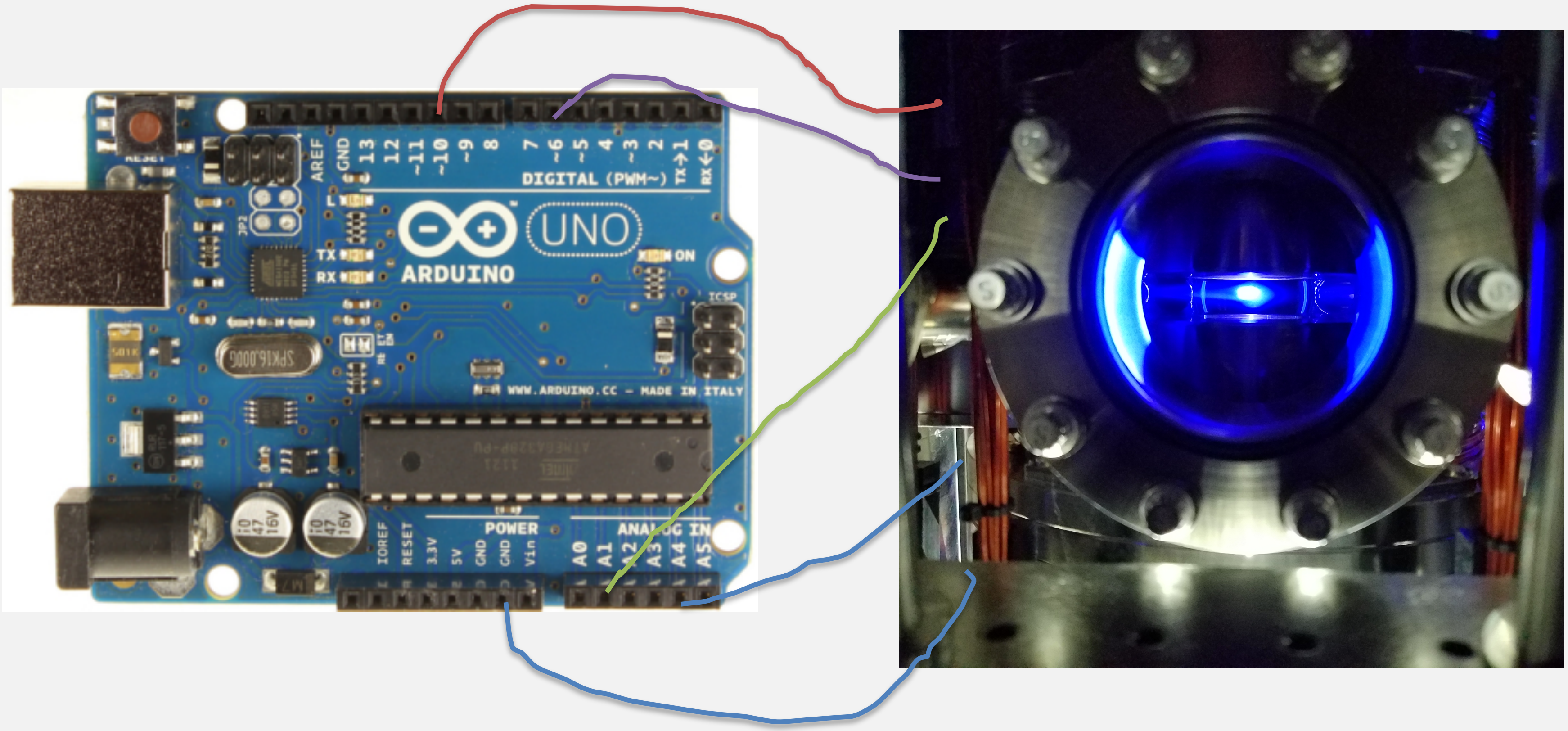
Physics 124: Lecture 1



Course Structure

Crash Course for Arduino

Crash Course in C

adapted from T. Murphy's slides

Course Structure

- MWF Lecture->MW?? at least for first 5 weeks
  - 4% of course grade on participation/attendance (down from 7%)
- Structured Labs first 4 weeks (building blocks)
  - demonstrated performance is 36% of grade (9% each)
  - must adhere to due dates to prevent falling behind
- Midterm to demonstrate simple coding, 10% of grade
- Creative project second half of quarter, 50% of grade!
  - final demonstration Friday March 24 with spectators
- Work in teams of 2
- Primary Lab periods: M/T 2−6
  - at least 2/3 of "help" will be on hand
  - will have access to lab space 24/7
- 2 TAs:
  - Darius Choksy and Rudy Pei, extensive research experience

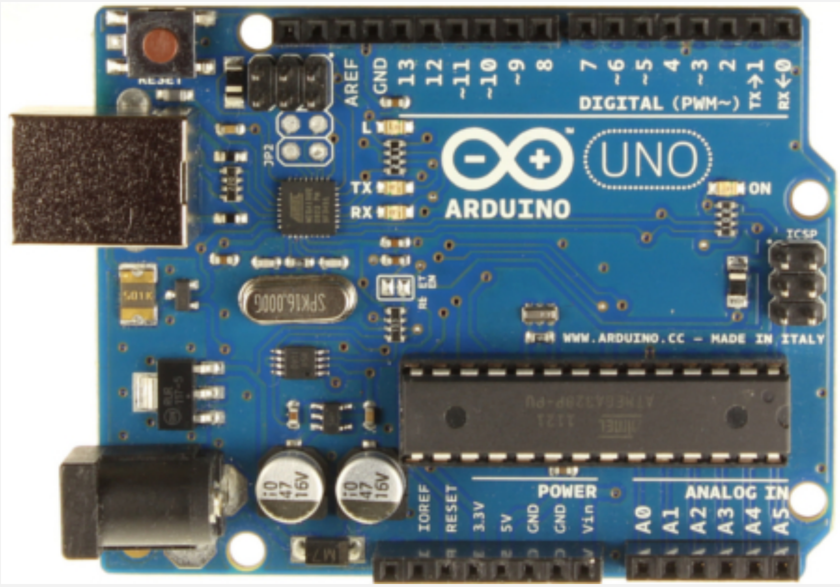
Project Rubric

- Three principal ingredients (a feedback loop)
  - Measure/Sense/Perceive
    - the most physics-related component
  - Process/Calculate/Think
    - usually via microcontroller
  - Act/React/Do
    - motors, lights, sound, display
- Examples from past (inadequately small sample)
  - robotic hand moving as real hand via Kinect
  - control type car parallel parks itself
  - automatic shifting on bike
  - rotating LED sphere changes color/intensity to music
  - see for more

Why is this a Physics Course?

- What about this is physics? Why do we bother?
- True that this is not front/center in physics research
- BUT…
  - has been useful in research (mine and former advisors)
  - learn about sensors
  - proficiency with a tool that can help control experiments
  - learn some coding in C (well-used language in physics)
  - more familiar with practical electronics
  - learn team dynamics/communication
  - deadlines
  - gain confidence in ability to do something unique
- Goal is fun enough to motivate real investment
  - a necessary ingredient to *real* learning

Arduino: This is our Brain in Phys124



Arduino Uno

Arduino Nano

- http://arduino.cc
- Packaged Microcontroller (ATMega 328)
  - lots of varieties; we'll primarily use Uno and Nano
  - USB interface; breakout to pins for easy connections
  - Cross-platform, Java-based IDE, C-based language
  - Provides higher-level interface to guts of device

Arduino Core Capabilities

- Arduino makes it easy to:
  - have digital input/output (I/O) (14 channels on Uno)
  - analog input (6 channels on Uno; 8 on Nano)
  - “analog” (PWM) output (6 of the digital channels)
  - communicate data via serial (over USB makes easy)
- Libraries available for:
  - motor control; LCD display; ethernet; SPI; serial; SD cards, and lots more
- “Shields” for hardware augmentation
  - stepper motor drivers
  - LCD display
  - GPS receiver
  - bluetooth, SD card, ethernet, wireless, and lots more

