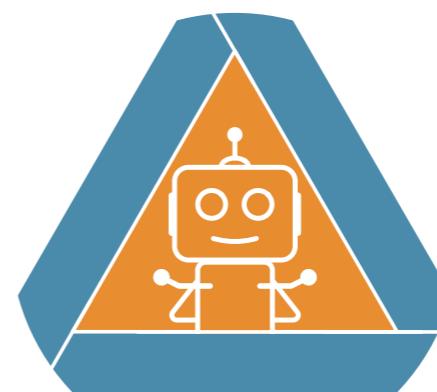


Better Electronics with Jupyter Notebooks

Chris Osterwood
Founder, Capable Robot Components
osterwood@capablerobot.com

<http://capablerobot.com>
<http://github.com/capablerobot>

<http://github.com/osterwood>
[@osterwood](https://twitter.com/osterwood)



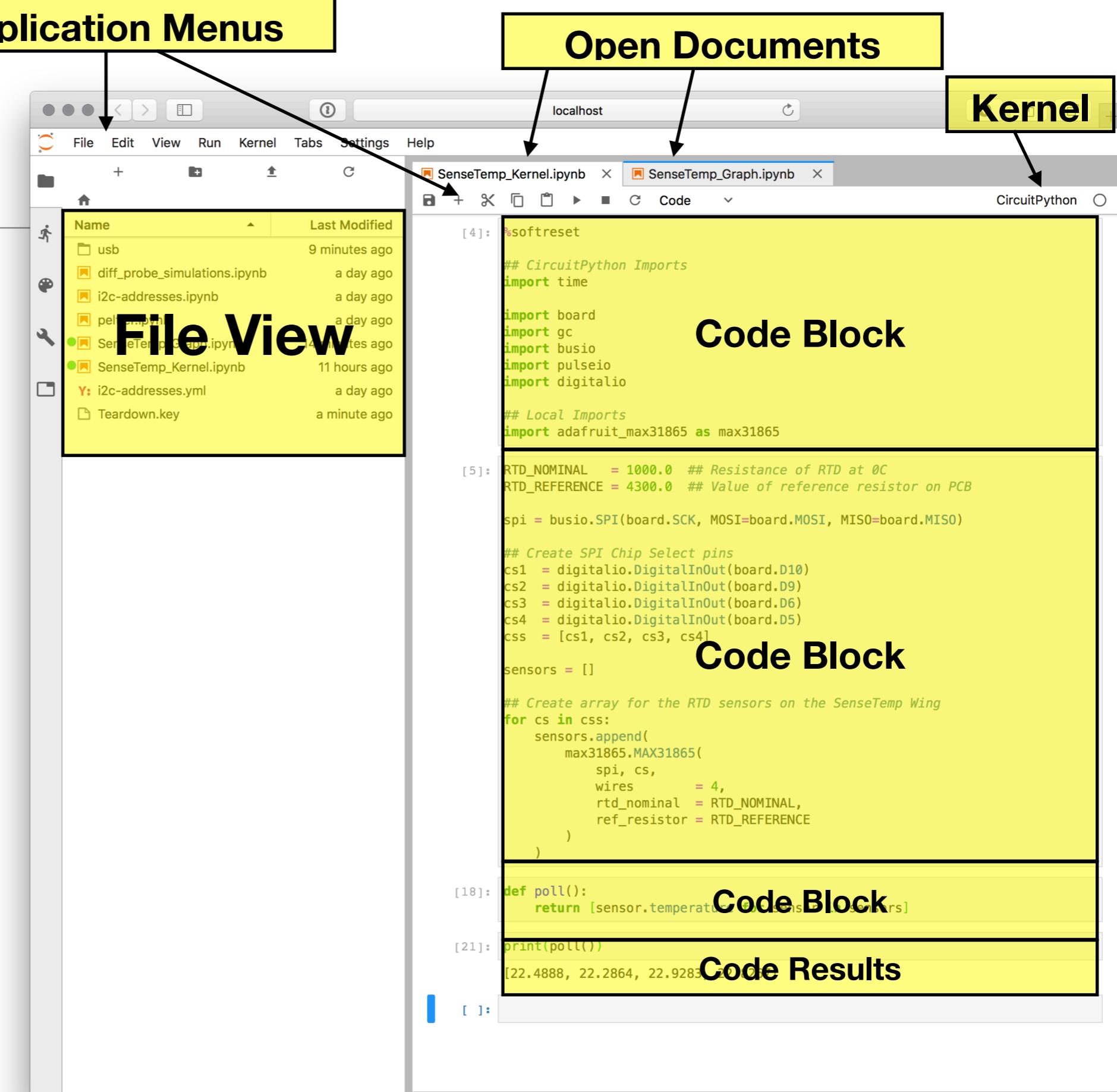
**Capable
Robot
Components**

What are Jupyter Notebooks?

- Web-based development & execution environment
- Support for many different runtimes / languages thru “kernels” which execute code (provided by frontend) in a backend, and which return results to the frontend.
- **Main advantage of “normal” code execution is interleaving of program and results in a single document.**
 - **Results can be live.**
 - **Results can be formatted.**

