Introduction

In this workshop you will be introduced to MicroPython/CircuitPython, a variants of the python programming language that runs on the “bare silicon” of microprocessor chips. There are several advantages to running python rather than Arduino C++ on a microcontoller. First, python is easier to learn than C++, second, since python is a scripted language rather than a compiled language, development, modification, and debugging is easier.

The core python variant developed for microcontollers was MicroPython, often abbreviated as *uPy*. Its development began in 2013. Shortly after, AdaFruit forked CircuitPython (abbreviated as *CPy*) from MicroPython and developed many *modules* (python for a library) for using its microcontollers and hardware.

One big advantage with uPy/CPy is that they have an interactive interface with the microcontroller called the *REPL* for *R*ead – *E*valuate – *P*rint – *L*oop. You can enter and run one statement at a time, define functions and run them, import modules and try them – all from an interactive prompt. This makes developing code easy. I often try one line at a time, then add it to my code. I also use interactive help to find out what python and modules can do.

A second advantage of uPy/CPy is the support for many input-output pins on the microcontoller, hardware breakout boards and other hardware extensions. For example, in this workshop we will use light sensors, ADC (Analog to Digital Converters) on the microcontoller, light sensors, and stepper motor controllers.

The workshop title has a little lie in it – we will use CircuitPython rather than MicroPython. CPy has some nice advantages for us.

- The microcontoller will automatically mount as a USB drive. This allows you to edit the file directly on the microcontoller without an extra download step.
- There are over 250 modules (libraries) available for working with internal and external sensors and hardware.
- It is available for almost 300 different microcontoller boards.

Disadvantages:

- No direct interrupt programming like there is in MicroPython. This restricts ability to do fast timing, counting and coincidence measurements. Some indirect access is available.
- MicroPython is available on more microcontollers than CircuitPython.

Please Bring

I will provide the microcontollers, breadboards, breakout boards and other hardware for you to build one of the workshop projects. There will be up to 18 people in the workshop, so I need people to come prepared to work with that many participants. Here is what you need to make the workshop run smoothly.

- A laptop. Windows, Macs, or Linux will work.
- Install *thonny*, an editor and development environment for uPy/CPy. I recommend installing it from the Thonny website **`https://thonny.org/`**.
- On a Windows PC, install *putty* from **`https://the.earth.li/~sgtatham/putty/latest/w64/putty-64bit-0.76-installer.msi`**.
- On a Mac I use **`tio`**. I installed it using the **`brew`** package manager (I highly recommend **`brew`**.) for Mac’s. In a terminal type **`brew install tio`**.
- On a Linux machine I use *putty*.
- Wire strippers. I use the cheap ones like **`this`** for less than $9. You will be wiring circuits on a breadboard.

- A USB-C cable. The microcontollers have a female USB-C connector you need to connect to. ***Make sure your cable is* NOT *a charge-only cable! You need a data cable.*** Charge only cables are from the Pit of Hell. They have caused countless nightmares for makers and companies that sell stuff to makers.
- Willingness and skill to wire breadboard circuits. See the project handouts for examples.
- **Optional**: small wire cutter and small needle-nosed pliers. You can cut the wire with the wire strippers and bend the wire with your fingers, but sometimes it goes faster with a wire cutter and pliers.
- An account (free) with **`https://io.adafruit.com/`**.

I Will Provide

All necessary microcontollers, sensors, breakout boards, wire, cases, and other hardware. Small screwdrivers.

Planned Projects

I will bring kits for self-contained projects.

All participants will receive:

- Half breadboard.
- 3D printed case.
- Microcontroller Wemos/Lolin S2Mini, $4 each.

One of the three different project kits: *Four-Wire Conductivity versus Temperature*, *Laser Beam Profiler,* or *Alpha Particle Counting*.

Projects for Everyone

- Everyone will connect the microcontoller to WiFi and have it serve a web page that displays the CPU temperature and has a button so you can turn its LED on and off.
- Everyone will start by setting up the microcontoller, connecting it to WiFi and posting the CPU temperature to a cloud account at **`https://io.adafruit.com/`**.