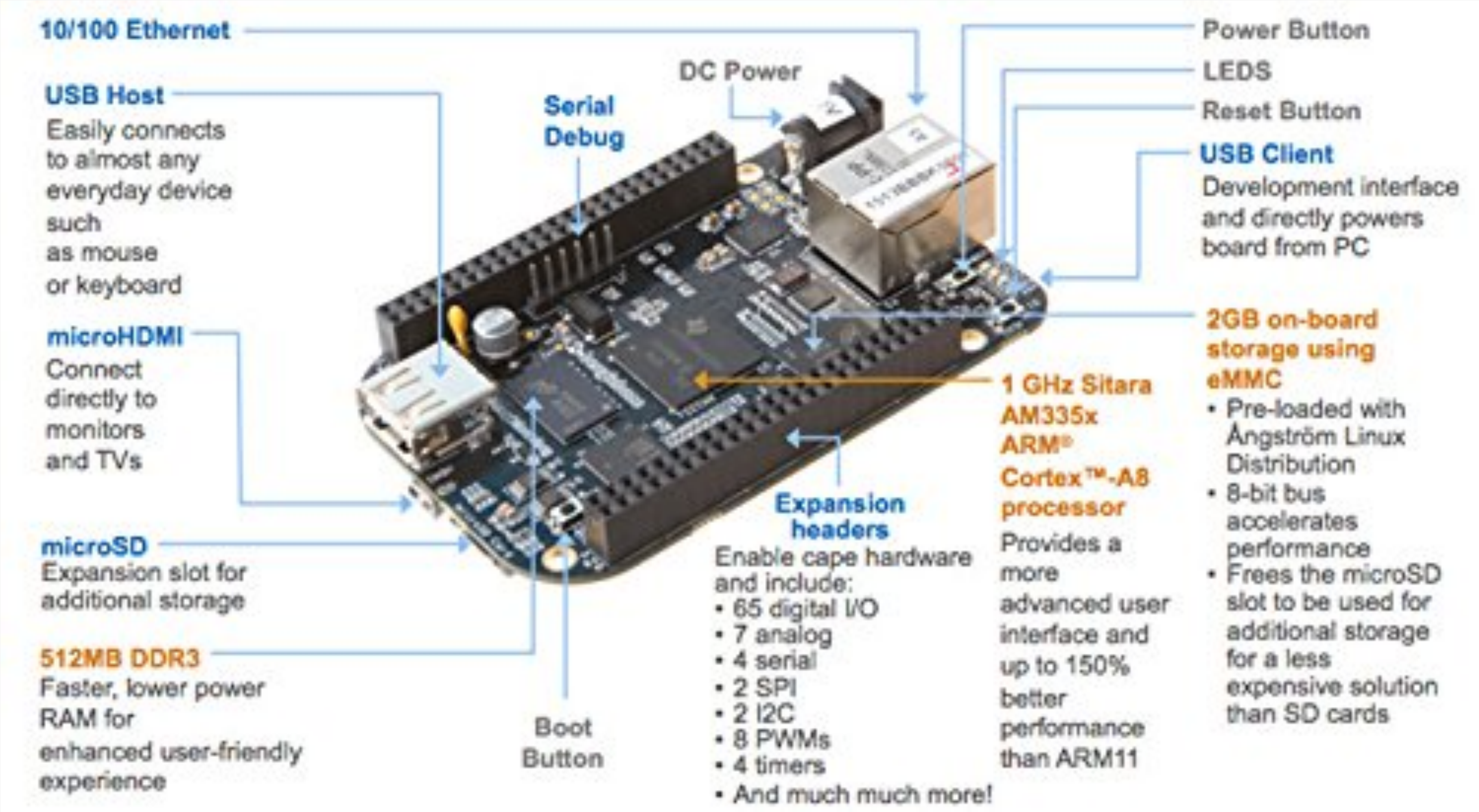
Why Arduino?

- Popular in labs 2005-2012, has key elements
- Arduino is for all platforms Mac/Linux/Windows
- Arduino is cheap (<=$16 vs RPi3 $40, BBB $55, FPGA $150)
  - so students can afford to play on their own (encouraged!)
- Arduino programming usefully transfers to research
  - C
- Intermediate high-level functions mean less time at register/bit level
  - more time to learn about sensors, put amazing projects together, rather than dwell on computer engineering
- low-level understanding is useful

What’s popular in university labs nowadays?

- Since 2013, Beaglebone Black kicked-off in many leading (AMO) labs (Raspberry Pi is also popular)
- Embedded computers
- Advantages:
  - Higher level programming: Python
- Disadvantages:
  - Steeper learning curve: networking, unix, python
  - Programmable Real-time Unit uses C, but assembly code is best
- However, Beaglebone Blue for education led by UCSD (ECE) is now available!
  - https://beagleboard.org/blue
- Since 2015, undergrads in leading AMO labs are programming FPGAs, but they *already knew* Arduino.

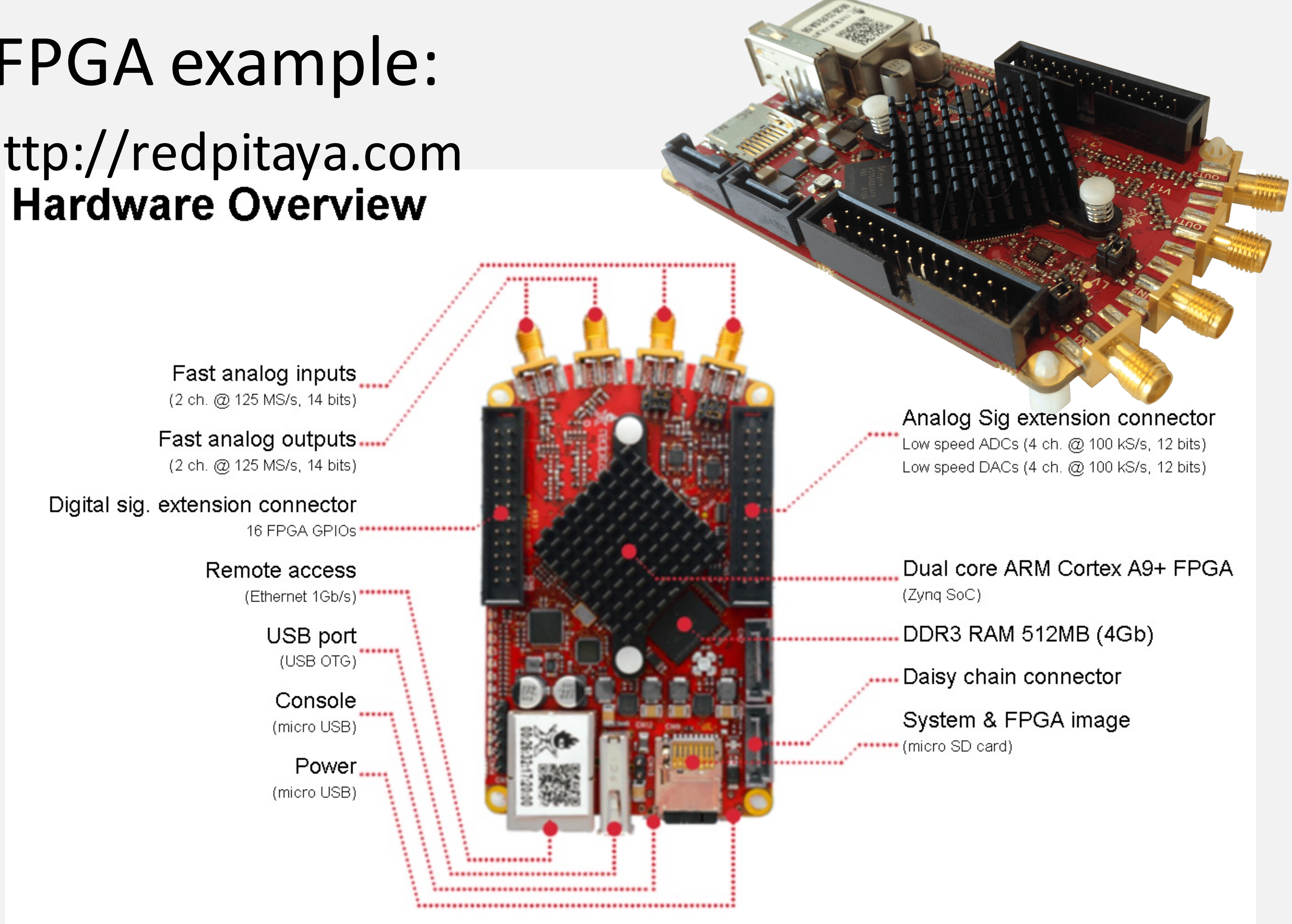
Beaglebone Black (http://beagleboard.com/black)



No need for ethernet shield, SD shield, display shield, PRU runs at 200MHz, wireless instead of eth version available

FPGA example:

http://redpitaya.com

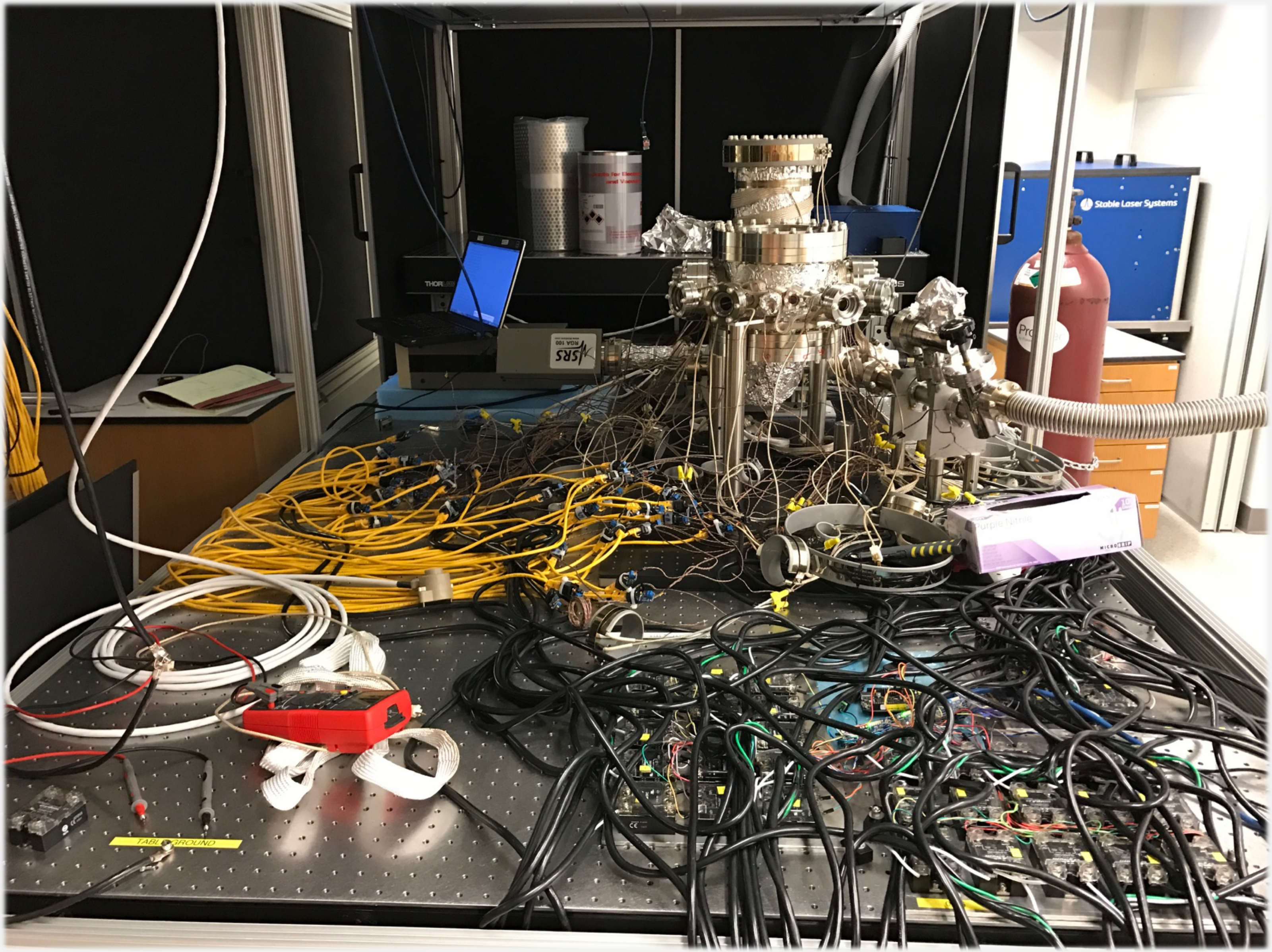


Best for fast signal processing, it can become an oscilloscope, spectrum analyzer, fast feedback control, LCR meter, anything! Programming: VHDL, labview, matlab, python

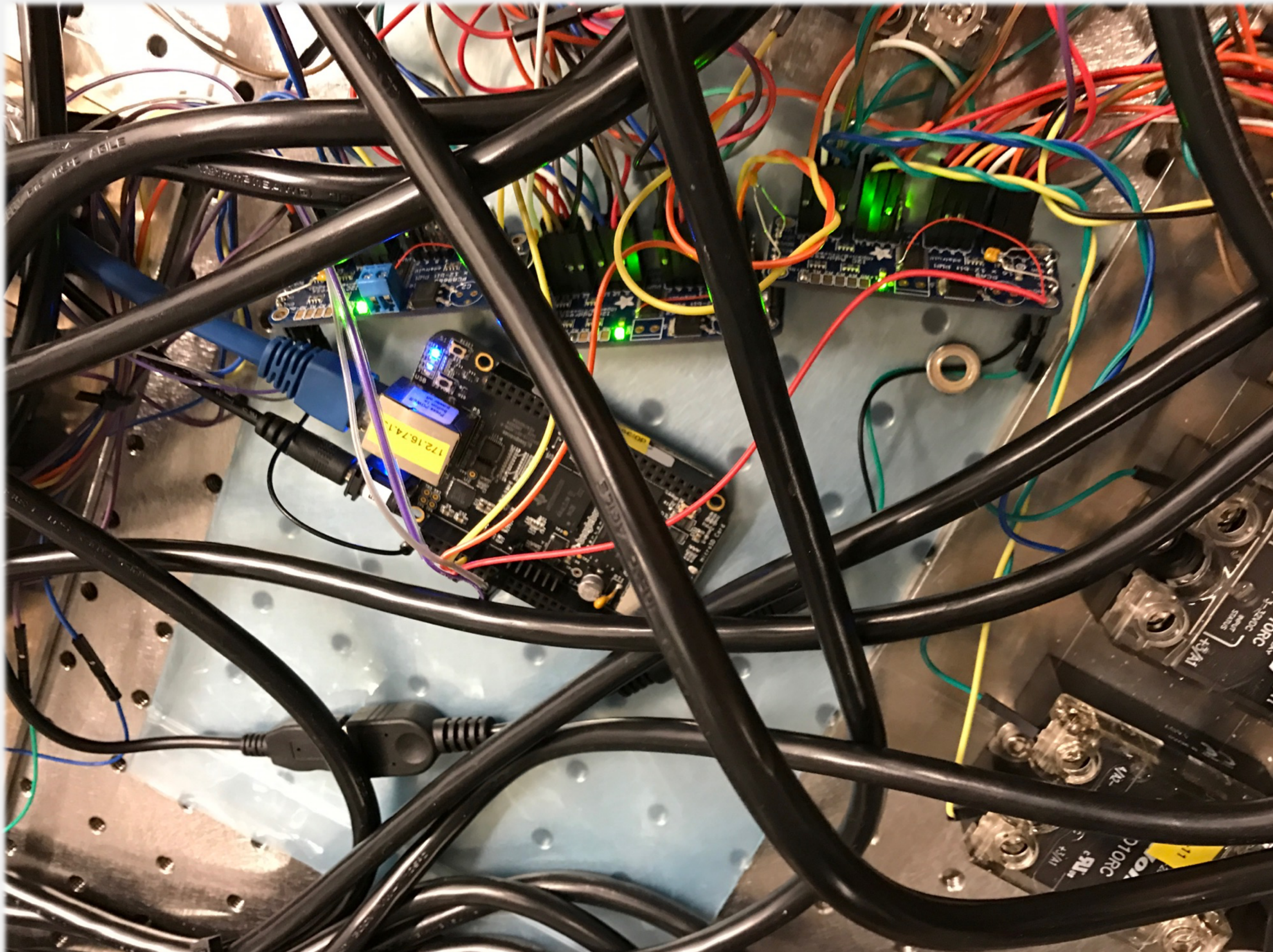
A few examples from my lab

- Lab temp monitor (undergraduate project)
  - I2C over ethernet cable, tested to 20 ft! eight devices.
  - http://lab1.barreiro.ucsd.edu http://lab2.barreiro.ucsd.edu

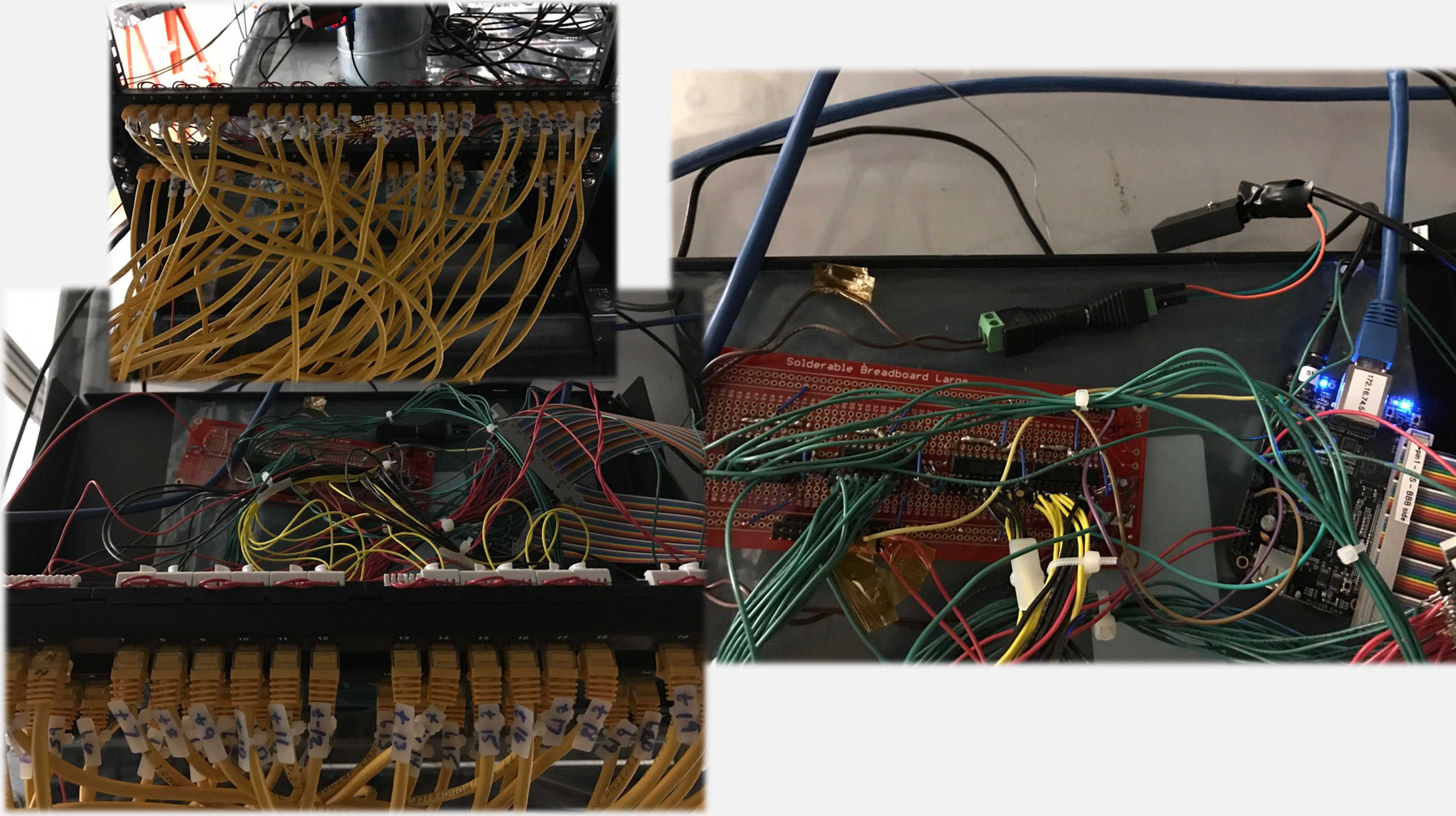
Ultrahigh Vacuum bake: heating and monitoring