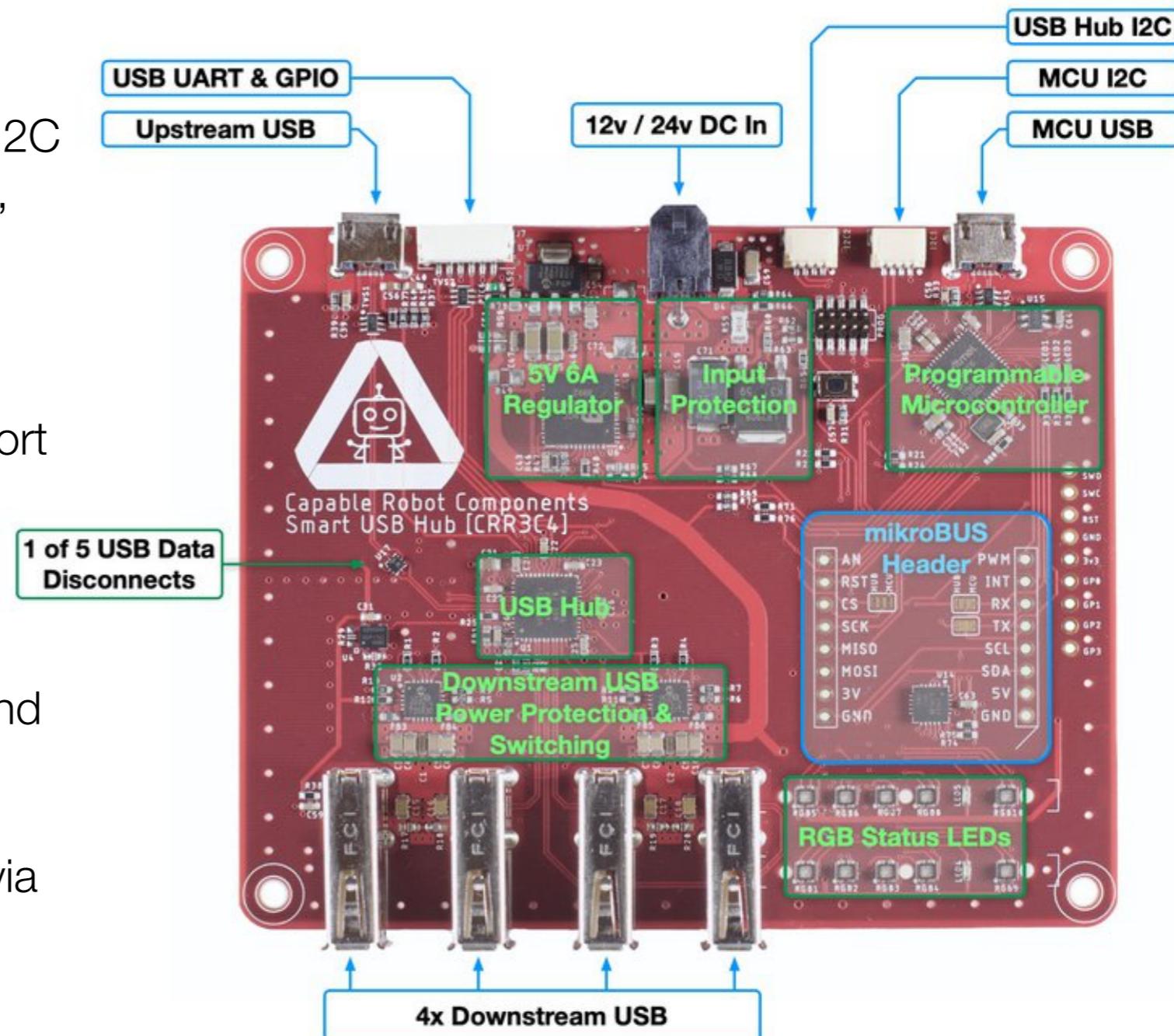
Jupyter Lab

- New generation of Jupyter Notebook
- Still in active development, but very useable and fully featured.
- Not all extensions are supported yet.
- Blocks can be:
 - Code : Executable
 - Markdown : Edit in plain text, render in rich text.
 - Raw : Plain text
- Blocks can be executed in order, or one at a time.