Breakout Projects

After the initial introduction and first projects, you will either choose or be assigned a breakout project. You will build the hardware and I will provide the CPy code to run it.

- ***Four wire conductance device*** with a thermocouple sensor. The conductance measurement can be used to measure the I-V curve of a diode, or the resistance of a sample.
  - I measured the thermal coefficient of resistance of a resistor.
  - The thermocouple can used down to 75 K, so you can use this to measure the transition of a high temperature superconductor. This will **not** be done during the workshop.
- ***Alpha particle detector***. PIN photodiodes with the glass window removed can directly sense alpha particles. You will build an op amp circuit to amplify the photodiode current so it triggers a counting pin on the microcontoller. You will use an ^{241}Am source from a smoke detector as a source.
  - I use this as a first project in my Advanced Lab course. The students learn to build a system and get familiar with non-Gaussian statistics and alpha particle physics.
- ***Laser beam profiler***. The microcontroller will use a stepper motor to translate a razor blade across a laser beam. The data can be used extract the width and profile of the beam. This data can be fit to determine if the beam is Gaussian and extract the Gaussian width.

Getting Started

You will start by downloading *Thonny*, an open source IDE (programming environment) and *rshell* a file transfer program. You will start using interactive python commands *via* the REPL (Read – Evaluate – Print – Loop). Then use the editor to write code and upload the files to the a microcontoller for execution.

Next you will upload network manager code that lets you connect to a WiFi network without having the TinyPICO plugged in. Another task is to connect a battery that lets the TinyPICO operate remotely.

Next there are options for what to implement. Choices are:

- An environmental sensor device that uploads temperature, pressure, and humidity to a data logging cloud service, **adafruit.io**.
- An alpha particle counter using a PIN photodiode. This requires an op amp circuit for pulse shaping, and an interrupt timer implemented in MicroPython.
- Measuring the I-V (current – voltage) curve of a diode. The TinyPICO has a genuine DAC. (Many microcontroller only have PWM (Pulse Width Modulation) you have to low-pass to get an analog voltage.) A simple circuit can be wired to measure the I-V curve of a diode or LED.
- Laser beam profiler.

This is a make-and-take project. The cost of materials is about $40 depending on the project.

Ways To Get Deeper

I think a great way to get more experienced is to buy an Adafruit ESP32-S2 TFT Feather for $25. It has a built in WiFi and a 240x135 pixel color display. Then follow the great tutorial **https://learn.adafruit.com/adafruit-esp32-s2-tft-feather**. It take you through many topics like connecting to WiFi, posting and reading using the MQTT protocol that is widely used for IOT (Internet Of Things), expansion boards, and more.

Other resources are the ***Welcome to CircuitPython*** beginner guide and the more in depth ***CircuitPython Essentials*** from AdaFruit. All three documents are on the GitHub site for the workshop.

Appendix - Troubleshooting Library Code

One of my proverbs is “Nothing works the first time.” A critical skill students have to learn is how to find and correct mistakes. For hardware, the go to instrument should be an oscilloscope. Microcontoller projects also involve software, so how do you debug software?

Python itself is easy to debug by putting in extra print statements. I very rarely have situations where I need to use the Python debugger.

Most of my development time for CPy is figuring out which libraries to use and what to do when (not if!) they don’t work. What follows is a case study in getting a thermocouple breakout board to work. The last part of this gets in depth in python code, but it is how I got the board to work. I hope there are some helpful hints for you along the way.

Initial Attempt

I bought **these** Type K thermocouple breakout boards from Adafruit. They interface using a SPI protocol. The master is the microcontoller and the slave is the thermocouple board. The documentation for this board says it only communicates *to* the

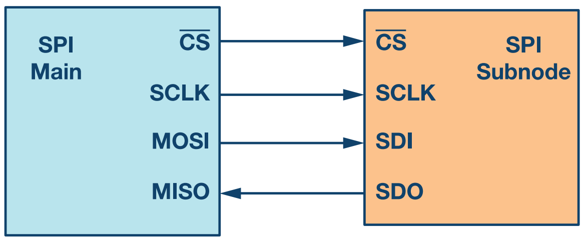

Figure 1: The SPI bus. The CS is chip select, SCLK is the clock pulse, MOSI is Master Out Slave In, MISO is opposite direction.

master so the MOSI connection is not used, so I only need to wire the CS, CLK, and MOSI connections. These are clearly marked on the breakout board.