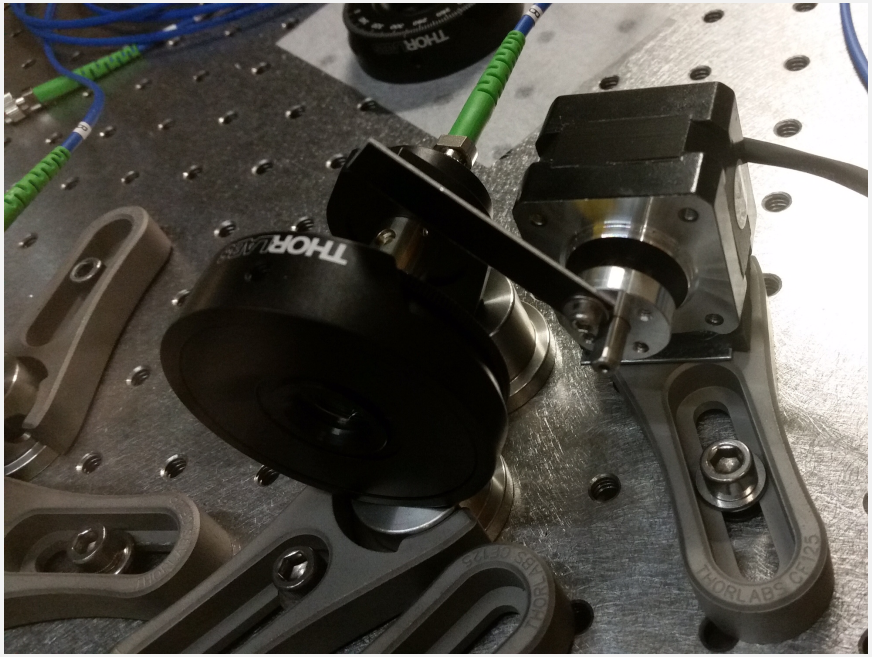
Heating with Solid State Relays, ~40

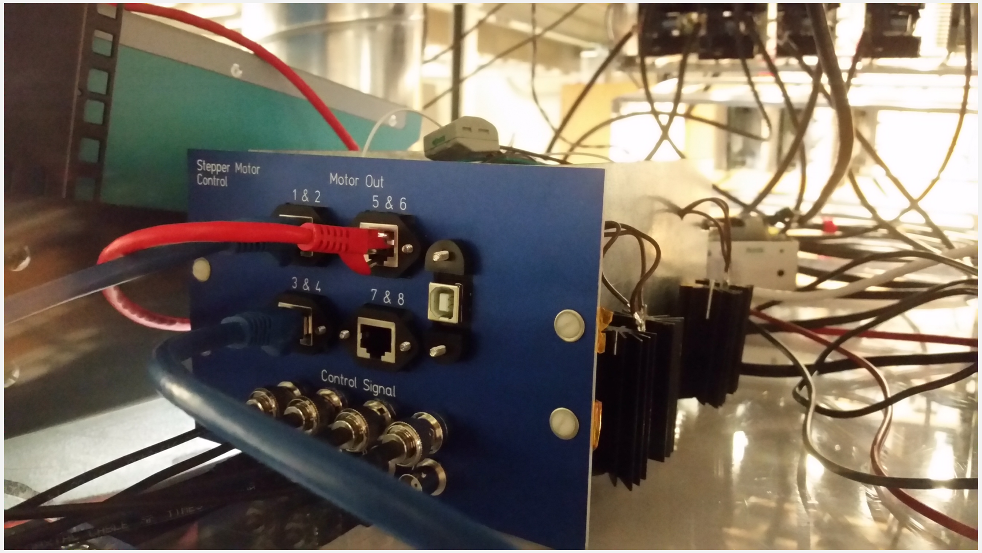


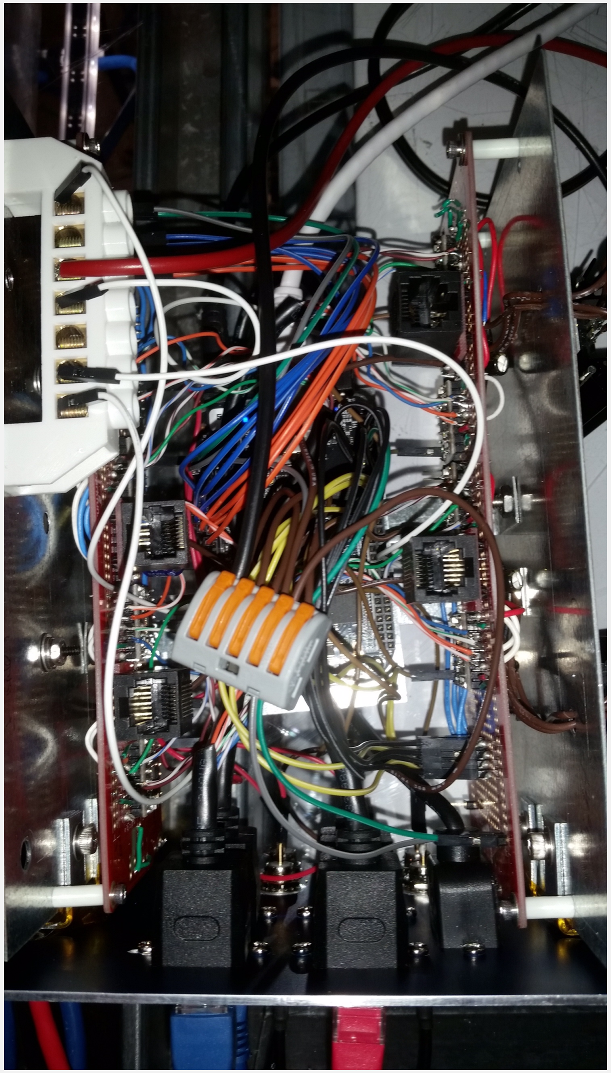
Monitoring with thermocouples (48!)



