How to Become an IoT Developer (and Have Fun!)

Justin Mclean
Class Software

Email: justin@classsoftware.com
Twitter: @justinmclean

APACHECON
North America

Who am I?

- Freelance Developer - programming for 25 years
- Incubator PMC and Apache Flex PMC plus a few others, Apache member and a mentor for several incubating projects
- Run IoT meetup in Sydney Australia

How I got here

- Been coding since the 80s
- Started on low level machine code and C programming
- Worked on a few early “IoT” projects
- Internet come along
- Open Source Hardware come along
- First conference talk on Arduino
- Started IoT Sydney Meetup
- Back to coding in C and working on hardware

Things have changed

- Access to low cost easy to program hardware
- Constrained hardware has more memory and speed
- “Modern” development tools and IDEs
- Some standardisation
- Open Source hardware community
- Open Source libraries

Hardware is hard

- Can’t revert changes easily or make changes once deployed
- People underestimate time taken of developing firmware
- It harder to debug and find errors with hardware
- Hard to update firmware
- Security issues
- Power issues

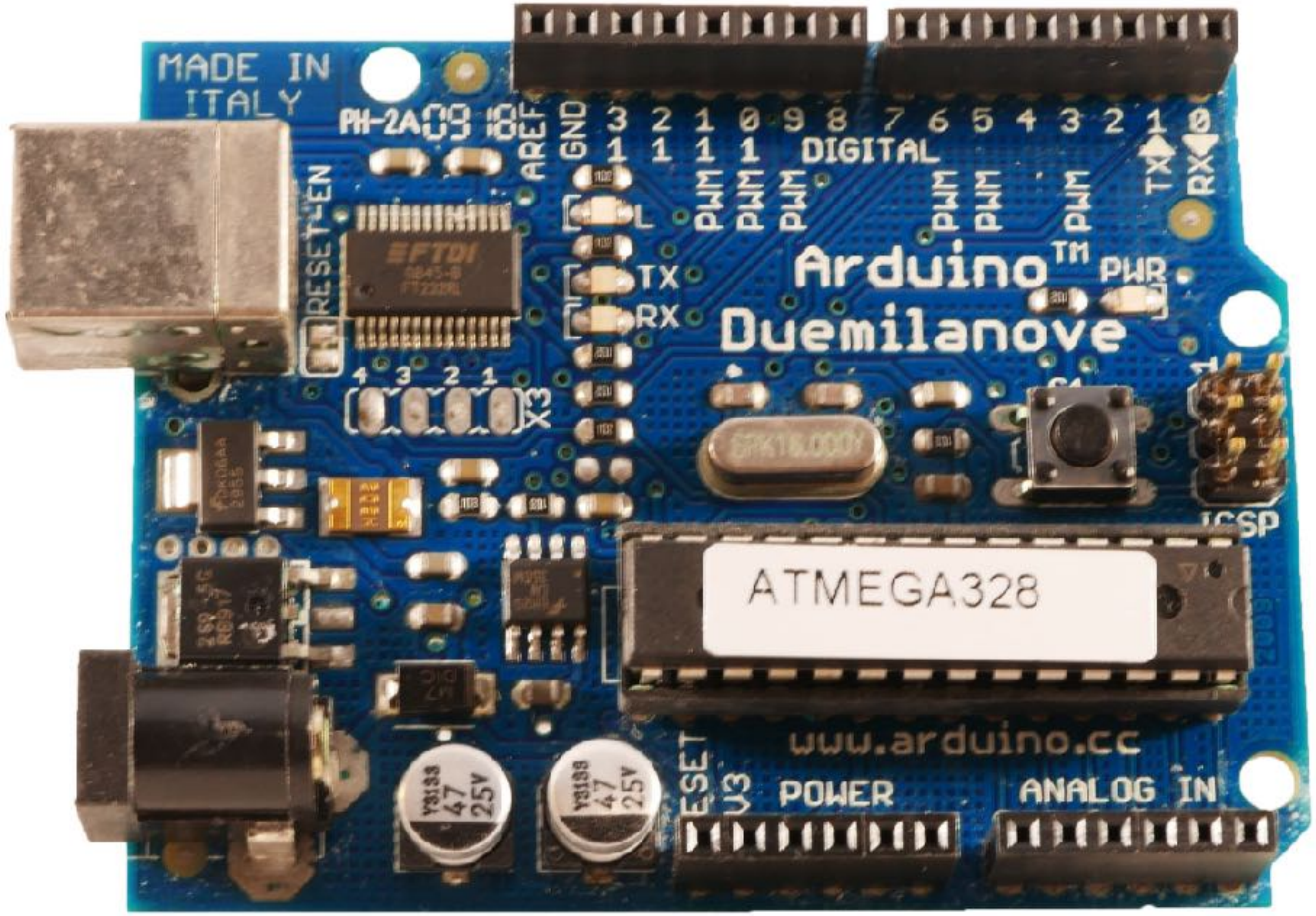
So you want to become an IoT developer?

One name different jobs

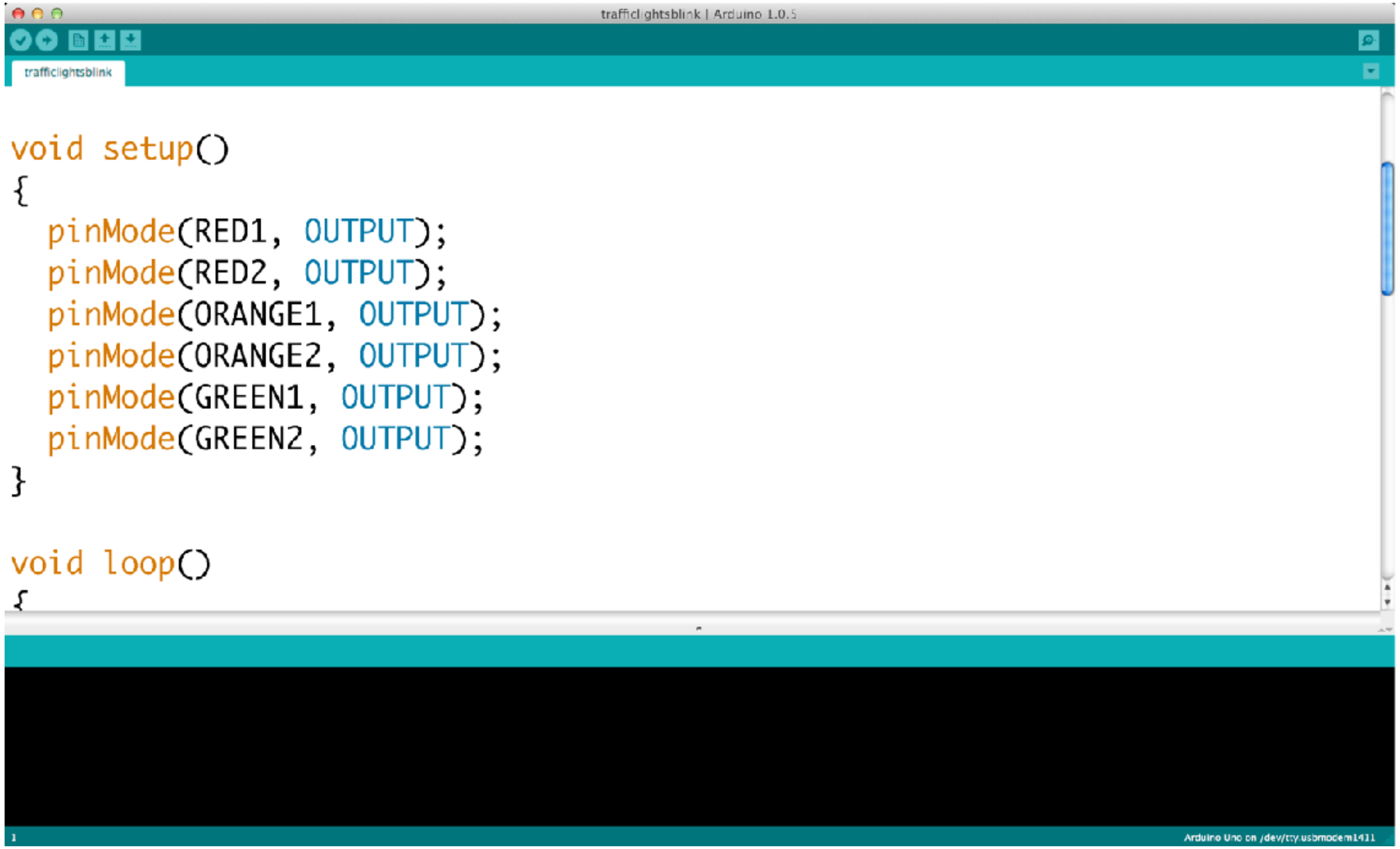
- You can be an IoT developer without touching the hardware i.e big data project
- I’m focussing on the embedded / hardware side but form a software point of view