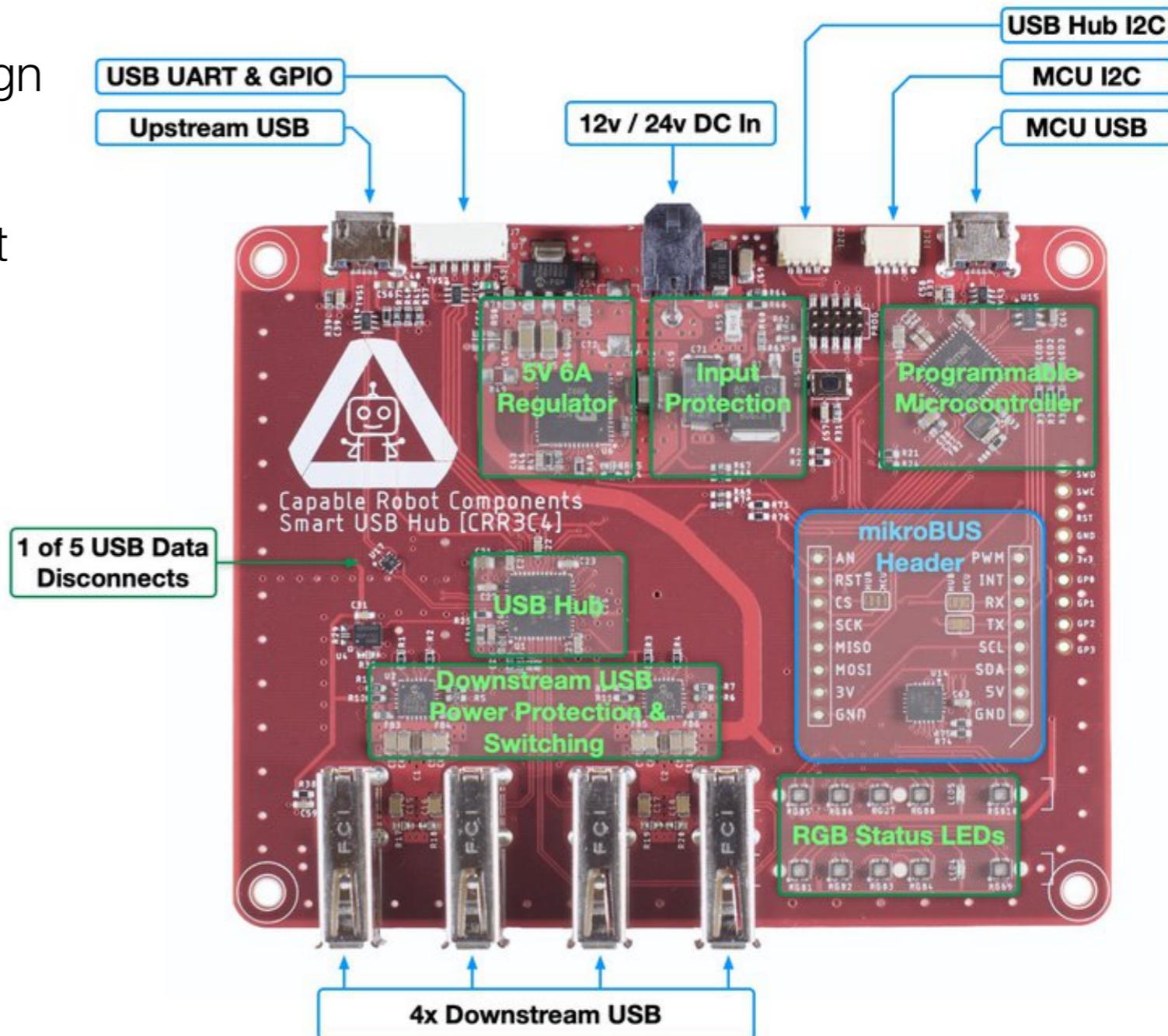
Programmable USB Hub

- Launched on Crowd Supply today!
- 4 Port High Speed USB Hub, that is also:
 - A CircuitPython based development board.
 - A bridge between your computer and I2C (via Sparkfun Qwiic connectors), GPIO, and SPI (via its mikroBUS header).
 - A power supply, providing 6 A of 5 V power to downstream devices and 13 mA resolution monitoring (per-port). Port power is individually limitable and switchable.
 - A USB to TTL Serial adapter.
 - A flexible embedded electronics test and development tool. USB data pairs are individually switchable, allowing you to emulate device removal and insertion via software.



Programmable USB Hub

- Jupyter used for:
 - I2C address assignment / conflict resolution
 - Input protection calculations & design
 - 5V DC regulator design
 - Device testing & driver development



I2C Address Assignment / Conflict Resolution

The screenshot shows a terminal window titled "i2c-address.py" running on a Mac OS X system. The code in the file is a Python script for managing I2C device addresses. It defines a class "Devices" with methods for adding devices, solving conflicts, and printing results. The script then creates a "Devices" object, adds several I2C devices, solves the address conflict, and prints the results.

```
i2c-address.py
41     self.devices = []
42     self.taken = {}
43
44     def add(self, name):
45         if name in self.data:
46             addr = self.data[name]
47             self.devices.append([name, addr, 7 - addr.count(0) - addr.count(1)])
48         else:
49             raise Exception("Cannot find device: {}".format(name))
50
51     def solve(self):
52         ## Iterate through the selected devices by the number of adjustable bits
53         for dev in sorted(self.devices, key=lambda d: d[2]):
54             incr = 0
55
56             while True:
57
58                 try:
59                     bits = resolve(dev[1], incr)
60                 except Exception:
61                     print("WARN : Cannot resolve {}, no free addresses".format(dev[0]))
62                     break
63
64                 address = numeric_addr(bits)
65
66                 if address not in self.taken:
67                     self.taken[address] = dev + [bits]
68                     break
69
70                 incr += 1
71
72             return self.taken
73
74     def print(self):
75         for addr, dev in self.taken.items():
76
77             print("{} @ {}".format(dev[0], hex(addr)))
78             print("\t", dev[1])
79             print("\t", dev[3])
80             print()
81
82     devices = Devices()
83
84     devices.add('TLC59116')
85     devices.add('TLC59116')
86     devices.add('UCS2113-1')
87     devices.add('UCS2113-2')
88     devices.add('MCP23008')
89     devices.add('24AA025E48')
90     devices.add('ATECC508A')
91     devices.add('MCP9808')
92
93     devices.solve()
94     devices.print()
```

[> python3 i2c-address.py

UCS2113-1 @ 0x57
[1, 0, 1, 0, 1, 1, 1]
[1, 0, 1, 0, 1, 1, 1]

UCS2113-2 @ 0x58
[1, 0, 1, 1, 0, 0, 0]
[1, 0, 1, 1, 0, 0, 0]

ATECC508A @ 0x60
[1, 1, 0, 0, 0, 0, 0]
[1, 1, 0, 0, 0, 0, 0]

24AA025E48 @ 0x50
[1, 0, 1, 0, 0, 'A1', 'A0']
[1, 0, 1, 0, 0, 0, 0]

MCP23008 @ 0x20
[0, 1, 0, 0, 'A2', 'A1', 'A0']
[0, 1, 0, 0, 0, 0, 0]

MCP9808 @ 0x18
[0, 0, 1, 1, 'A2', 'A1', 'A0']
[0, 0, 1, 1, 0, 0, 0]

TLC59116 @ 0x61
[1, 1, 0, 'A3', 'A2', 'A1', 'A0']
[1, 1, 0, 0, 0, 0, 1]

TLC59116 @ 0x62
[1, 1, 0, 'A3', 'A2', 'A1', 'A0']
[1, 1, 0, 0, 0, 1, 0]

Line 13, Column 1 2 misspelled words Spaces: 4 Python

I2C Address Assignment / Conflict Resolution

The screenshot shows a Python IDE window with two panes. The left pane displays the source code for `i2c-address.py`, and the right pane shows the command-line output of running the script.