Stepper motors as laser shutters

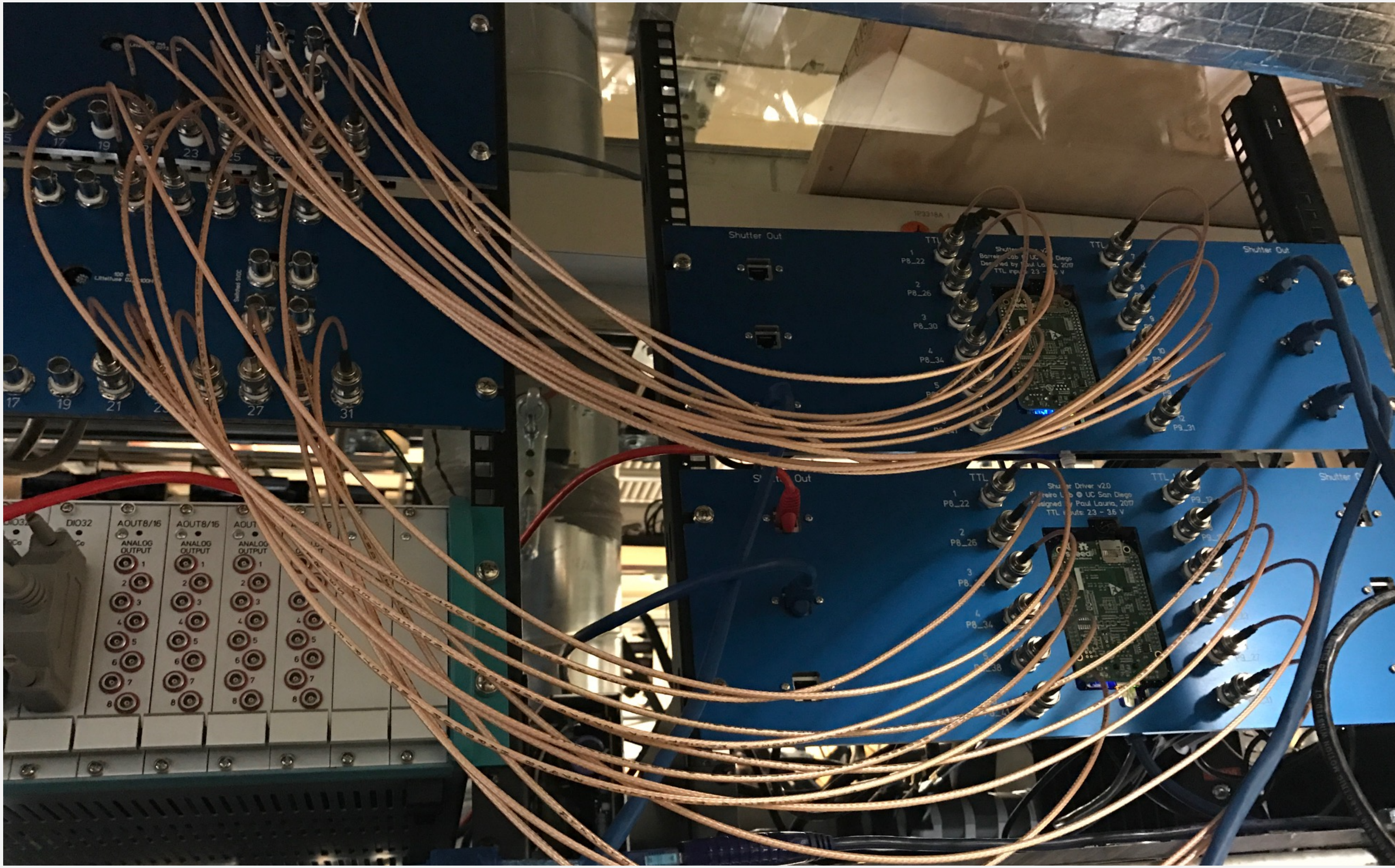






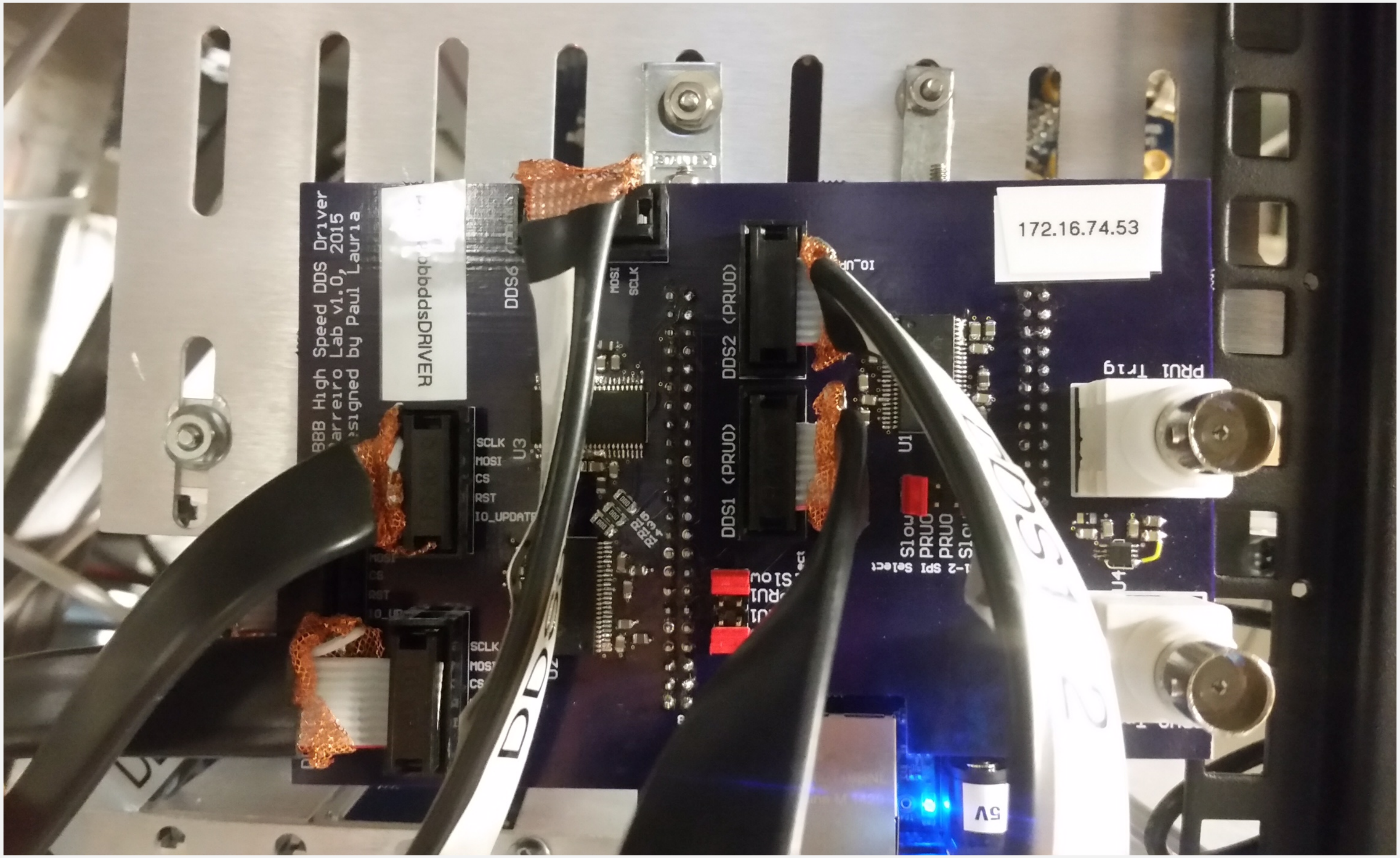
v2 under construction, PCB,
12 motors/board/bbb

Our laser shutter control, pro version

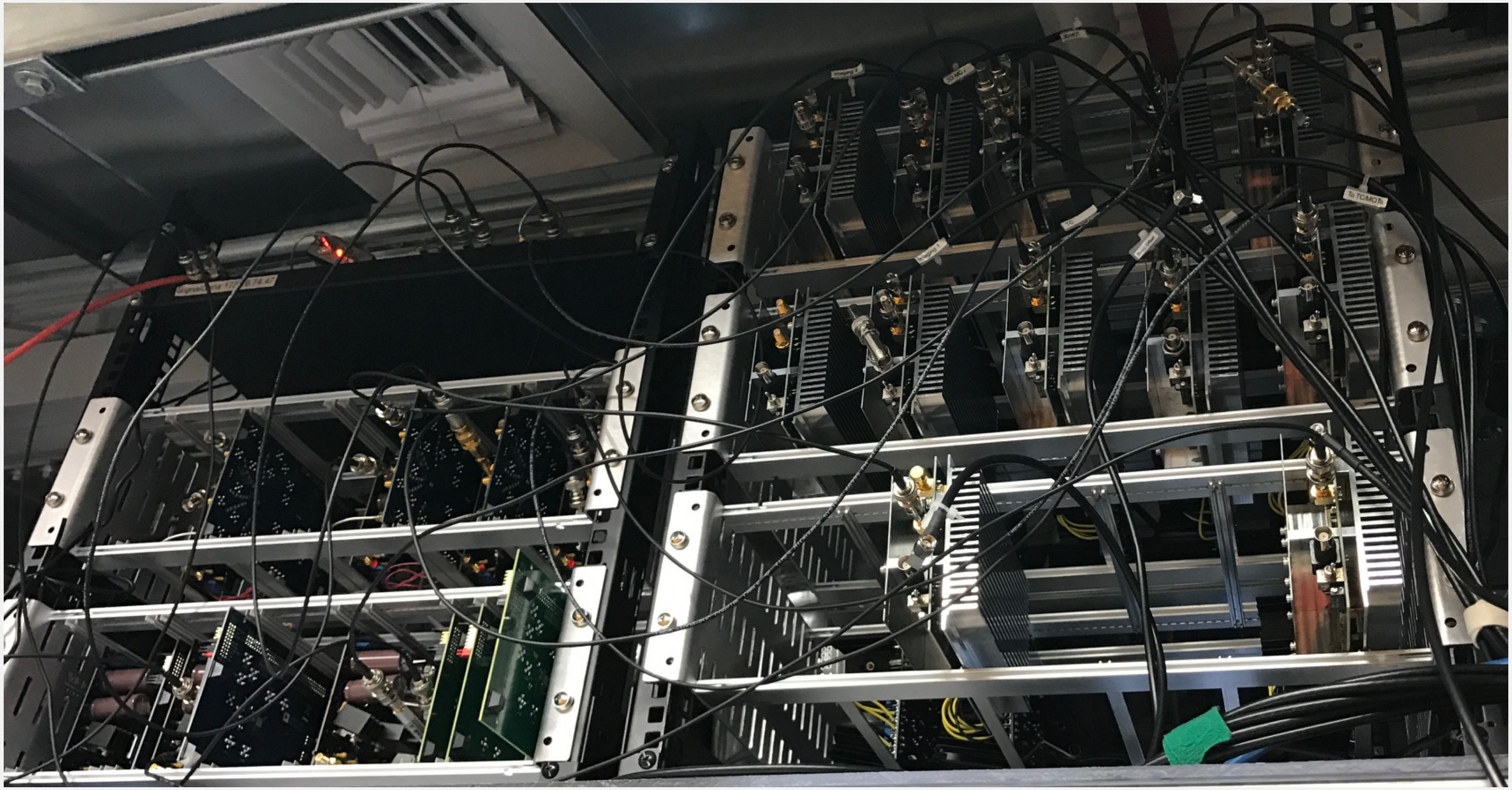


Direct Digital Synthesis chip programming

- A single BBB receives programming instructions from ethernet for 4 DDS (undergrad project!).