Play with toys

- Get yourself an Arduino or Raspberry Pi or similar
- Find yourself a project
  - simple as blinking leds
  - or monitoring the environment
  - or displaying messages
  - or logging your beer brewing


MADE IN ITALY
DIGITAL
Arduino™
Duemilanove
ATMEGA328
www.arduino.cc
POWER
ANALOG IN
ICSP

Arduino


trafficlightsblink | Arduino 1.0.5
trafficlightsblink
1
Arduino Uno on /dev/tty.usbmodem1411

```
void setup()
{
  pinMode(RED1, OUTPUT);
  pinMode(RED2, OUTPUT);
  pinMode(ORANGE1, OUTPUT);
  pinMode(ORANGE2, OUTPUT);
  pinMode(GREEN1, OUTPUT);
  pinMode(GREEN2, OUTPUT);
}

void loop()
{
```

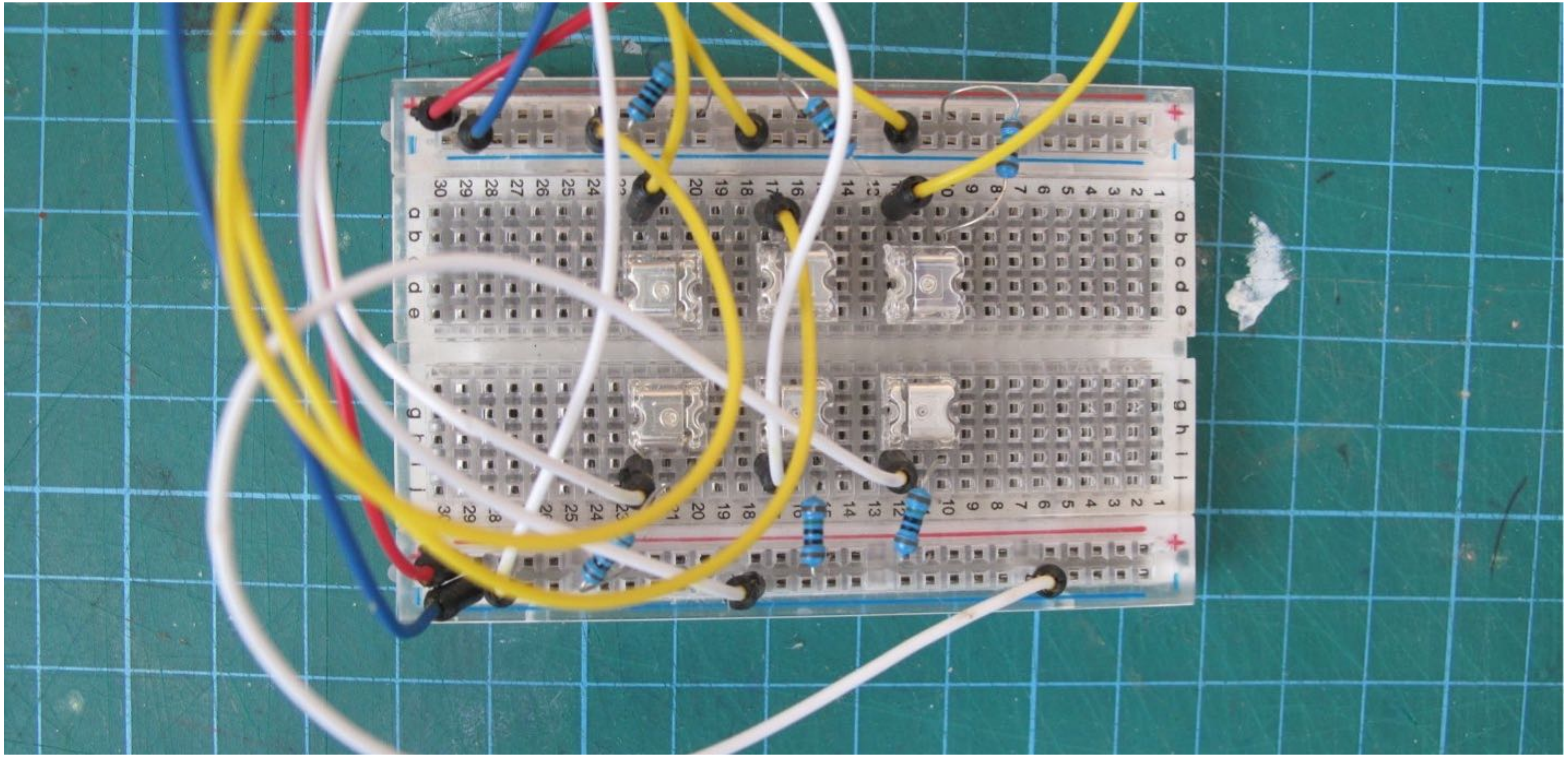
Arduino

Were to get stuff

- Adafruit
  https://www.adafruit.com
- SparkFun
  https://www.sparkfun.com
- Seeed Studio
  https://www.seeedstudio.com
- eBay but you generally get what you pay for!