Code (i2c-address.py):

```
41     self.devices = []
42     self.taken = {}
43
44     def add(self, name):
45         if name in self.data:
46             addr = self.data[name]
47             self.devices.append([name, addr, 7 - addr.count(0) - addr.count(1)])
48         else:
49             raise Exception("Cannot find device: {}".format(name))
50
51     def solve(self):
52         ## Iterate through the selected devices by the number of adjustable bits
53         for dev in sorted(self.devices, key=lambda d: d[2]):
54             incr = 0
55
56             while True:
57
58                 try:
59                     bits = resolve(dev[1], incr)
60                 except Exception:
61                     print("WARN : Cannot resolve {}, no free addresses".format(dev[0]))
62                     break
63
64                 address = numeric_addr(bits)
65
66                 if address not in self.taken:
67                     self.taken[address] = dev + [bits]
68                     break
69
70                 incr += 1
71
72             return self.taken
73
74     def print(self):
75         for addr, dev in self.taken.items():
76
77             print("{} @ {}".format(dev[0], hex(addr)))
78             print("\t", dev[1])
79             print("\t", dev[3])
80             print()
81
82     devices = Devices()
83
84     devices.add('TLC59116')
85     devices.add('TLC59116')
86     devices.add('UCS2113-1')
87     devices.add('UCS2113-2')
88     devices.add('MCP23008')
89     devices.add('24AA025E48')
90     devices.add('ATECC508A')
91     devices.add('MCP9808')
92
93     devices.solve()
94     devices.print()
```

Output:

```
python3 i2c-address.py
WARN : Cannot resolve ATECC508A, no free addresses
UCS2113-1 @ 0x57
[1, 0, 1, 0, 1, 1, 1]
[1, 0, 1, 0, 1, 1, 1]

UCS2113-2 @ 0x58
[1, 0, 1, 1, 0, 0, 0]
[1, 0, 1, 1, 0, 0, 0]

ATECC508A @ 0x60
[1, 1, 0, 0, 0, 0, 0]
[1, 1, 0, 0, 0, 0, 0]

24AA025E48 @ 0x50
[1, 0, 1, 0, 0, 'A1', 'A0']
[1, 0, 1, 0, 0, 0, 0]

MCP23008 @ 0x20
[0, 1, 0, 0, 'A2', 'A1', 'A0']
[0, 1, 0, 0, 0, 0, 0]

MCP9808 @ 0x18
[0, 0, 1, 1, 'A2', 'A1', 'A0']
[0, 0, 1, 1, 0, 0, 0]

TLC59116 @ 0x61
[1, 1, 0, 'A3', 'A2', 'A1', 'A0']
[1, 1, 0, 0, 0, 0, 1]

TLC59116 @ 0x62
[1, 1, 0, 'A3', 'A2', 'A1', 'A0']
[1, 1, 0, 0, 0, 1, 0]
```

An orange arrow points from the code editor to the command-line output.

Page-Footer:

Line 13, Column 1 2 misspelled words Spaces: 4 Python

I2C Address Assignment / Conflict Resolution

- Results can now be displayed and formatted richly in an HTML table
- Code which solves I2C address assignment can be collapsed & hidden from view when not being edited.
- Python Mustache library used for HTML templating

The screenshot shows a Jupyter Notebook interface with the following details:

- Title Bar:** localhost, File, Edit, View, Run, Kernel, Tabs, Settings, Help, Launcher, i2c-addresses.ipynb, py_37_env.
- Code Cell:** [3]:

```
devices = Devices()  
  
devices.add('TLC59116')  
devices.add('TLC59116')  
devices.add('UCS2113-1')  
devices.add('UCS2113-2')  
devices.add('MCP23008')  
devices.add('24AA025E48')  
devices.add('ATECC508A')  
devices.add('MCP9808')  
  
devices.solve(html=True)
```

A yellow box highlights the word "Collapsed" next to a three-dot ellipsis icon, with an orange arrow pointing to it.
- Table:** An HTML table showing I2C device assignments:

IDX	Device	Address	Address Struct	Mapping
0	UCS2113-1	0x57	1 0 1 0 1 1 1	
1	UCS2113-2	0x58	1 0 1 1 0 0 0	
2	ATECC508A	0x60	1 1 0 0 0 0 0	
3	24AA025E48	0x50	1 0 1 0 0 A1 A0	A1:0 A0:0
4	MCP23008	0x20	0 1 0 0 A2 A1 A0	A2:0 A1:0 A0:0
5	MCP9808	0x18	0 0 1 1 A2 A1 A0	A2:0 A1:0 A0:0
6	TLC59116	0x61	1 1 0 A3 A2 A1 A0	A3:0 A2:0 A1:0 A0:1
7	TLC59116	0x62	1 1 0 A3 A2 A1 A0	A3:0 A2:0 A1:1 A0:0

I2C Address Assignment / Conflict Resolution

- Results can now be displayed and formatted richly in an HTML table
- Code which solves I2C address assignment can be collapsed & hidden from view when not being edited.
- Python Mustache library used for HTML templating

The screenshot shows a Jupyter Notebook interface with the following details:

- Title Bar:** localhost, i2c-addresses.ipynb, py_37_env
- Toolbar:** File, Edit, View, Run, Kernel, Tabs, Settings, Help
- Code Cell:** [4]:

```
devices = Devices()  
  
devices.add('TLC59116')  
devices.add('TLC59116')  
devices.add('UCS2113-1')  
devices.add('UCS2113-2')  
devices.add('MCP23008')  
devices.add('24AA025E48')  
devices.add('ATECC508A')  
devices.add('ATECC508A')  
devices.add('MCP9808')  
  
devices.solve(html=True)
```
- HTML Output:** A table showing I2C device assignments:

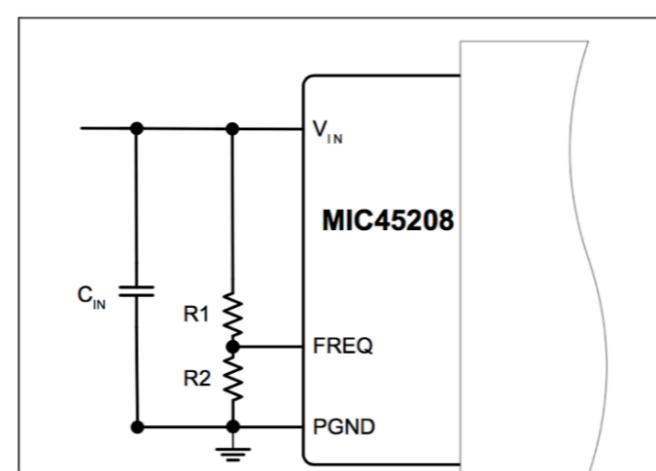
IDX	Device	Address	Address Struct	Mapping
0	UCS2113-1	0x57	1 0 1 0 1 1 1	
1	UCS2113-2	0x58	1 0 1 1 0 0 0	
2	ATECC508A	0x60	1 1 0 0 0 0 0	
3	ATECC508A	NONE	1 1 0 0 0 0 0	
4	24AA025E48	0x50	1 0 1 0 0 A1 A0	A1:0 A0:0
5	MCP23008	0x20	0 1 0 0 A2 A1 A0	A2:0 A1:0 A0:0
6	MCP9808	0x18	0 0 1 1 A2 A1 A0	A2:0 A1:0 A0:0
7	TLC59116	0x61	1 1 0 A3 A2 A1 A0	A3:0 A2:0 A1:0 A0:1
8	TLC59116	0x62	1 1 0 A3 A2 A1 A0	A3:0 A2:0 A1:1 A0:0