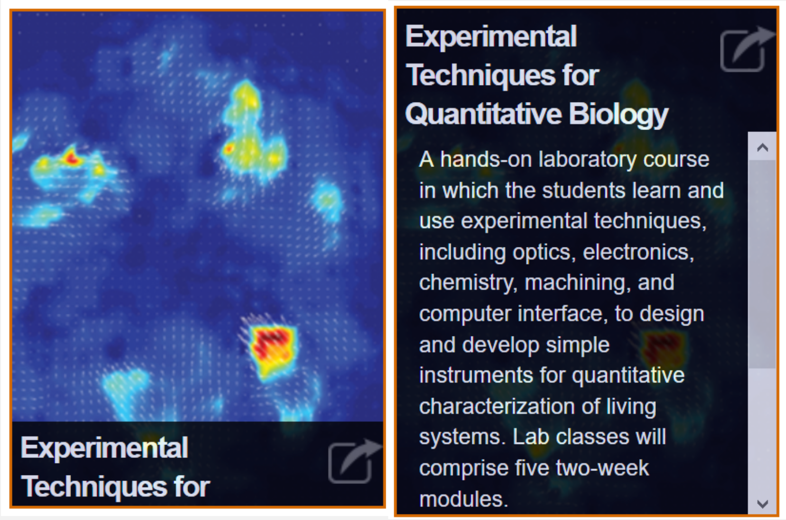


BBBs controlling DDS array, with RF amplifiers



Examples elsewhere on campus

- PHYS 270A in Graduate Qbio program
  - Experimental Techniques for Quantitative Biology
  - a lab projects class using Arduino for 2 weeks
  - examples: control temperature, illuminate and move microscope stage
  - http://qbio.ucsd.edu/courses.php
  - Search UCSD news article on the Hacker lab



IoT

Altera.com

- Internetworking of physical devices
- Applications categories:

Information and analysis			Automation and control		
1 Tracking behavior	**2 Enhanced situational awareness**	**3 Sensor-driven decision analytics**	**1 Process optimization**	**2 Optimized resource consumption**	**3 Complex autonomous systems**
Monitoring the behavior of persons, things, or data through space and time.	Achieving real-time awareness of physical environment.	Assisting human decision making through deep analysis and data visualization	Automated control of closed (self-contained) systems	Control of consumption to optimize resource use across network	Automated control in open environments with great uncertainty
Examples: Presence-based advertising and payments based on locations of consumers Inventory and supply chain monitoring and management	*Example:* Sniper detection using direction of sound to locate shooters	*Examples:* Oil field site planning with 3D visualization and simulation Continuous monitoring of chronic diseases to help doctors determine best treatments	*Examples:* Maximization of lime kiln throughput via wireless sensors Continuous, precise adjustments in manufacturing lines	*Examples:* Smart meters and energy grids that match loads and generation capacity in order to lower costs Data-center management to optimize energy, storage, and processor utilization	*Examples:* Collision avoidance systems to sense objects and automatically apply brake Clean up of hazardous materials through the use of swarms of robots

- All common to academic and industrial research, e.g., measurement and control of complex experiments.

Economic impact

The economic impact of the Internet of Things will be greater in advanced economies.

Share of economic impact, 2025, %

Advanced economies | Developing economies

Settings	Advanced economies	Developing economies	Reasons for varying impact
Human	89	11	Healthcare spending in advanced economies is twice that of developing economies
Homes	77	23	Higher penetration of the Internet of Things (IoT) in advanced economies; higher value of time saved
Offices	75	25	Impact of IoT more valuable in advanced economies because of higher costs and wages
Retail environments	71	29	Higher adoption of IoT in advanced economies
Vehicles	63	37	Higher costs in advanced economies
Cities	62	38	More autonomous vehicles in advanced economies
Factories	57	43	Larger investments in automation in advanced economies
Outside	56	44	Transportation/shipping spending higher in advanced economies
Work sites	54	46	Higher adoption of IoT in advanced economies
Overall estimate	62	38	

McKinsey&Company | McKinsey Global Institute analysis

IoT's interoperability could deliver over $4 trillion out of an $11.1 trillion economic impact.

The Internet of Things (IoT): examples of how interoperability enhances value	Potential economic impact,[1] 2025, $ trillion
Factories—data from different types of equipment used to improve line efficiency	1.3
Cities—video, cell-phone data, and sensors used to monitor traffic and optimize flow	0.7
Retail—payment and item-detection systems linked for automatic checkout	0.7
Work sites—worker- and machinery-location data used to avoid accidents	0.5
Vehicles—equipment-usage data used in presales analytics and insurance underwriting	0.4
Agriculture—multiple sensor systems used to improve farm management	0.3
Outside—inventory levels monitored at various stages of the supply chain	0.3
Homes—data from household energy systems used to track time usage	0.1
Offices—data from building systems and other buildings used to improve security	<0.1

[1]Includes sized applications only; includes consumer surplus.

McKinsey&Company | Source: Expert interviews; McKinsey Global Institute analysis

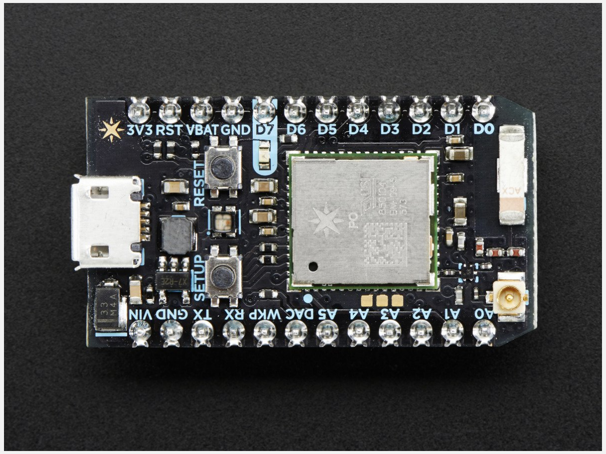
Arduino was a good place to start

Arduino specs:

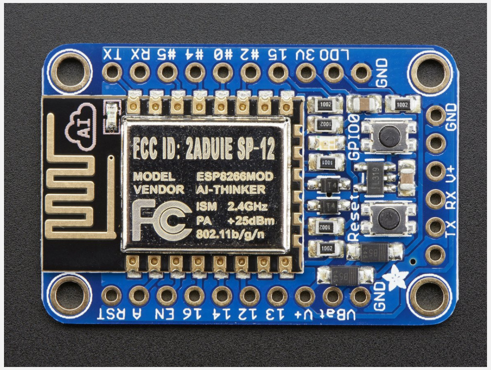
- 16MHz Atmega328 microcontroller
- 32KB flash, 2KB SRAM
- Digital I/O: 14 pins, Analog pins: 6

- *But there are more powerful and cheaper devices better suited for IoT, for example:*

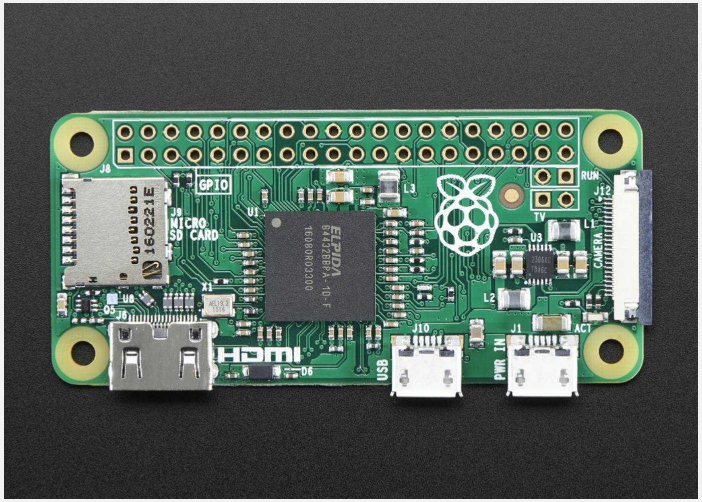
Particle Photon ($19)



ESP8266 WiFi ($10-$16)

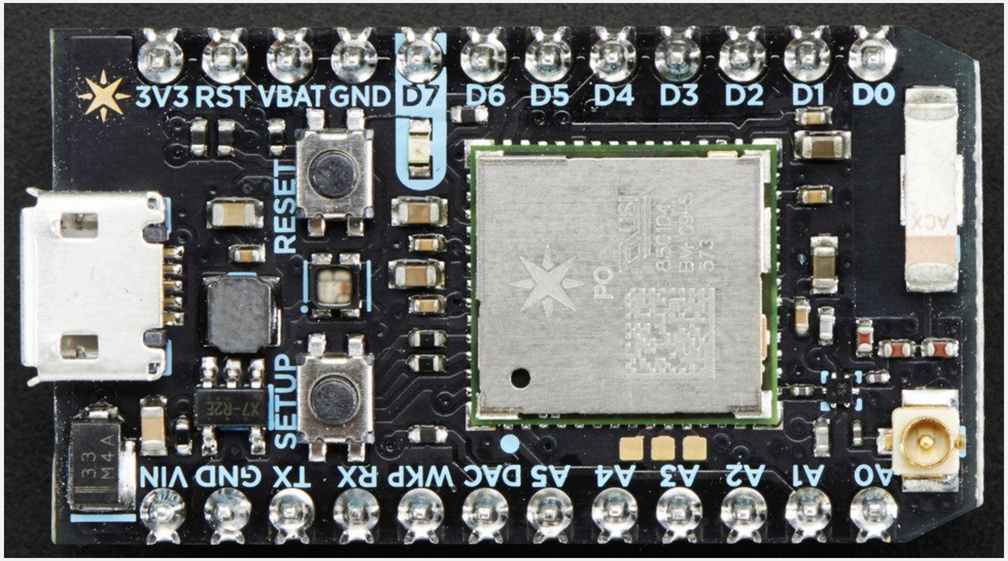


Rasberry PI Zero ($5)