Create a simple circuit

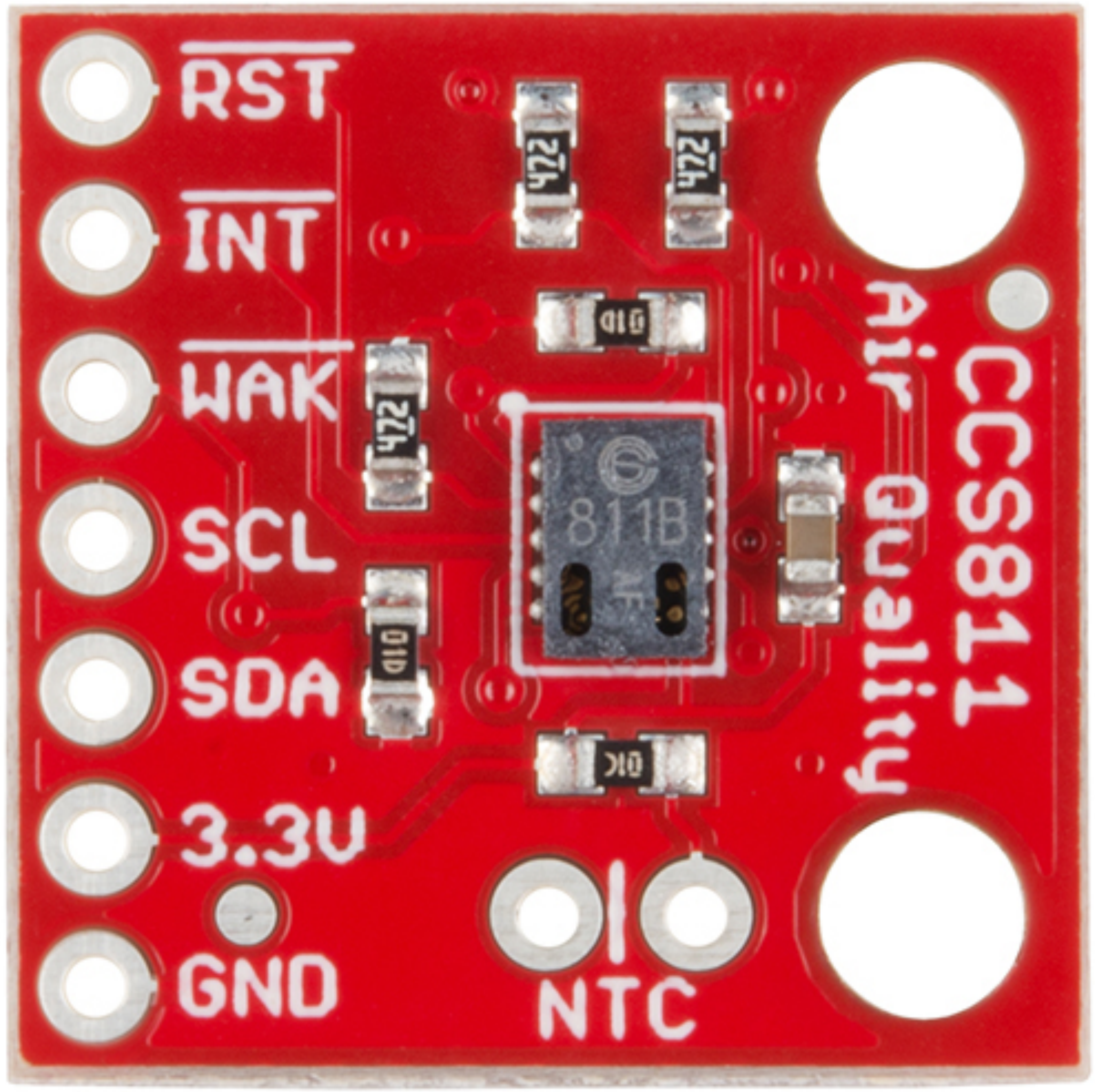
- Get a bead board and wires and make a simple circuit
- Try and create your prototype
- A multimeter may help here
- Depending on your style it may not look pretty



Breadboard

Use breakout boards

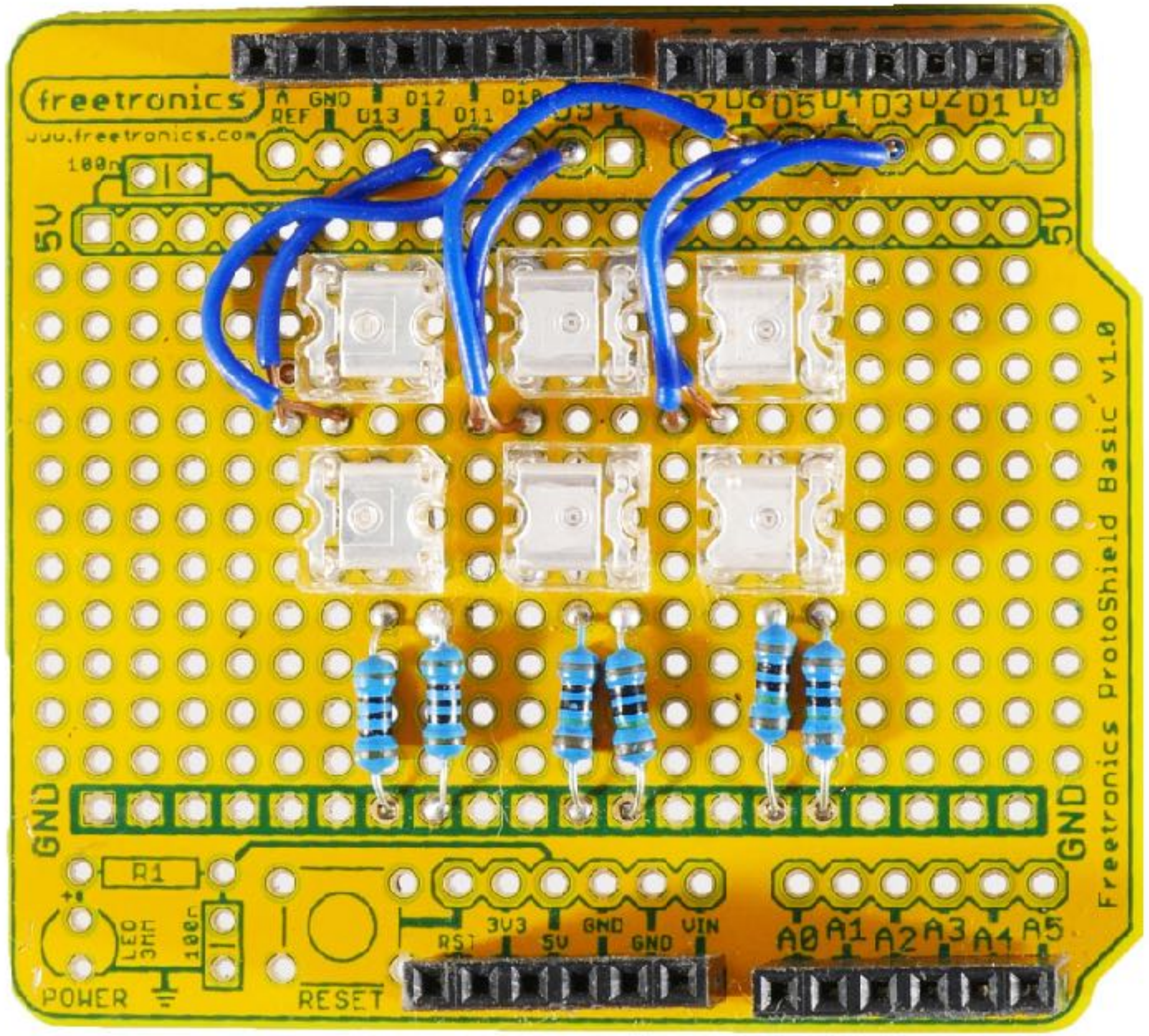
- Can get a lot of pre-assembled boards
- Easy to wire up to a breadboard
- Often use standard interfaces like I2C or SPI
- Think of them as lego blocks