DC Regulator Design

Important excerpts from the data sheet can be embedded in the notebook via image tags inside of Markdown Blocks

MIC45208 Device Properties

Switching Frequency



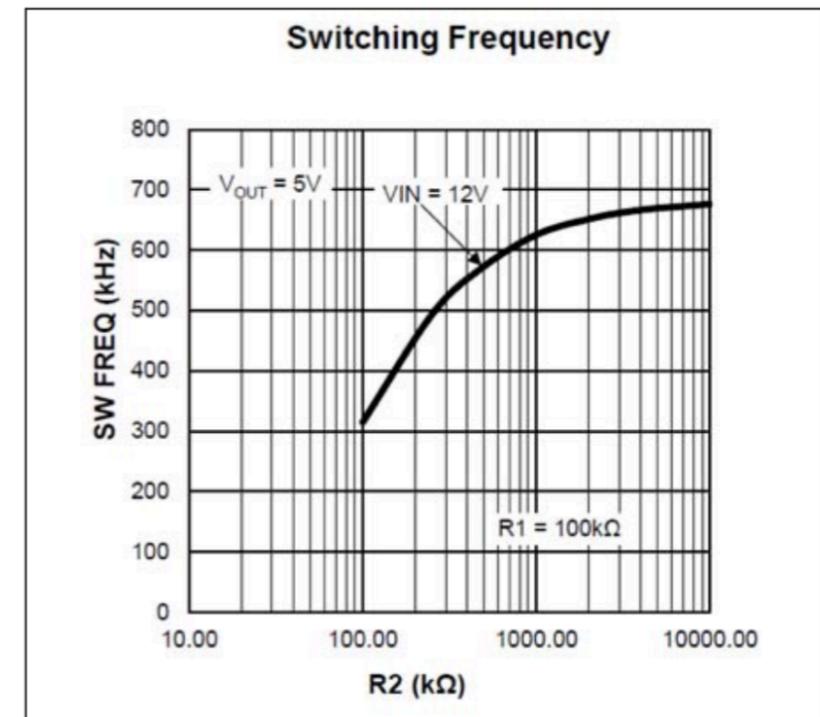
$f_{SW} = f_O \times \frac{R2}{R1 + R2}$

Where:

$f_O = 600 \text{ kHz}$ (typical per [Table 1-1](#))

$R1 = 100 \text{ k}\Omega$ is recommended

$R2$ = Needs to be selected in order to set the required switching frequency



```
[3]: ## Calculate switching frequency from Rup & Rdn
if Rdn == False:
    fsw = 600 * ureg.kHz
else:
    fsw = 600 * ureg.kHz * Rdn / (Rup + Rdn)

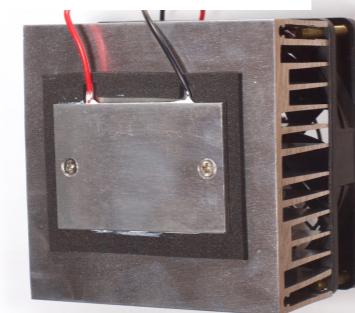
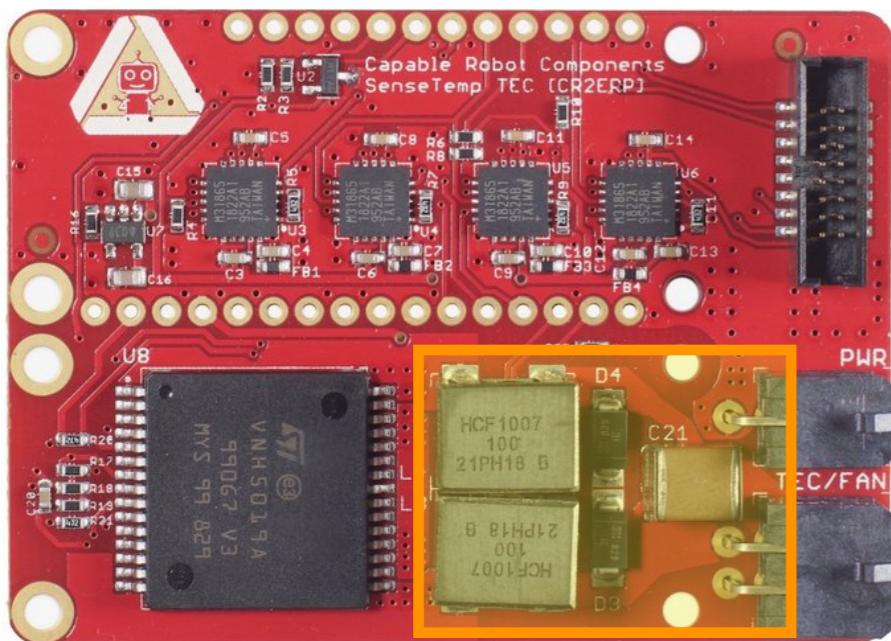
## Device Properties (from datasheet)
L      = 1.0 * ureg.uH      # Inductor is integrated into the package
Rdson = 16 * ureg.mΩ       # On-resistance of low-side power MOSFET
Icl   = 70 * ureg.uA        # Current-limit source current
Vcloffset = 14 * ureg.mV   # Current-limit threshold
Toffmin = 200 * ureg.ns    # Minimum off time
```