Images from http://adafruit.com

Particle Photon



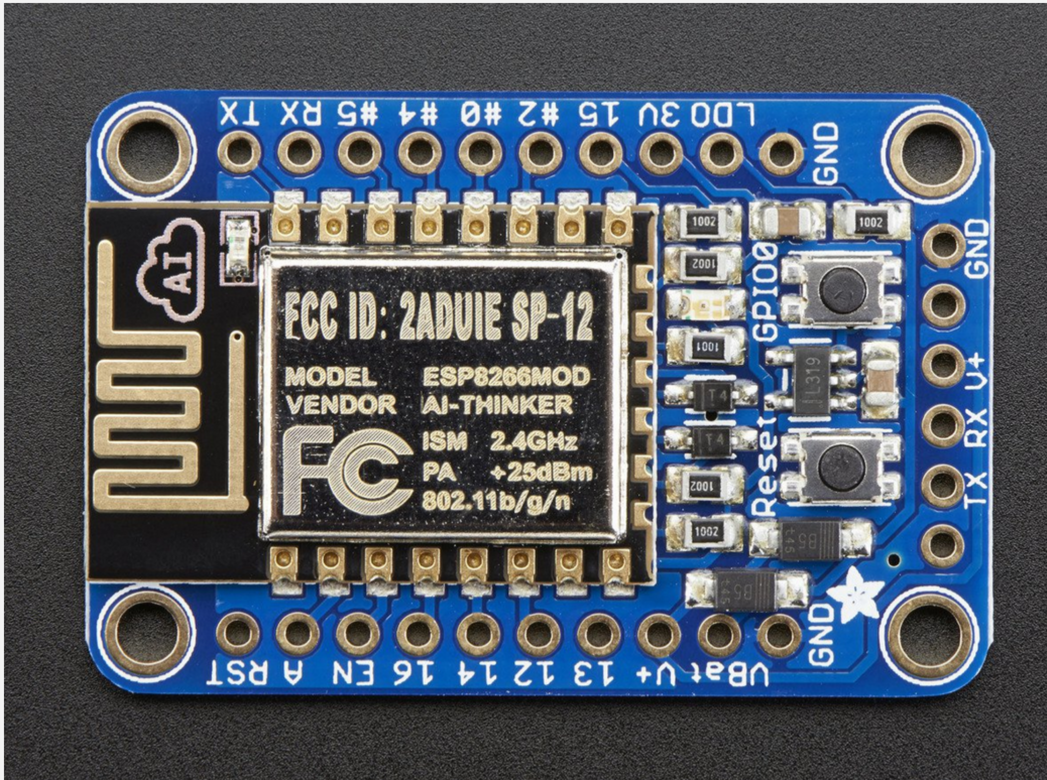
- 120MHz ARM processor
- 1MB flash, 128KB RAM
- WiFi 802.11b/g/n
- I/O:

Peripheral Type	Qty	Input(I) / Output(O)	FT[1] / 3V3[2]
Digital	18	I/O	FT/3V3
Analog (ADC)	8	I	3V3
Analog (DAC)	2	O	3V3
SPI	2	I/O	3V3
I2S	1	I/O	3V3
I2C	1	I/O	FT
CAN	1	I/O	FT
USB	1	I/O	3V3
PWM	9[3]	O	3V3

```
void setup() {
    Particle.publish("my-event","The internet just got smarter!");
}
```

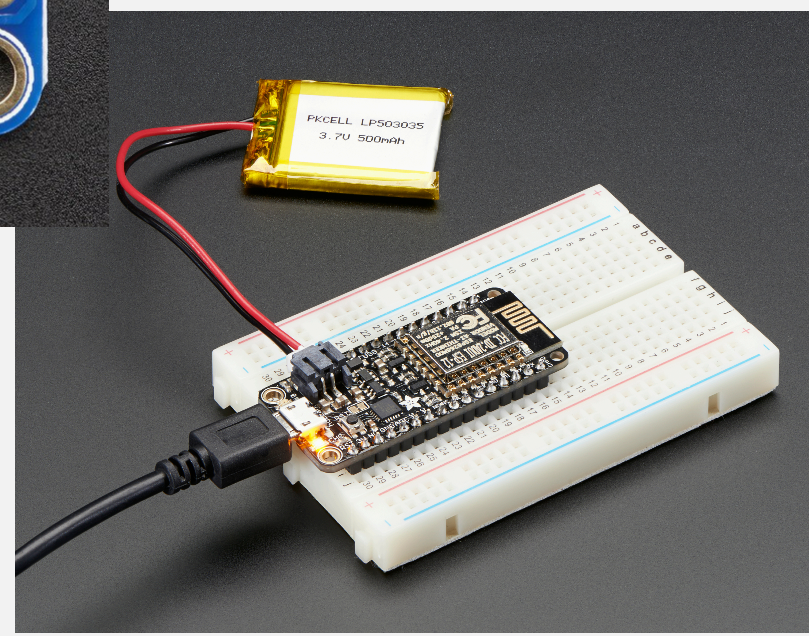
https://docs.particle.io/datasheets/photon-datasheet/

ESP8266 WiFi

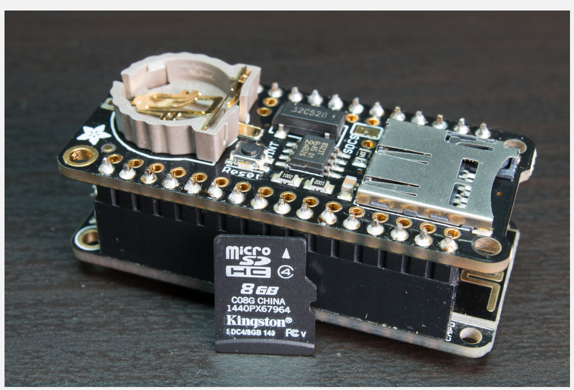


- Arduino IDE programmable
- 26-52MHz processor
- 1MB flash, 36KB RAM
- WiFi 802.11b/g/n
- 16 GPIO
- 1 ADC

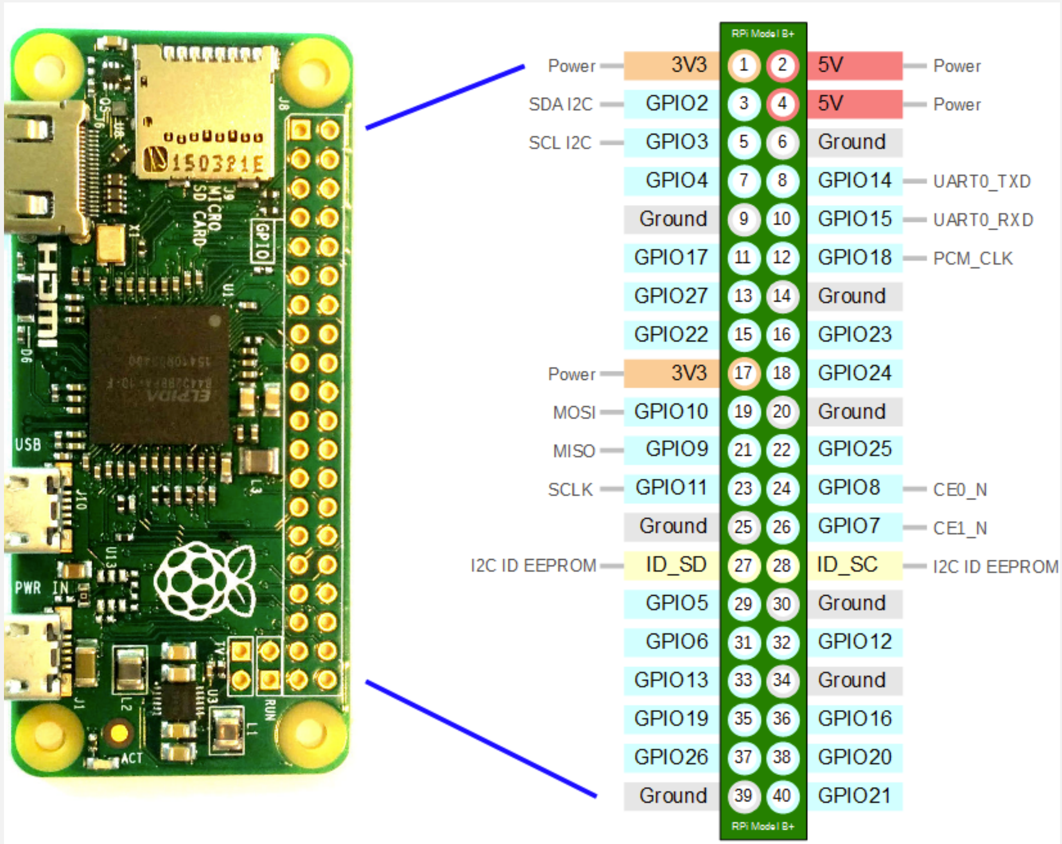
It can run MicroPython!