Breakout Board

Learn how to solder

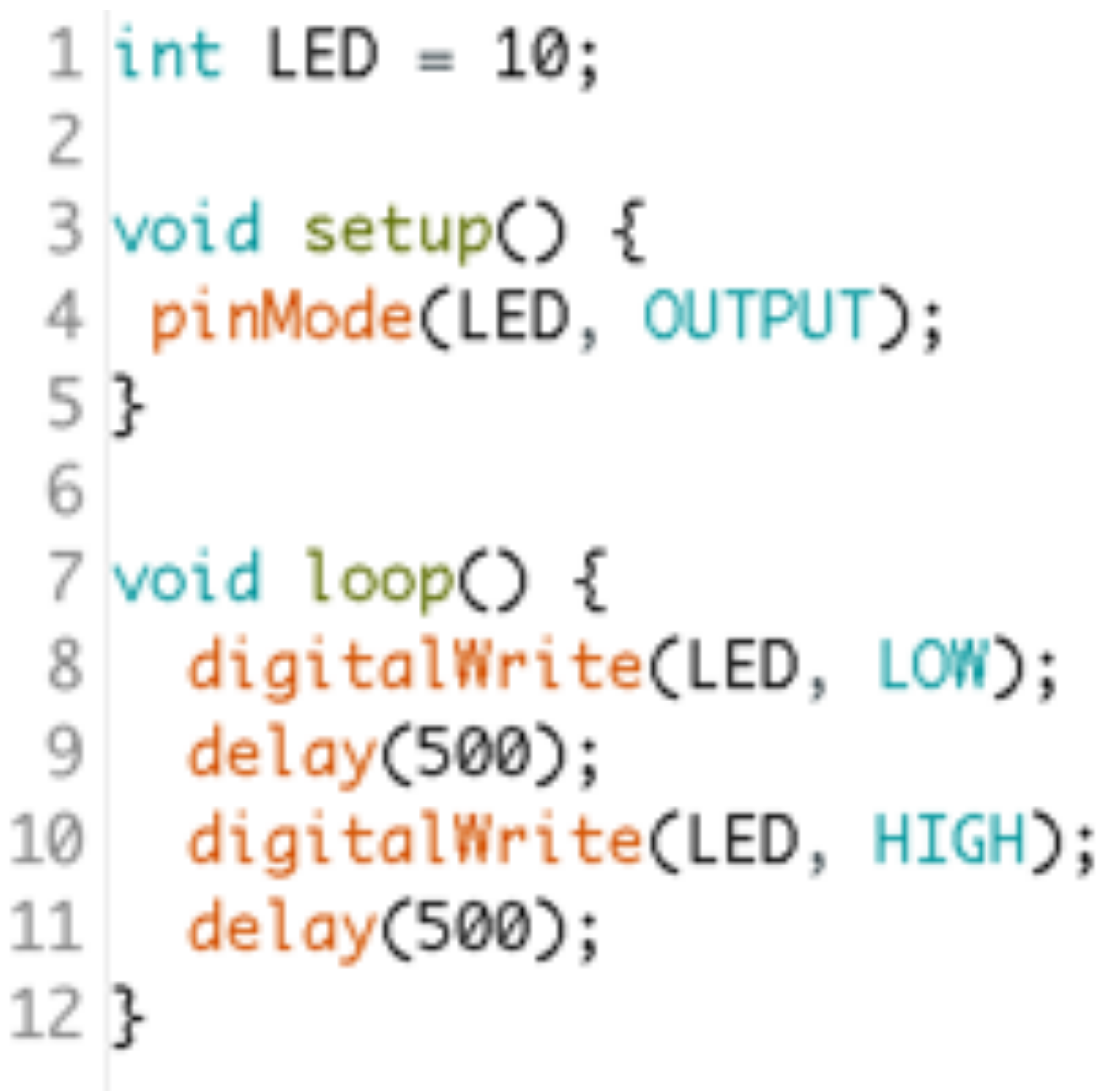
- It easier than you think
- Use the right tip and solder
- Use a flux pen
- Learn how to correct mistakes
- Start with large through hole items
- Use sockets for ICs
- Use solder braid


freetronics
www.freetronics.com
Freetronics ProtoShield Basic v1.0
5V
GND
POWER
RESET
RST
3V3
5V
GND
GND
VIN
A0
A1
A2
A3
A4
A5

Prototype

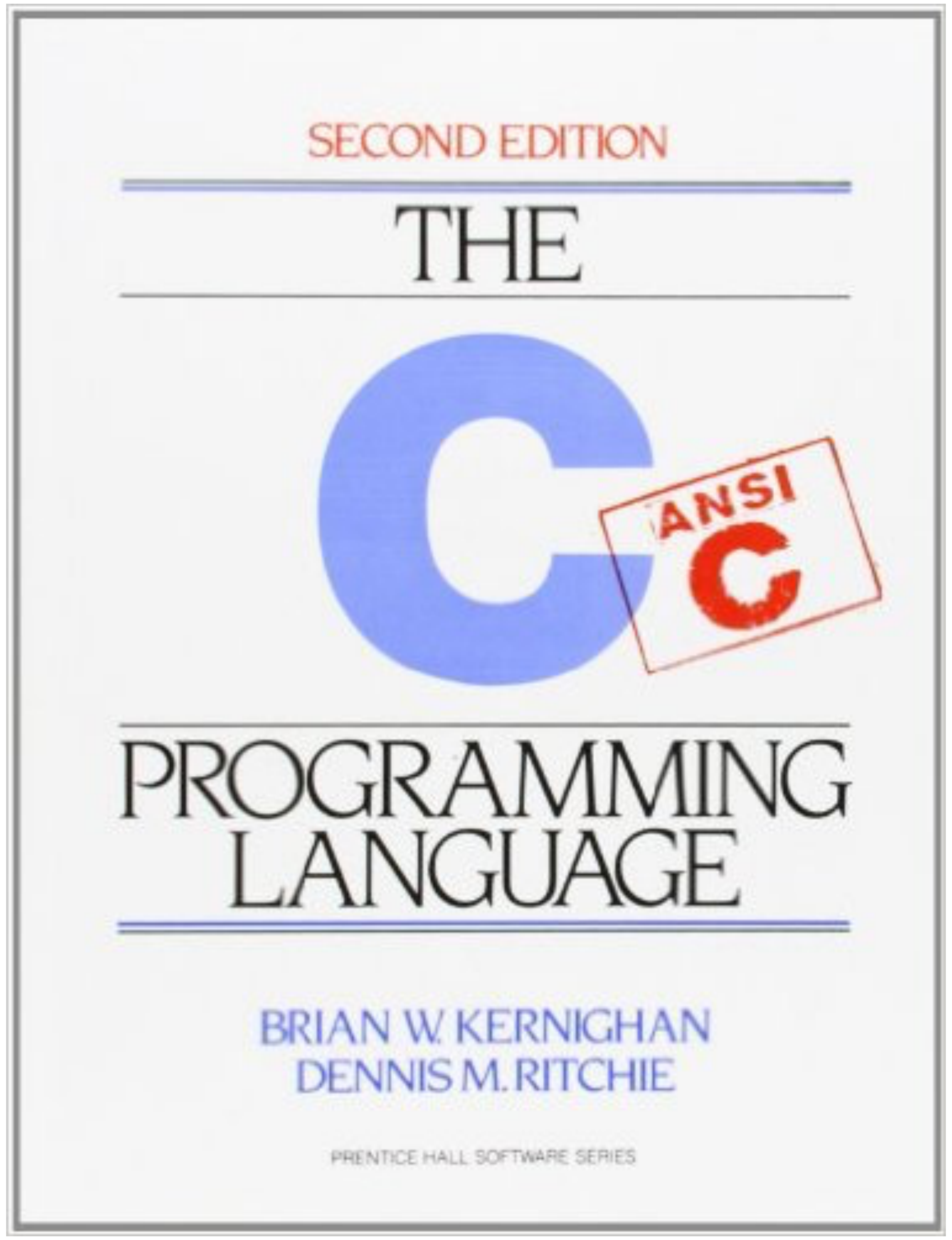
Learn a new language

- If you don't know it learn C
- Other languages exist on embedded platforms but C is most common
- May need forget some of what you know
- C is not as complex as you may think
- Modern C style is a little different