Live Graphing Demos

- USB Hub Current Monitor
- SenseTemp

Spice Simulation TEC Controller

- DC motor driver (PWM based)
- Non-inductive load : Peltier Element
- What inductors, diodes, & capacitors are needed to make the motor driver happy?



```
import math
import numpy as np
import matplotlib.pyplot as plt

from PySpice.Unit import *

from PySpice.Probe.Plot import plot
from PySpice.Spice.Netlist import Circuit
from PySpice.Spice.Library import SpiceLibrary
from PySpice.Doc.ExampleTools import find_libraries

spice_library = SpiceLibrary('/Users/chris/Dropbox/Books/spicelib/')

circuit = Circuit('TEC Model')
circuit.include(spice_library['SMA6J12A'])

# 100 us -> 10 kHz
# 50 us -> 20 kHz
source = circuit.PulseVoltageSource('input', 'pwm', circuit.gnd,
                                      initial_value=0@u_V, pulsed_value=12@u_V,
                                      pulse_width=1@u_us, period=50@u_us)

## TEC is 12V, 5A -> 2.4 ohm
r_tec = 2.4@u_Ω
inductor = 10@u_uH

Rtec = circuit.R('TEC', 'TECP', 'TECM', r_tec)
L1 = circuit.L(1, 'pwm', 'TECP', inductor)
L2 = circuit.L(2, circuit.gnd, 'TECM', inductor)

C5 = circuit.C(5, 'TECP', 'TECM', 2*47@u_uF)

circuit.X('D1', 'SMA6J12A', 'TECP', 'snub')
circuit.X('D2', 'SMA6J12A', 'snub', 'TECM')

Rtec.minus.add_current_probe(circuit)

simulator = circuit.simulator(temperature=25, nominal_temperature=25)
step_time = source.period / 100
analysis = simulator.transient(step_time=step_time, end_time=source.period*10)

figure = plt.figure(1, (20, 10))

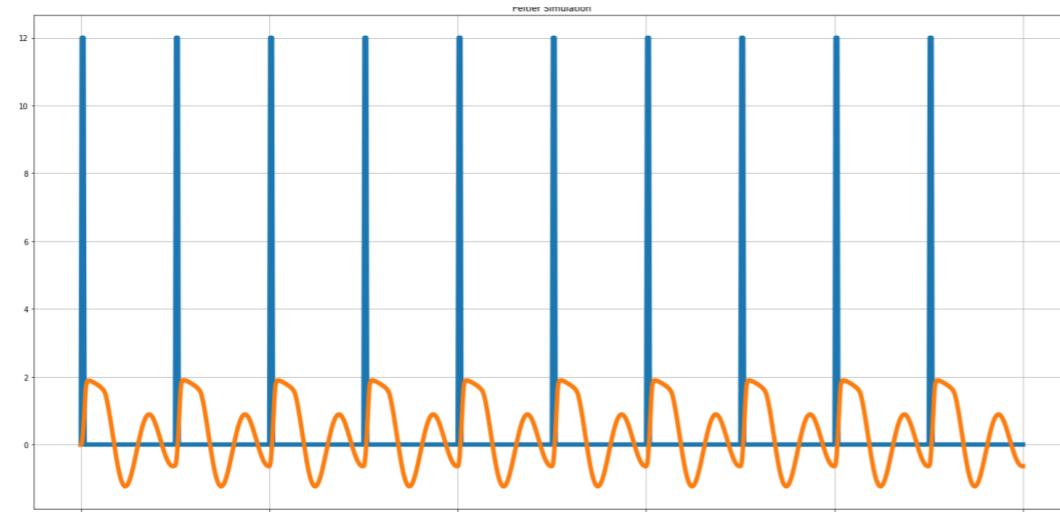
plot(analysis['pwm'])
plot(analysis['TECP'] - analysis['TECM'])

plt.title("Peltier Simulation")
plt.grid()
plt.legend(('VIN', 'VTEC'), loc=(.8,.8))
plt.tight_layout()
plt.show()
```