First problem: which pins do I use on the microcontoller?

Solution: Connect to the microcontoller and type

Note: the output is edited.

```
>>> import board
>>> help(board)
object <module 'board'> is of type module
  __name__ -- board
  board_id -- lolin_s2_mini
  BUTTON -- board.BUTTON
  IO0 -- board.BUTTON
  A0 -- board.IO3
  IO5 -- board.IO5
  D0 -- board.IO5
  IO7 -- board.IO7
  SCK -- board.IO7
  D5 -- board.IO7
  IO9 -- board.IO9
  MISO -- board.IO9
  D6 -- board.IO9
  IO11 -- board.IO11
  MOSI -- board.IO11
  D7 -- board.IO11
  LED -- board.LED
  IO15 -- board.LED
  IO35 -- board.IO35
  SDA -- board.IO33
  SCL -- board.IO35
  D1 -- board.IO35
  I2C -- <function>
```

This list tells me the default pins for SPI are SCK – pin board.IO7, MISO – pin board.IO9. Also notice that three of the SPI pins are grouped as pins IO7, 9, and 11. The pins SDA and SCL are for the I2C protocol. The (CS) chip select is usually independent of the other SPI pins. The reason is that SPI can only connect to one device at a time so the user has to have a different connection to each device on an SPI bus.
How did I know how to do this? Experience. And
Hint:
When connected to CPy, type `help(“modules“)`. This is a list of all the modules already present in CPy on your microcontroller. (It may vary depending on your microcontoller and version of CPy.)

```
__future__        digitalio         onewireio         time
__main__          displayio         os                touchio
_asyncio          dualbank          paralleldisplay   traceback
adafruit_bus_device                 errno             ps2io             ulab
adafruit_bus_device.i2c_device      espidf            pulseio           ulab
adafruit_bus_device.spi_device      fontio            pwmio             ulab.fft
adafruit_pixelbuf framebufferio     qrio              ulab.linalg
aesio             frequencyio       rainbowio         ulab.numpy
alarm             gc                random            ulab.scipy
analogio          getpass           re                ulab.scipy.linalg
array             gifio             rgbmatrix         ulab.scipy.optimize
atexit            i2cperipheral     rotaryio          ulab.scipy.signal
```

```
audiobusio          imagecapture       rtc                ulab.scipy.special
audiocore           io                 sdcardio           ulab.utils
audiomixer          ipaddress          select             usb_cdc
binascii            json               sharpdisplay       usb_hid
bitbangio           keypad             socketpool         usb_midi
bitmaptools         math               ssl                uselect
board               mdns               storage            vectorio
builtins            microcontroller    struct             watchdog
busio               micropython        supervisor         wifi
canio               msgpack            synthio            zlib
collections         neopixel           sys
countio             neopixel_write     terminalio
Plus any modules on the filesystem
```

Some of the modules you need to get familiar with are **`board`**, **`microcontroller`**, and **`time`**. These will help you wire up the microcontoller and connect with breakout boards. For more documentation on these to the **`CircuitPython documentation`** page.
If you are going into more depth with CPy it is worth poking around these libraries.
What is surprising is the **`ulab`** library. It is a subset of the python **`numpy`** and **`scipy`** libraries. It include many sophisticated math libraries like **`numpy arrays`**, **`fft`**, minimization, **`where`**, basic matrix operations, **`log`**, **`exp`**, **`mean`**, **`median`**, **`max`**, **`min`**, **`interp`**, trig functions, and more.

So I confidently refer to my pin out figure for the S2Mini and wire these to the thermocouple breakout board.