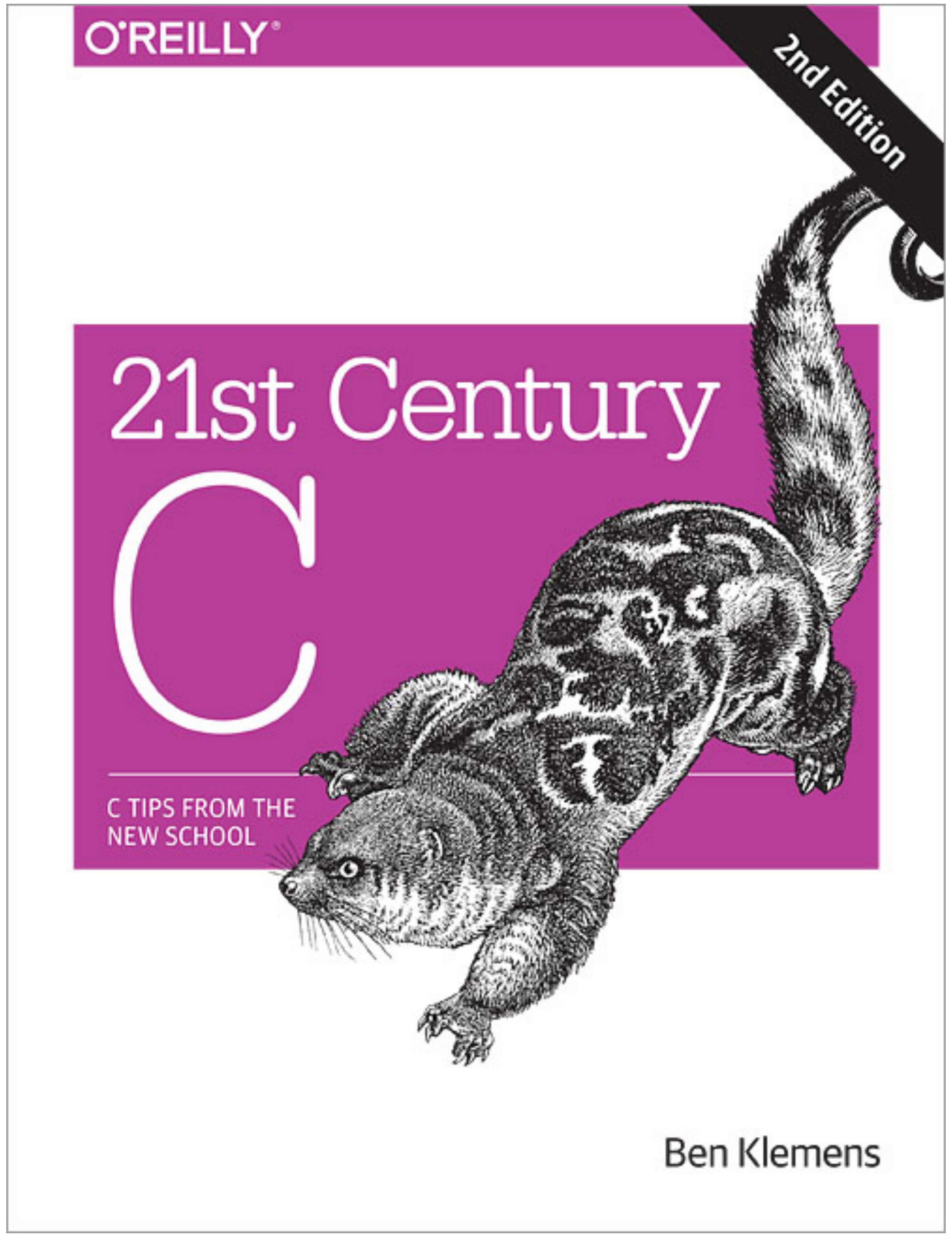


```
int LED = 10;

void setup() {
 pinMode(LED, OUTPUT);
}

void loop() {
  digitalWrite(LED, LOW);
  delay(500);
  digitalWrite(LED, HIGH);
  delay(500);
}
```

Forget what you know



Read the classics



Or a more modern book

C has improved

- K&R C, C89, C99, C11
- Well perhaps only a little :-)
- Some useful C99 features:
  - bool and int types
  - auto sizing of arrays
  - floating point numbers
  - inline functions

Optimise your code later

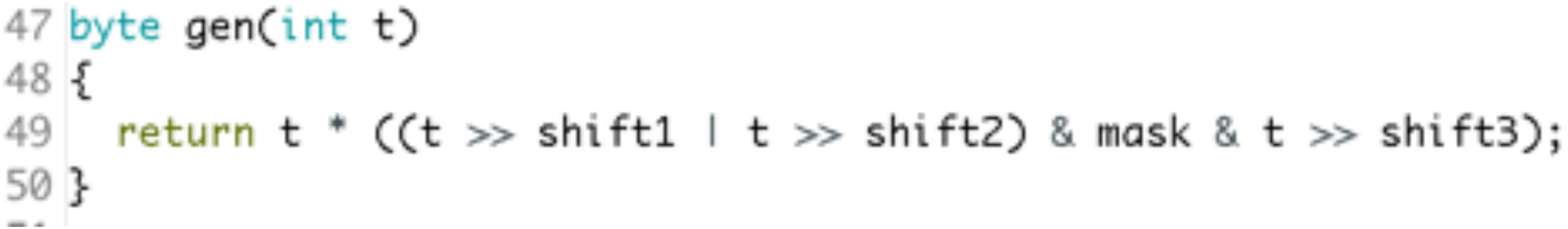
- Compiler is good at optimising code
- Only optimise if you need to
- Better to keep code simple and readable
- Refresh yourself on operator order

Code carefully

- May be best to avoid dynamic allocation of memory
- Use pointers sensibly
- Break it up - can always inline later
- Encapsulate the hard bits
- Used sized ints
- Take care with strings
- Document your code - doxygen

Size matters

- You can do a lot in a small amount of code
- Arduino web server in about 20 lines of code compiles to 2K



```
byte gen(int t)
{
  return t * ((t >> shift1 | t >> shift2) & mask & t >> shift3);
}
```

Generative Formula

Know some electronic basics

- Focus on digital logic 5V or 3.3V = 1 and 0V = 0
- Current limiting leds
- Transistors for switching
- Filtering caps
- Pull up / pull down resistors
- Voltage divider