RTC + SD add on:

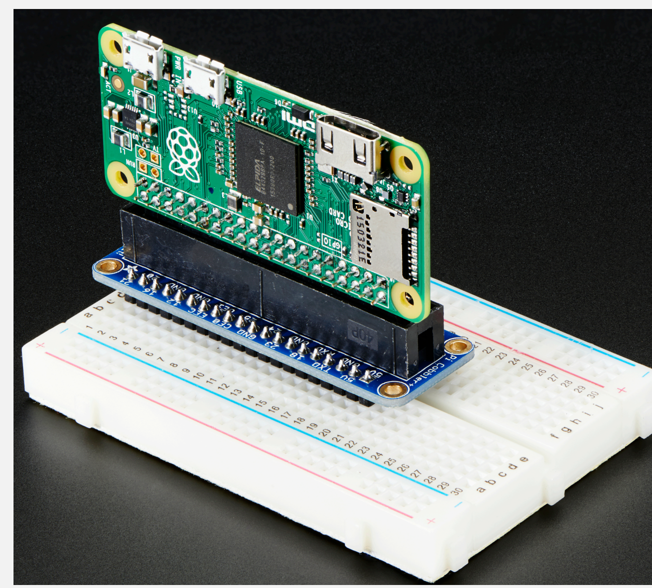


Raspberry Pi Zero



https://leanpub.com/site_images/jerpi/rpiZ-08.png

- 1GHz ARM processor
- SD card holder
- Video+Audio out
- USB port OTG
- I2C, SPI, lots of GPIO
- No WiFi/eth
- Cons: no programmable real time unit



Mission: Get up to Speed Fast

- We're going to do a crash course this first week to get you going super-fast
- Involves some hardware proficiency (PHYS120)
  - hooking up elements in breadboard, e.g.
- But mostly it's about coding and understanding how to access Arduino functions
- Emphasis will be on *doing* first, *understanding* later
  - not always a natural approach, but four weeks is short
- Monday lecture will often focus on upcoming lab
- Wed. will elaborate and show in-class examples
- Friday may often provide context/background

Every Arduino "Sketch"

- Each "sketch" (code) has these common elements

```
// variable declarations, like
const int LED=13;

void setup()
{
  // configuration of pins, etc.
}

void loop()
{
  // what the program does, in a continuous loop
}
```

- Other subroutines can be added, and the internals can get pretty big/complex

Rudimentary C Syntax

- Things to immediately know
  - anything after // on a line is ignored as a comment
  - braces { } encapsulate blocks
  - semicolons ; must appear *after every command*
    - exceptions are conditionals, loop invocations, subroutine titles, precompiler things like `#include`, `#define`, and a few others
  - every variable used in the program needs to be declared
    - common options are `int`, `float`, `char`, `long`, `unsigned long`, `void`
    - conventionally happens at the top of the program, or within subroutine if confined to { } block
  - Formatting (spaces, indentation) are irrelevant in C
    - but it is to your great benefit to adopt a rigid, readable format
    - much easier to read/debug if indentation follows consistent rules

Example Arduino Code

```
// blink_LED. . . . . . . slow blink of LED on pin 13
const int LED = 13;     // LED connected to pin 13
                        // const: will not change in prog.

void setup()            // obligatory; void->returns nada
{
  pinMode(LED, OUTPUT);// pin 13 as output (Arduino cmd)
}

void loop()             // obligatory; returns nothing
{
  digitalWrite(LED, HIGH); // turn LED ON (Arduino cmd)
  delay(1000);             // wait 1000 ms (Arduino cmd)
  digitalWrite(LED, LOW);  // turn LED OFF
  delay(1000);             // wait another second
}
```

Comments on Code

- Good practice to start code with descriptive comment
  - include name of sketch so easy to relate print-out to source
- Most lines commented: also great practice
- Only one integer variable used, and does not vary
  - so can declare as `const`
- `pinMode()`, `digitalWrite()`, and `delay()` are Arduino commands
- `OUTPUT`, `HIGH`, `LOW` are Arduino-defined constants
  - just map to integers: 1, 1, 0, respectively
- Could have hard-coded `digitalWrite(13,1)`
  - but less human-readable than `digitalWrite(LED, HIGH)`
  - also makes harder to change output pins (have to hunt for each instance of 13 and replace, while maybe not every 13 should be)

Arduino-Specific Commands

- Command reference: http://arduino.cc/en/Reference/HomePage
  - Also abbr. version in Appendix C of *Getting Started* book (2nd ed.)
- In first week, we'll see:
  - `pinMode(`*pin*, [INPUT | OUTPUT]`)`
  - `digitalWrite(`*pin*, [LOW | HIGH]`)`
  - `digitalRead(`*pin*`)` → `int`
  - `analogWrite(`*pin*, [0…255]`)`
  - `analogRead(`*pin*`)` → `int` in range [0..1023]
  - `delay(`*integer milliseconds*`)`
  - `millis()` → `unsigned long` (ms elapsed since reset)

Arduino Serial Commands

- Also we'll use serial communications in week 1:
  - `Serial.begin(`*baud*`)`: in `setup`; 9600 is common choice
  - `Serial.print(`*string*`)`: *string* → `"example text "`
  - `Serial.print(`*data*`)`: prints *data* value (default encoding)
  - `Serial.print(`*data,encoding*`)`
    - *encoding* is DEC, HEX, OCT, BIN, BYTE for format
  - `Serial.println()`: just like `print`, but CR & LF (`\r\n`) appended
  - `Serial.available()` → `int` (how many bytes waiting)
  - `Serial.read()` → `char` (one byte of serial buffer)
  - `Serial.flush()`: empty out pending serial buffer

Types in C

- We are likely to deal with the following types

```
char c;              // single byte
int i;               // typical integer
unsigned long j;     // long positive integer
float x;             // floating point (single precision)
double y;            // double precision

c = 'A';
i = 356;
j = 230948935;
x = 3.1415927;
y = 3.14159265358979;
```

- Note that the variable `c='A'` is just an 8-bit value, which happens to be 65 in decimal, 0x41 in hex, 01000001
  - could say `c = 65`; or `c = 0x41`; with equivalent results
- Not much call for double precision in Arduino, but good to know about for other C endeavors

Changing Types (Casting)

- Don't try to send float values to pins, and watch out when dividing integers for unexpected results
- Sometimes, we need to compute something as a floating point, then change it to an integer
  - `ival = (int) fval;`
  - `ival = int(fval);  // works in Arduino, anyhow`
- Beware of integer math:
  - 1/4 = 0; 8/9 = 0; 37/19 = 1
  - so sometimes want `fval = ((float) ival1)/ival2`
  - or `fval = float(ival1)/ival2 //okay in Arduino`

Conditionals

- The if statement is a workhorse of coding
  - `if (i < 2)`
  - `if (i <= 2)`
  - `if (i >= -1)`
  - `if (i == 4) // note difference between == and =`
  - `if (x == 1.0)`
  - `if (fabs(x) < 10.0)`
  - `if (i < 8 && i > -5)          // && = and`
  - `if (x > 10.0 || x < -10.0)  // || = or`
- Don’t use assignment (=) in test clauses
  - Remember to double up ==, &&, ||
- Will execute single following command, or next { } block
  - wise to form { } block even if only one line, for readability/expansion
- Can combine with else statements for more complex behavior

If..else construction

- Snippet from code to switch LED ON/OFF by listening to a button

```
void loop()
{
  val = digitalRead(BUTTON);
  if (val == HIGH){
    digitalWrite(LED, HIGH);
  } else {
    digitalWrite(LED, LOW);
  }
}
```

- BUTTON and LED are simply constant integers defined at the program start
- Note the use of braces
  - exact placement/arrangement unnec., but be consistent

For loops

- Most common form of loop in C
  - also `while`, `do..while` loops
  - associated action encapsulated by braces

```
int k,count;

count = 0;
for (k=0; k < 10; k++)
{
  count += 1;
  count %= 4;
}
```

- `k` is iterated
  - assigned to zero at beginning
  - confined to be less than 10
  - incremented by one after each loop (could do `k += 1`)
- `for(;;)` makes infinite loop (no conditions)
- `x += 1` means `x = x + 1`; `x %= 4` means `x = x % 4`
  - `count` will go 1, 2, 3, 0, 1, 2, 3, 0, 1, 2 then end loop

#define to ease the coding

```
#define NPOINTS 10
#define HIGHSTATE 1
```

- `#define` comes in the "preamble" of the code
  - note no semi-colons
  - just a text replacement process: any appearance of NPOINTS in the source code is replaced by 10
  - Convention to use all CAPs to differentiate from normal variables or commands
  - Now to change the number of points processed by that program, only have to modify one line
  - Arduino.h defines handy things like HIGH = 0x1, LOW = 0x0, INPUT = 0x0, OUTPUT = 0x1, INPUT_PULLUP = 0x2, PI, HALF_PI, TWO_PI, DEG_TO_RAD, RAD_TO_DEG, etc. to make programming easier to read/code

Voices from the Past

- avoid magnets in projects (2013)
- heat sinks are there for a reason (2013)
- make circuit diagrams & update changes (2013)
- robots are **stupid** (2013, 2014)
- use the oscilloscope (2013)
- **save often**, and different versions (2013, 2014, 2015)
- some lectures are boring, **but boring ≠ useless** (2013)
- start early (2014)
- comment your code (2014)
- take more time to think than to code (2014)
- don't use perf-board unless you rock at soldering (2014)

Voices, Continued

- Listen to Professors and TAs (2014)
- Use Serial Monitor and DVM for debugging (2014, 2015)
- Pin conflicts are real! (2014)
- Know what pins are used by your shield (2014)
- Read the data sheets (2014)
- Walk away if something doesn't work (2014)
- Know the purpose of every line of code (2015)
- A simple concept might not be so simple (2015)
- Pick a project that can be scaled up or down (2015)
- Get your own Arduino & practice/explore (2015)
- Batteries can be a real pain (2015)
- Make a set schedule with partner (2015)

Announcements

- Can go to lab right after class to start on kits
  - otherwise Monday or Tuesday lab at normal 2PM start time
- Late labs (even by an hour) incur grade-point penalty
  - very important (for project) to avoid slippage
  - can accelerate by jumping through labs ahead of schedule
- Will have midterm to check basic coding proficiency
- Grading scheme:
  - 50% project (proposal, implementation, success, report)
  - 36% weekly lab (4 installments: success/demo, write-up)
  - 10% midterm (coding example)
  - 4% participation/attendance of lecture

Course Website

- Visit

  http://physics124.barreiro.ucsd.edu

  - Assignments
  - Lab Exercises
  - Useful Links
  - Contact Info & Logistics