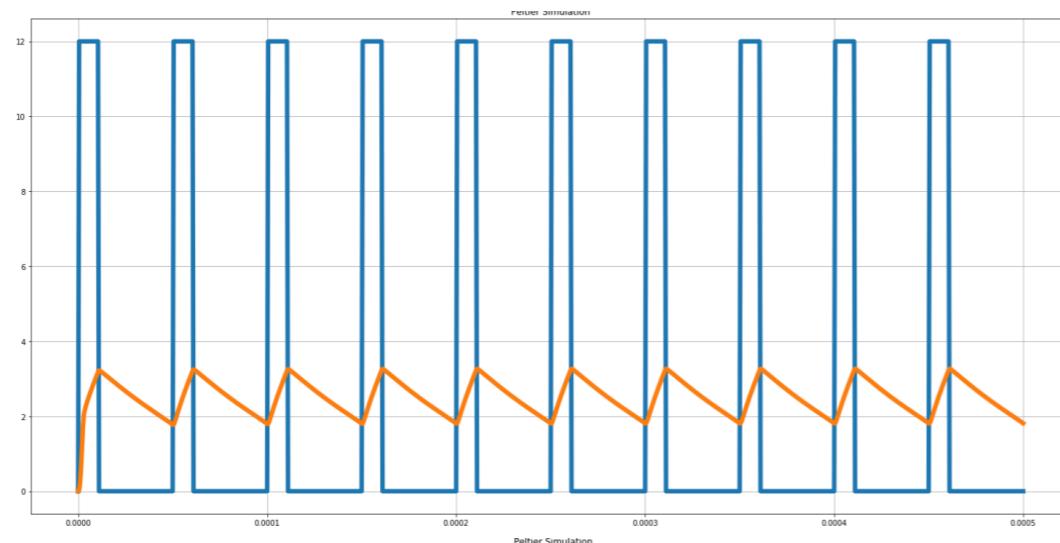
Spice Simulation TEC Controller

1 uH & 8 uF

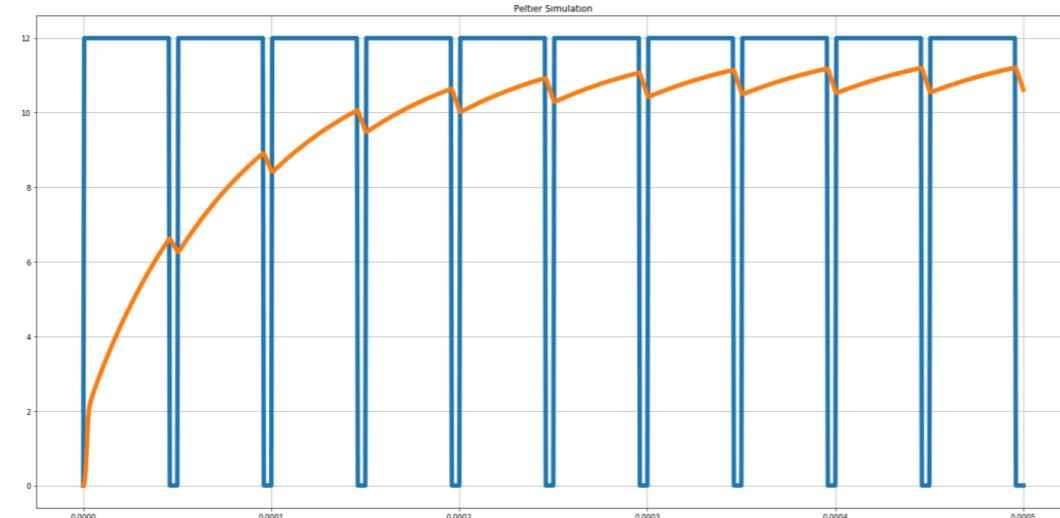
2%



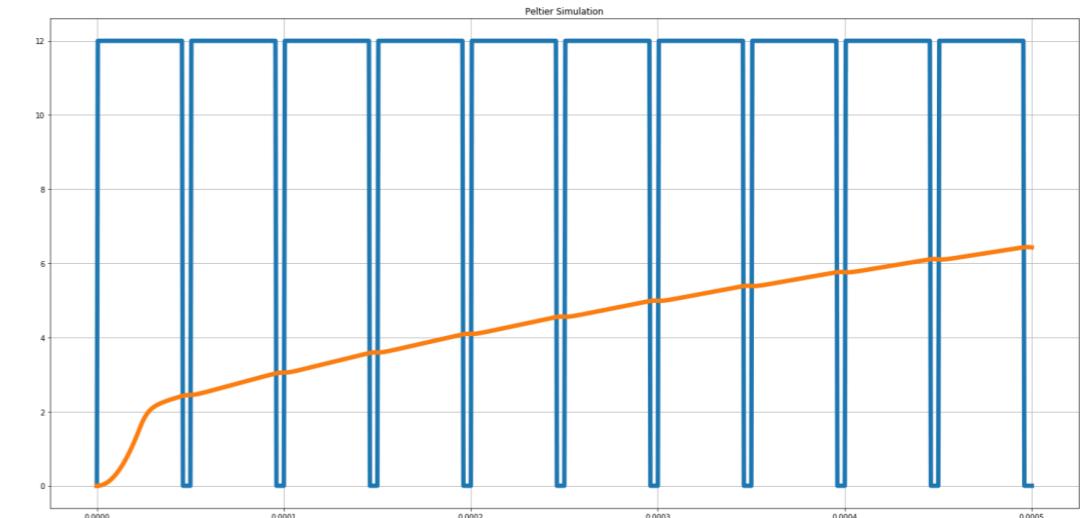
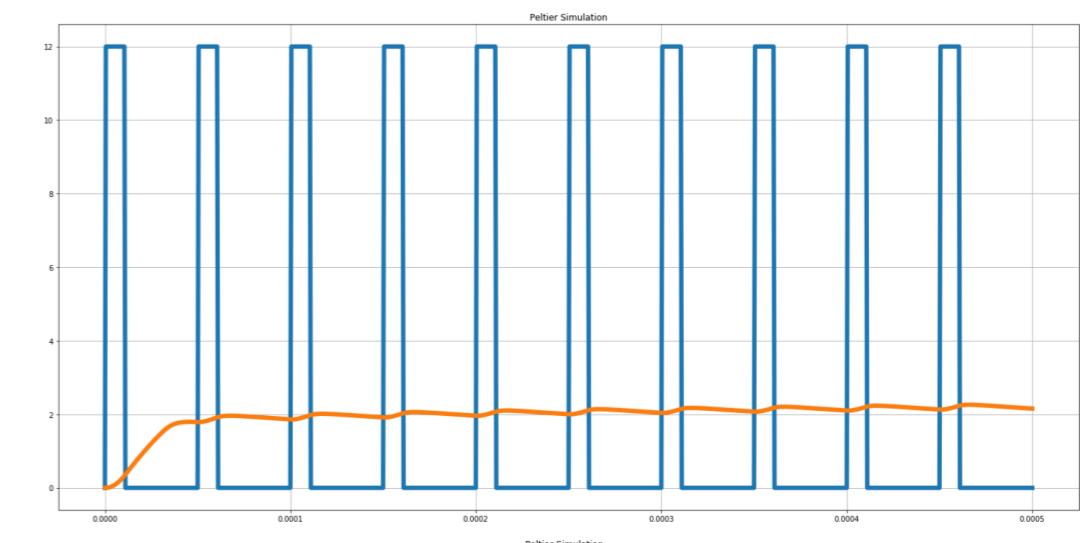
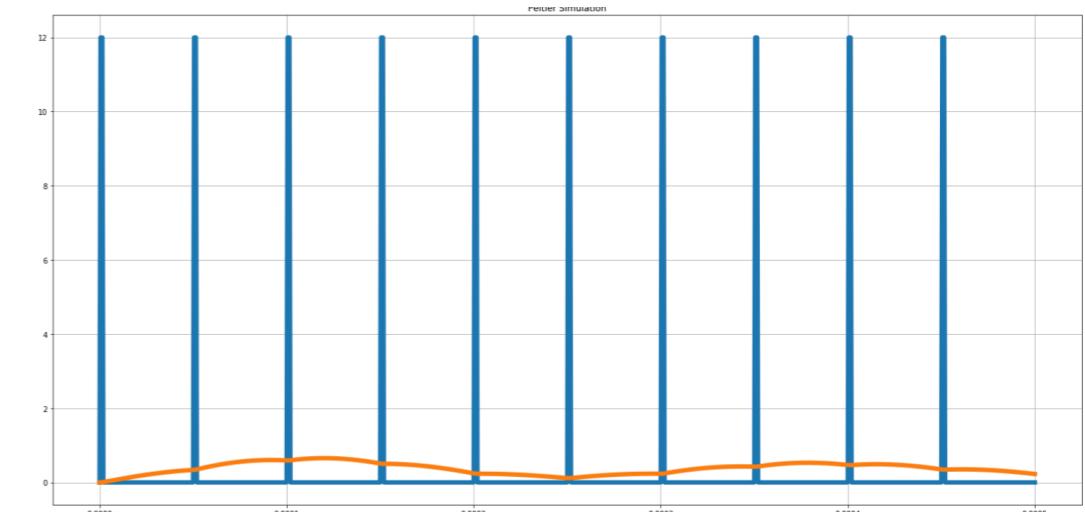
20%