Make a board

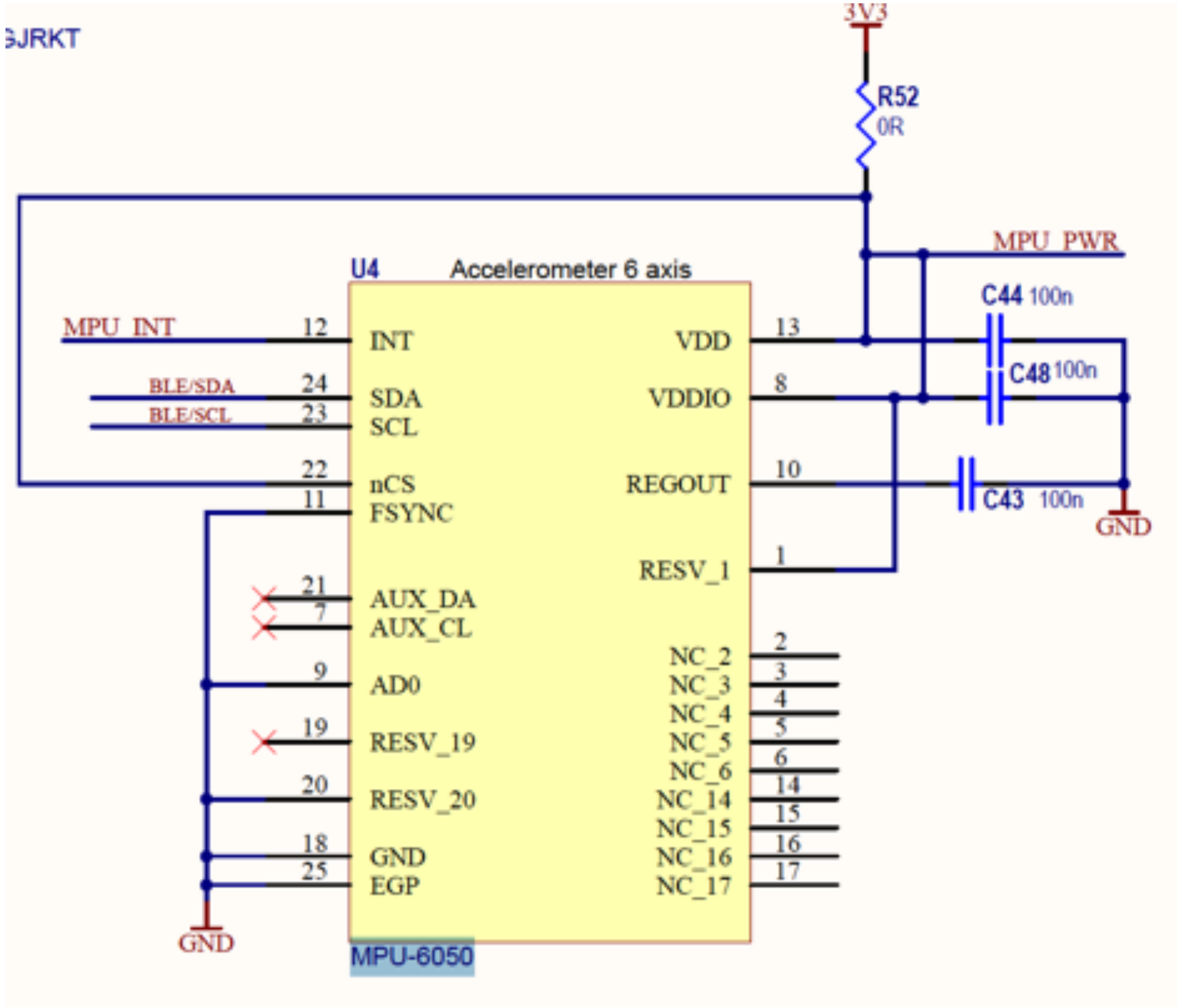
- Why? Making physical stuff is fun!
- Start off with basic PCB layout program like Fritzing
- It has bread board / circuit and PCB layout
- Don’t cross the tracks
- Use vias where needed
- Copper and ground fill

Read the data sheets

- Learn to look for important values
- Don’t worry if you don’t understand all of it
- Often contain sample circuits - bonus!
- Can contain import timing information
- Can vary in quality

Learn Schematic basics

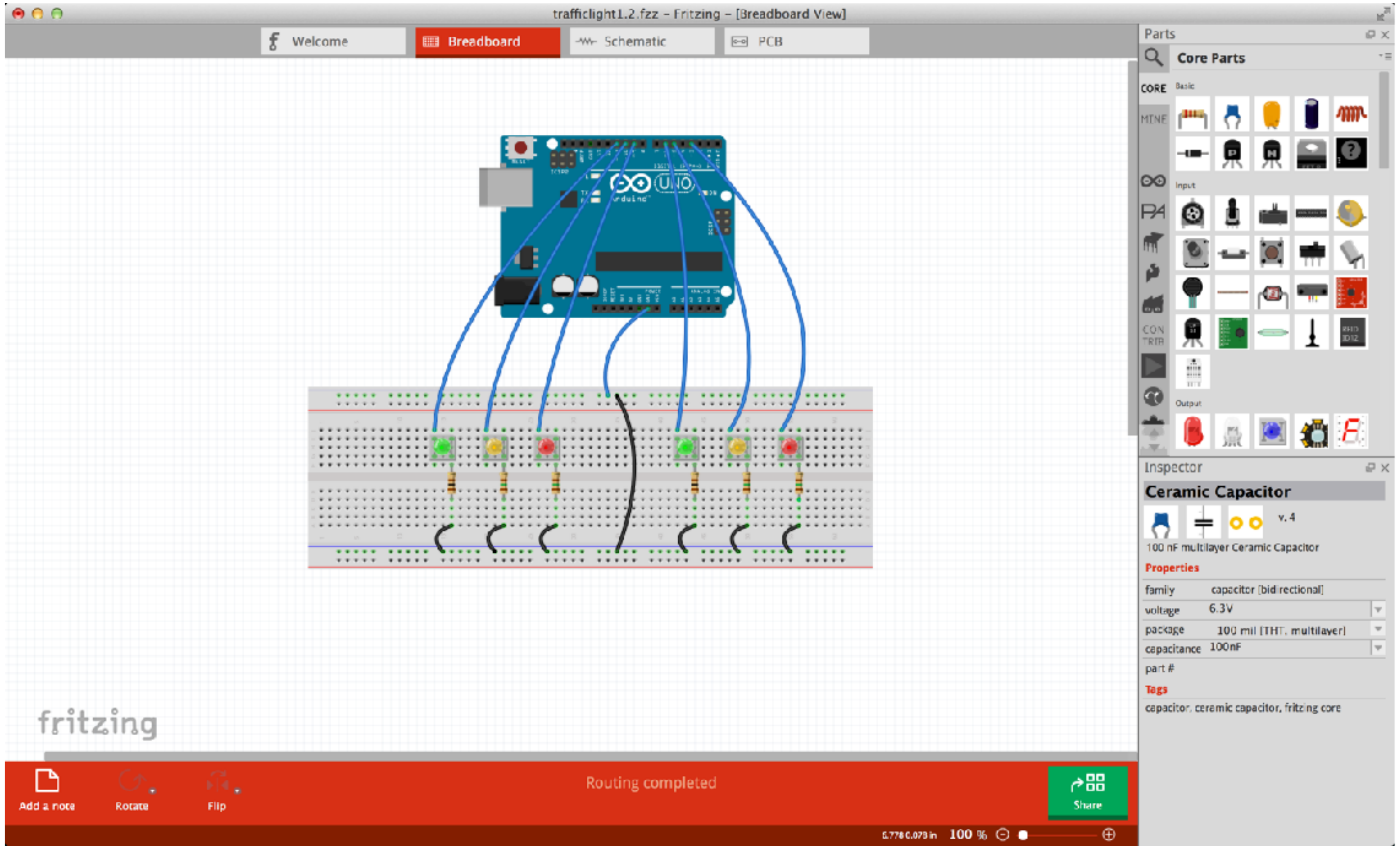
- Know the basic symbols
- Know how to match up pins on ICs


JRKT
3V3
R52
0R
MPU_PWR
U4
Accelerometer 6 axis
C44 100n
C48 100n
C43 100n
GND
MPU_INT
BLE/SDA
BLE/SCL
12 INT
24 SDA
23 SCL
22 nCS
11 FSYNC
21 AUX_DA
7 AUX_CL
9 AD0
19 RESV_19
20 RESV_20
18 GND
25 EGP
VDD 13
VDDIO 8
REGOUT 10
RESV_1 1
NC_2 2
NC_3 3
NC_4 4
NC_5 5
NC_6 6
NC_14 14
NC_15 15
NC_16 16
NC_17 17
GND
MPU-6050

Schematic

Fritzing

- http://fritzing.org/home/
- Very easy to use
- Easy export of files
- Handles surface mount and through hole components
- Comes with a decent library of footprints
- Auto route not very useful
- Breadboard view not compact