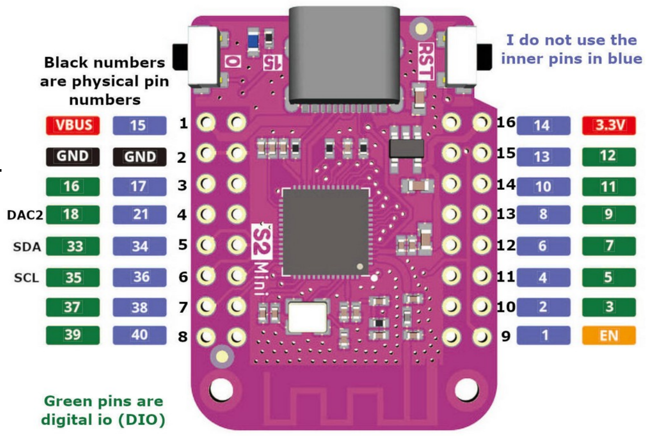

Figure 2: S2Mini Pin outs. I usually do not use the inner rows of pins.

Loading the Adafruit Library

One reason I like using AdaFruit is that they put in a lot of work providing tutorials and libraries for all of their products. I need to find and load the library file on the microcontoller. I go to the CircuitPython Libraries page and download the latest mpy libraries, **`adafruit-circuitpython-bundle-7.x-mpy-20220519.zip`**, and unzip it. It has a huge number of library files, so I only want the ones I need for my breakout board. I look through the lib folder and find **`adafruit_max31855.mpy`** and copy it to my **`code`** folder. Then I look in the examples folder and copy **`max31855_simpletest.py`** to my **`code`** folder.

Plug in the S2Mini and copy **`max31855_simpletest.py`** to the mounted microcontoller **`CIRCUITPY`** drive. Next open **`adafruit_max31855.mpy`** to the **`lib`** folder on **`CIRCUITPY`**. Open **`CIRCUITPY/max31855_simpletest.py`** in *Thonny*. Press the run icon ▶ or type <F5> (on a Mac laptop <fn>+<F5>.)

I am crushed with disappointment by the message

```

  File "<stdin>", line 9, in <module>
AttributeError: 'module' object has no attribute 'SPI'
>>>
```

Well, I look at help(board) and sure enough, there is an **`I2C`** function, but no **`SPI`** function.

Try A Different Microcontoller

I have an AdaFruit ESP32S2 board, so I wire the thermocouple board to it and **`max31855_simpletest.py`** works find spitting out the temperature in both Centigrade and Fahrenheit.

Try to Find or Make SPI()

I spend the next hour or so looking through the library code. The good thing is the python source is available for the libraries. I go back to the CPy Library page and download and unzip the latest mpy libraries, **`adafruit-circuitpython-bundle-7.x-py-20220519.zip`**. I look in **`adafruit_max31855.py`** and see that in the class **`MAX31855`** the **`__init__`** function has the line **`self.spi_device = SPIDevice(spi, cs)`**.

So I need to figure out how to make the **`spi`** object. I search for a file "spi device" and find **`spi_device.py`** in the library **`adafruit_bus_device`**. I see that in class **`SPIDevice`** the first parameter is a **`busio.SPI`** object.

Now the **`busio`** library is not a part of the AdaFruit bundle but it **is** part of CPy core. I go to the CPy Documentation web site and find **`busio`** documentation. I find

```
class busio.SPI(clock: microcontroller.Pin, MOSI: Optional[microcontroller.Pin] =
None, MISO: Optional[microcontroller.Pin] = None, half_duplex: bool = False)
```

This tell me I can create an SPI device by calling busio.SPI(clockPin, MOSIPin, MISOPin). I don't need a MOSI pin so I use None for that argument. Here is the change I make to the code in **`max31855_simpletest.py`**:

```
# If S2Mini make spi from busio
if 'lolin_s2_mini' in board.board_id:
    import busio
    print("S2Mini")
    spi = busio.SPI(board.IO7, None, board.IO9)
else:
    print("Not S2Mini")
    spi = board.SPI()
```

With a sigh of relief, it works!
Note: I tried this later with a different version of CPy and it failed because of **`None`** in the **`SPI`** call. The fix:

```
spi = busio.SPI(board.IO7, board.IO8, board.IO9)
```