95%

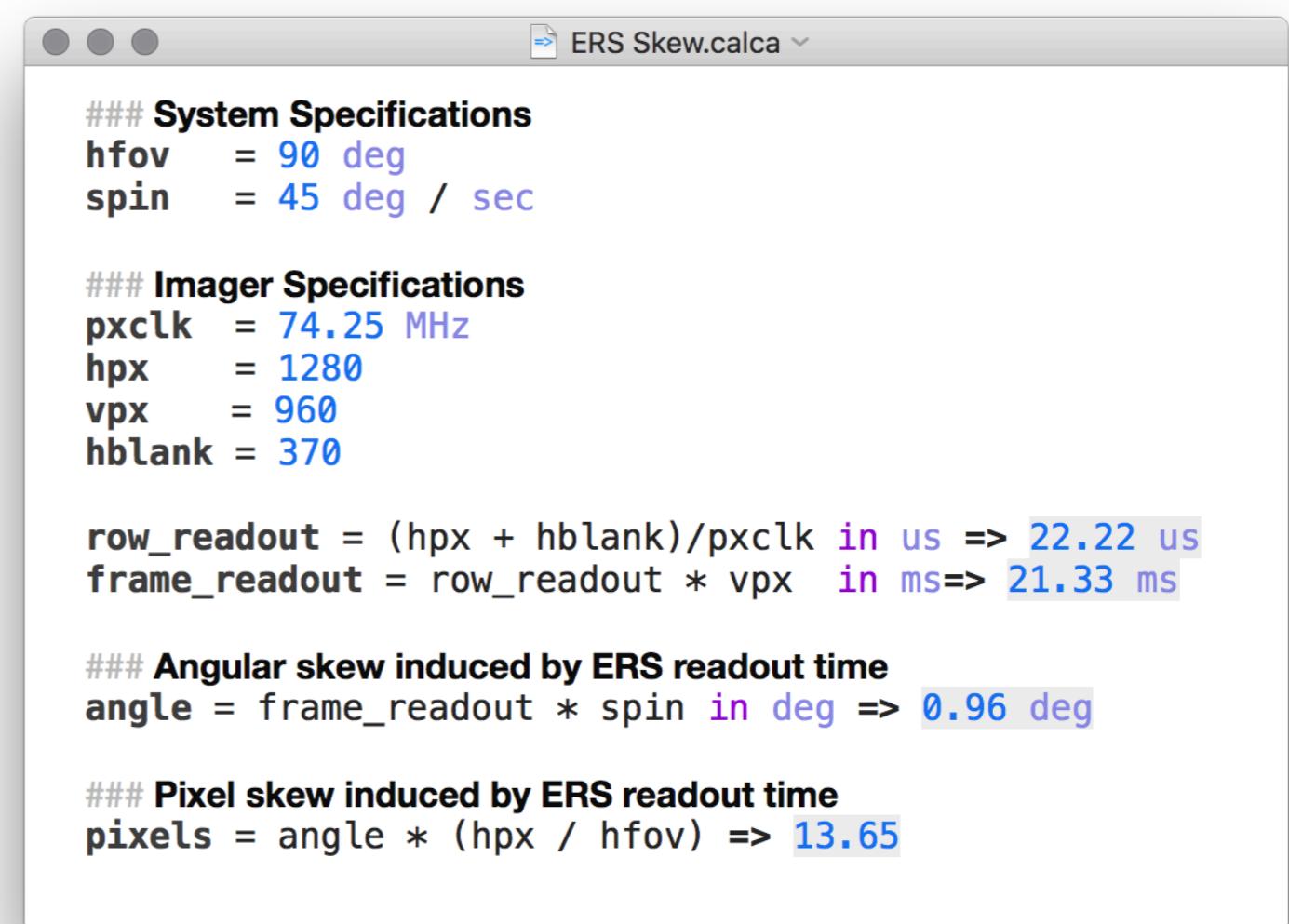


10 uH & 94 uF



Additional Resources & Tools

- <http://github.com/capablerobot/notebooks>
- <https://jupyter.org>
- <https://jupyterlab.readthedocs.io>
- <https://pint.readthedocs.io>
- <https://bqplot.readthedocs.io>
- <https://nbconvert.readthedocs.io>
- <https://cocalc.com>
- <https://github.com/olbrich/ruby-units>



```
### System Specifications
hfov = 90 deg
spin = 45 deg / sec

### Imager Specifications
pxclk = 74.25 MHz
hpx = 1280
vpx = 960
hblank = 370

row_readout = (hpx + hblank)/pxclk in us => 22.22 us
frame_readout = row_readout * vpx in ms => 21.33 ms

### Angular skew induced by ERS readout time
angle = frame_readout * spin in deg => 0.96 deg

### Pixel skew induced by ERS readout time
pixels = angle * (hpx / hfov) => 13.65
```