Inspector

Ceramic Capacitor

v. 4

100 nF multilayer Ceramic Capacitor

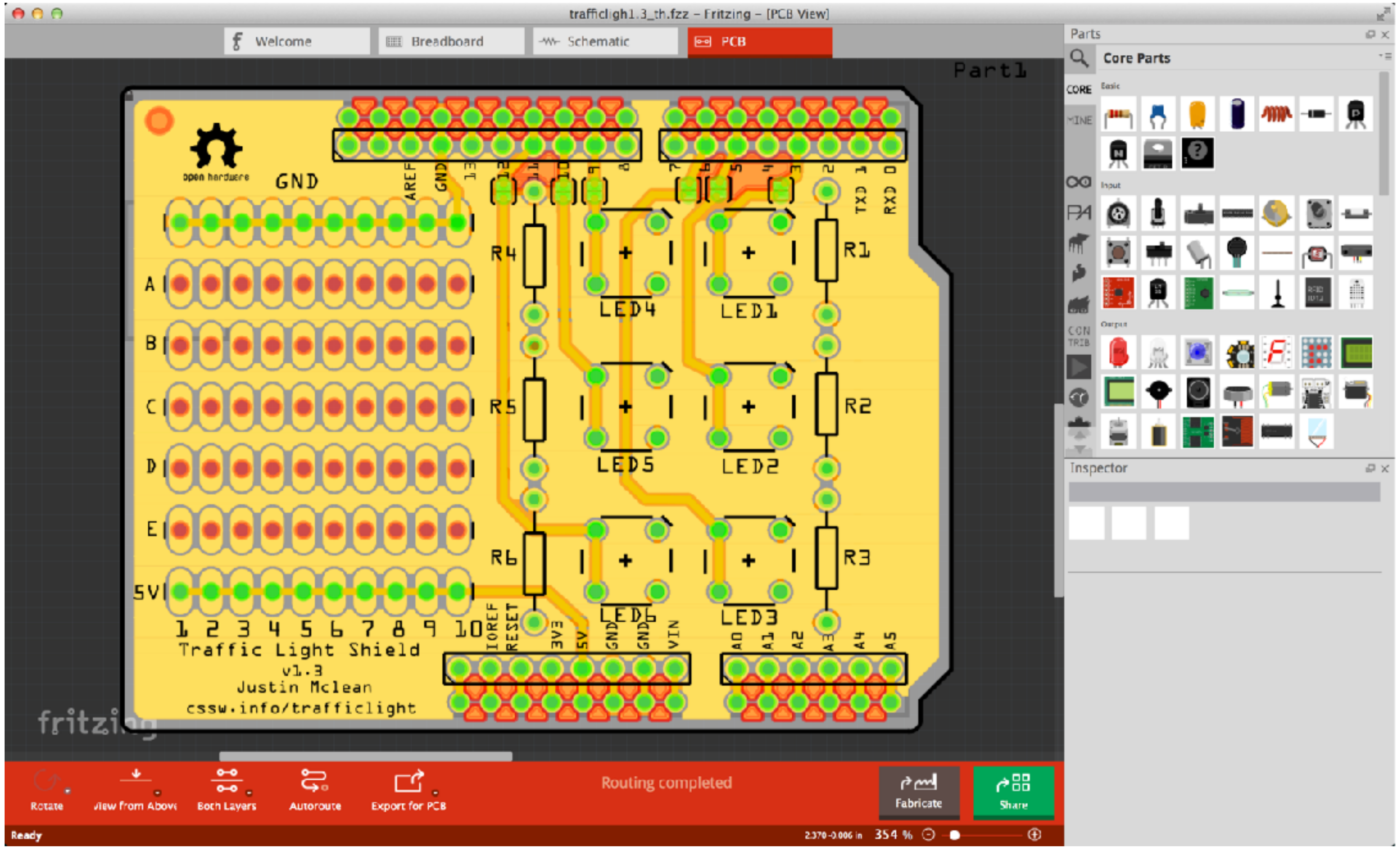
Properties

family	capacitor [bidirectional]
voltage	6.3V
package	100 mil [THT, multilayer]
capacitance	100nF
part #	

Tags

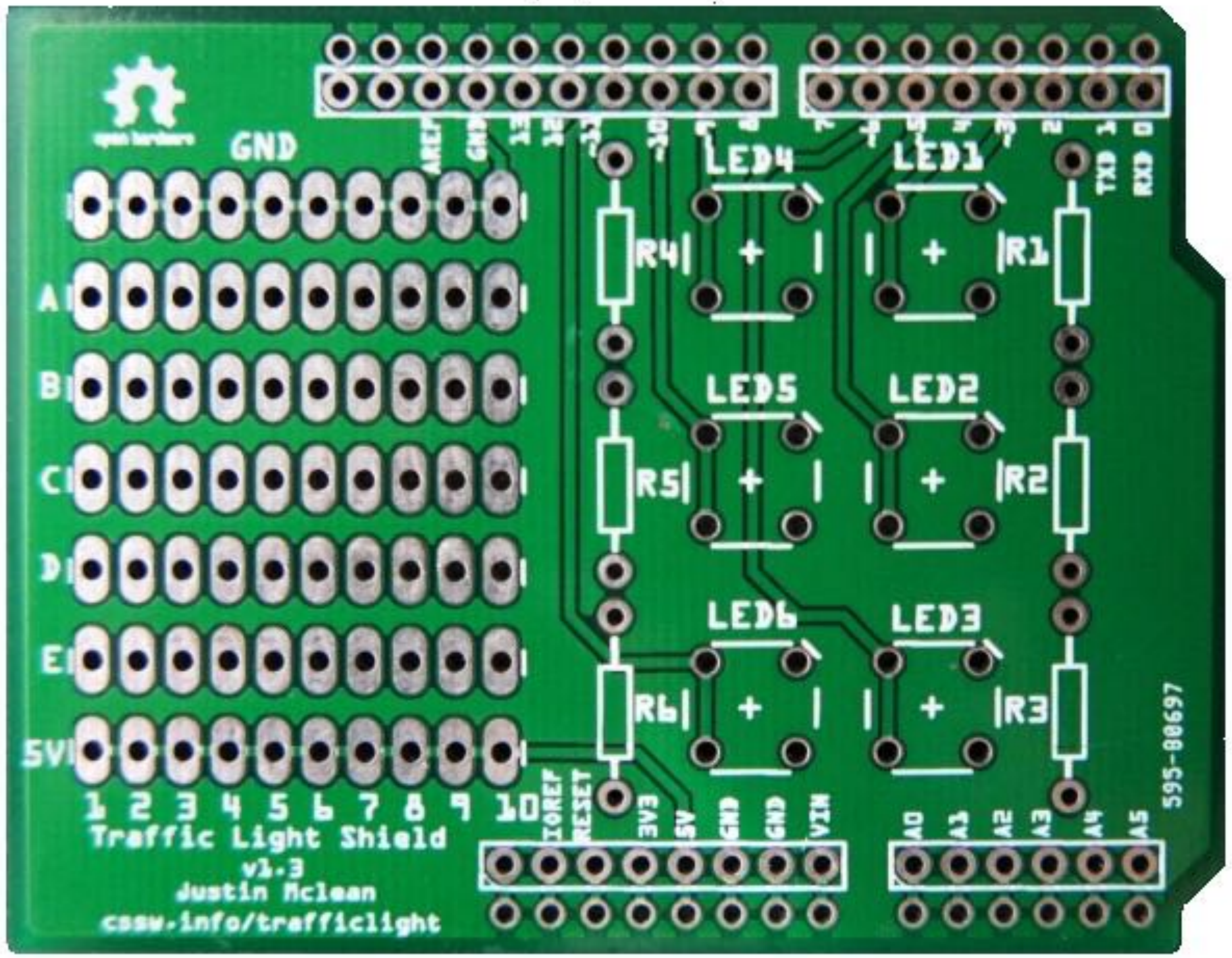
capacitor, ceramic capacitor, fritzing core

