flag from the *elf2uf2* command. This change is needed since WSL2 does not have access to the USB devices on Windows. The modified runner will still create a *.uf2* file in the *target/thumbv6m-none-eabi/debug* directory, but you will need to manually copy or drag it into the mounted UF2 drive.
- Download (or fork and clone) this example code. After putting the Feather RP2040 into UF2 mode, run *cargo run* from the command line. This should build and flash the chip, and you should see the build-in LED blink.

CircuitPython

CircuitPython is a fork of MicroPython maintained by Adafruit Industries and the community. It includes a (mostly) CPython-compatible interpreter that runs on the microcontroller, as well as a set of official libraries.

To flash the Feather RP2040 with CircuitPython and start writing Python scripts, follow these steps:

- Download the most recent stable version of CircuitPython for your board. Your browser will save the file with the *.UF2* suffix.
- Activate the UF2 bootloader on your Feather board. Plug the board into your computer via USB. Then, hold down the *BOOTSEL* button while you press the *RESET* button once. Your computer should recognize and mount the board as an external filesystem, with a name similar to *RPI-RP2*.
- Then, simply drag-and-drop the CircuitPython UF2 file you downloaded in step 1 onto the newly mounted RP2040. After copying the file over, the board should reboot and be re-mounted as another drive on your development computer, named something like *CIRCUITPY*. This filesystem exposes the flash chip on the board as a USB mass storage device.
- To test that your flashing procedure was successful, try to connect to the REPL on the board via the USB serial port. On Mac or other Unix-like systems, *minicom* is a good approach to use; on Windows, *putty* or *nRF Terminal* via VSCode are good approaches.
- For more information, see the detailed overview on the Adafruit Learning System.