Breadboard View



Routing completed

PCB View


GND
AREF
GND
13
12
~11
~10
~9
8
7
~6
~5
4
~3
2
TX→1
RX←0
TXD
RXD
LED4
LED1
LED5
LED2
LED6
LED3
R4
R1
R5
R2
R6
R3
A
B
C
D
E
5V
1 2 3 4 5 6 7 8 9 10
IOREF
RESET
3V3
5V
GND
GND
VIN
A0
A1
A2
A3
A4
A5
595-80697
Traffic Light Shield
v1.3
Justin Mclean
cssw.info/trafficlight

Boards

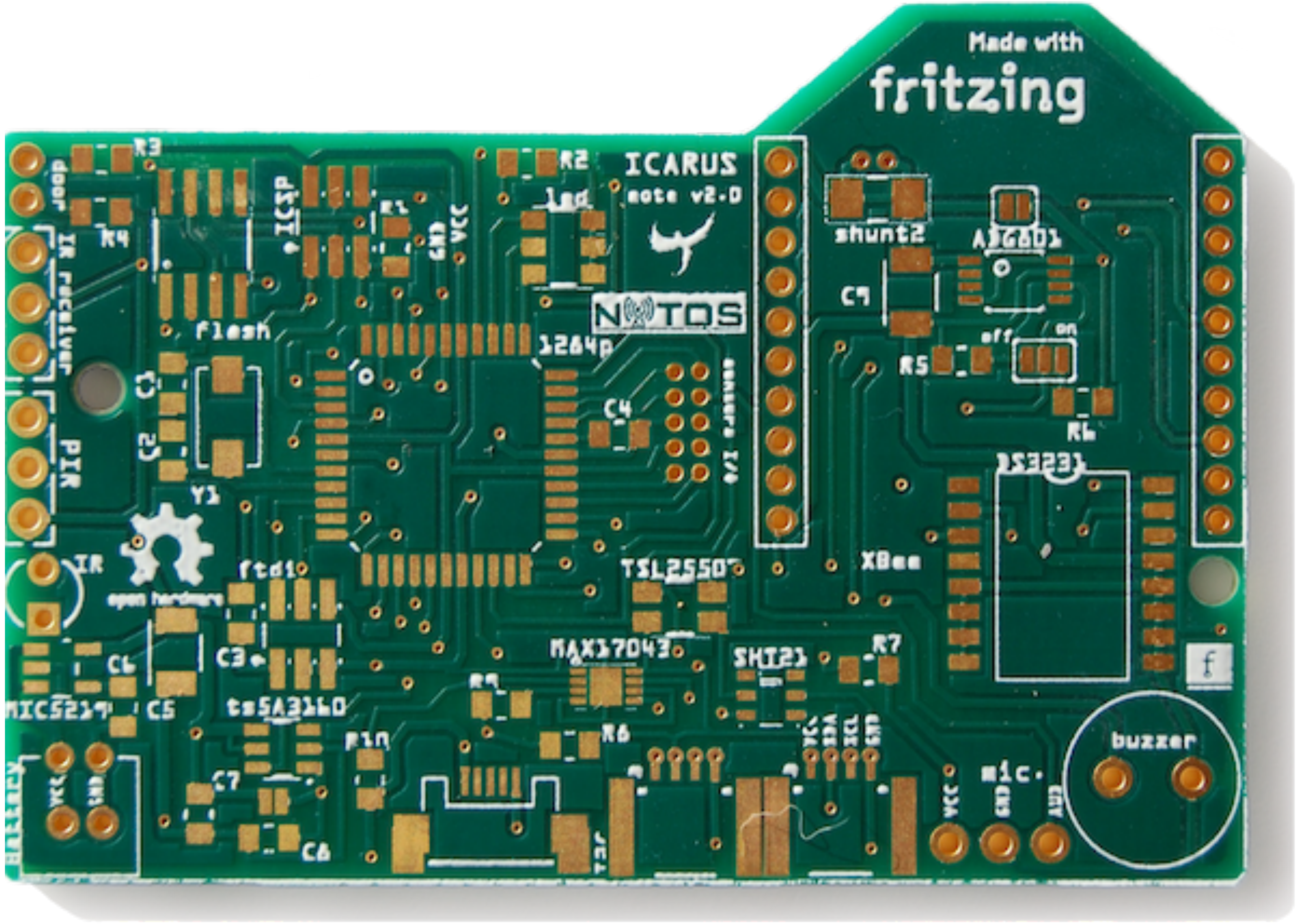
Have the right tools

- Get a good soldering iron
- Multimeter
- Side cutters
- Flux pen
- Solder braid
- Heat gun

VICTOR 70C
DIGITAL MULTIMETER
DC
276.8
mV
AUTO
APO
MAX/MIN
Hz/DUTY
HOLD
RANGE
AUTO POWER OFF
REL/RS232
POWER
DC/AC
Hz/°C/°F
mA
A
CE
10A
mA
COM
TEMP
VΩHz

Multimeter

It works!



Not just basic boards

Don't go small too quickly

- Temptation to use small cheap surface mount components right away
- Keeps the cost down but means the device may be hard to debug
- Increase time (or makes it impossible) to make modifications to the board
- Boards may have higher defect rates
- Physical copy and paste errors

Test the hardware

- Have some way of testing the hardware - usually custom program or part of the startup sequence
- Have physical test points on the board
- Make a testing rig if you need to test a number of boards

Don’t use the hardware

- Compile and test your code locally
- Standard C will work just about everywhere
- Some platforms (like Apache Mynewt) have simulators
- Stub out things that are hardware dependant
- Can be a faster workflow to work this
- Can run unit tests easily

Test on the hardware

- You need to test on real hardware
- Most modern platforms you can debug, set breakpoints, step line by line etc etc
- Make sure you test release builds as well as debug ones

Watch your memory

- While 32K or 128K sounds like a lot you may run out off memory
- Avoid dynamically allocating memory if possible
- Tools / RTOS generally have a way of showing memory usage
- Perform a burn in test
- Make sure memory doesn’t climb over time

Software is always at fault

- If something doesn’t work it’s likely to be the software not the hardware
- If you can’t find the bug it still likely to be the software
- It likely to be in your code not the 3rd party library used by 1000’s of people
- No changing libraries will not fix it
- Yes it is a bug in your code

Except when it’s the hardware

- Hardware works except when it doesn't
- If you lucky it will be DOA and do nothing or have a short and consume all the power
- If you are unlucky it will mostly work.
- Examples I’ve recently seen:
  - unmarked GPS antenna passive not active
  - crystals rotated 90 deg
  - incorrect accelerometer circuit

Log all the things

- Often hard to know what hardware is doing at any point of time
- Log what going on when debugging
- Have some way of viewing the logs (especially when the debugger is connected)
- Remove most of the logging (but not all) in production

Blinkly lights

- Use indicator leds to indicate status
- But don’t be annoying

Code on bare metal

- All the memory and speed is yours!
- Nothing else gets in the way
- All the bugs are yours!
- Some things can be more complex

Use an RTOS

- Usually have some form of simple threading or tasks
- Breaking program up into tasks can simplify code
- Take care with shared resources
- May provide other benefits re power consumption
- Be careful of vendor lock-in
- Can be more abstract / complex in some cases
- Look at documentation and support options

The not so fun bits :-(

OTA Updates

- How do you update your device?
- May be a lot harder than you think
- Bootloader
- Check and download new images
- Where do you store them?
- Verify images
- Swap between images
- Use an RTOS that supports all of this

Security

- Can be hard on constrained devices
- May not be able to do TLS due to memory or speed constraints
- Select platforms that have built in crypto or can off load crypto to another chip

Power

- Power may be a limiting factor
- Need to sleep / deep sleep / turn off all devices
- Time to wake up
- RTOS may help here

My journey

- I’ve learn lots of new skills
- Met a lot of nice people
- Be involved in a couple of communities
- Had a lot of fun
- Hope your journey will be the same