Arduino Workshop Breakout II

IEEE Arduino Workshop, Fall 2015

Kashev Dalmia & Brady Salz

Tell Me About You.

- What's your Year in School?
- Have you Taken...
  - ECE 220/190/198kl?
  - CS 225?
  - ECE 391?
  - CS 431?
  - ECE 445?
  - ECE 310/311?
- Rate Your Experience With...
  - Arduino
  - C++
  - DSP
  - Building Stuff

This is us.

- Graduate Students
- Brady: Hardware
- Kashev: Software
- Been doing this sort of thing a while.

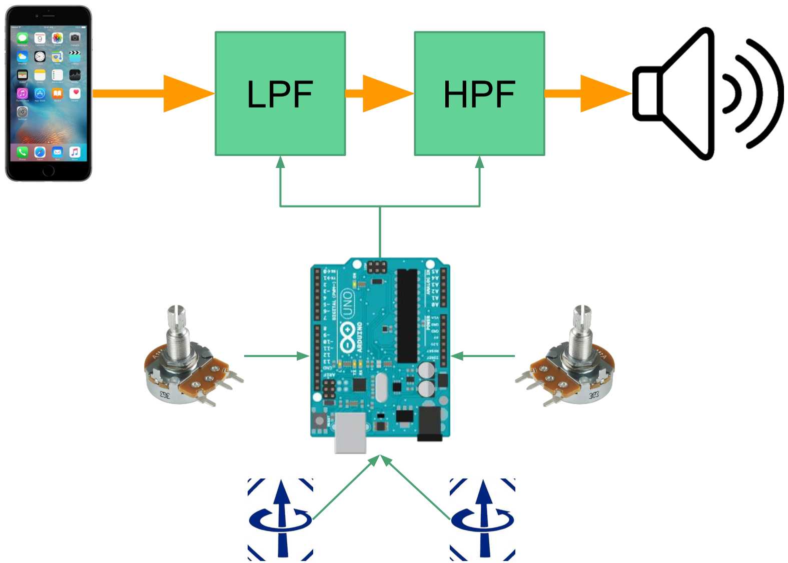
Today’s Agenda for Breakout II: **Learn About…**

- **Understanding Hardware+Software Systems**
- **Interacting with more ‘Advanced’ Parts**
- **Writing Good Software**

Understanding Hardware Systems

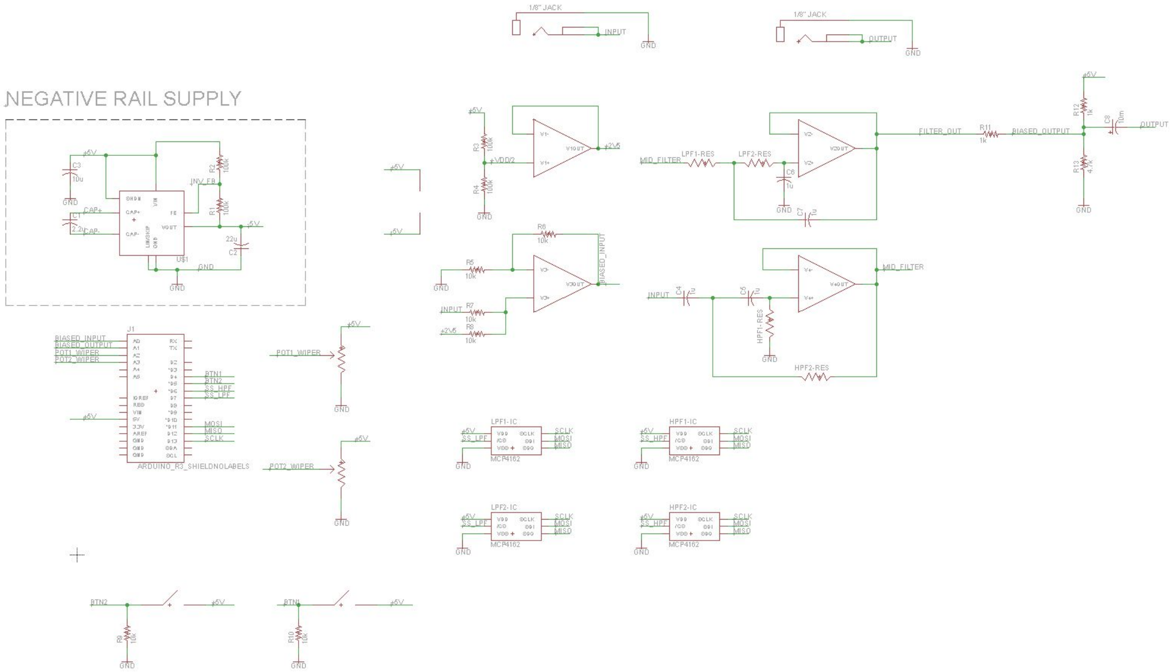
Shield System Diagram (HW Design)

- This is already done for you!
  - Thanks, Brady
- This is already assembled for you!
  - Thanks, Team
- Things to Think About:
  - Why is the audio path not running through the Arduino?
  - What are the Inputs to the Arduino?
  - What are the Outputs of the Arduino?



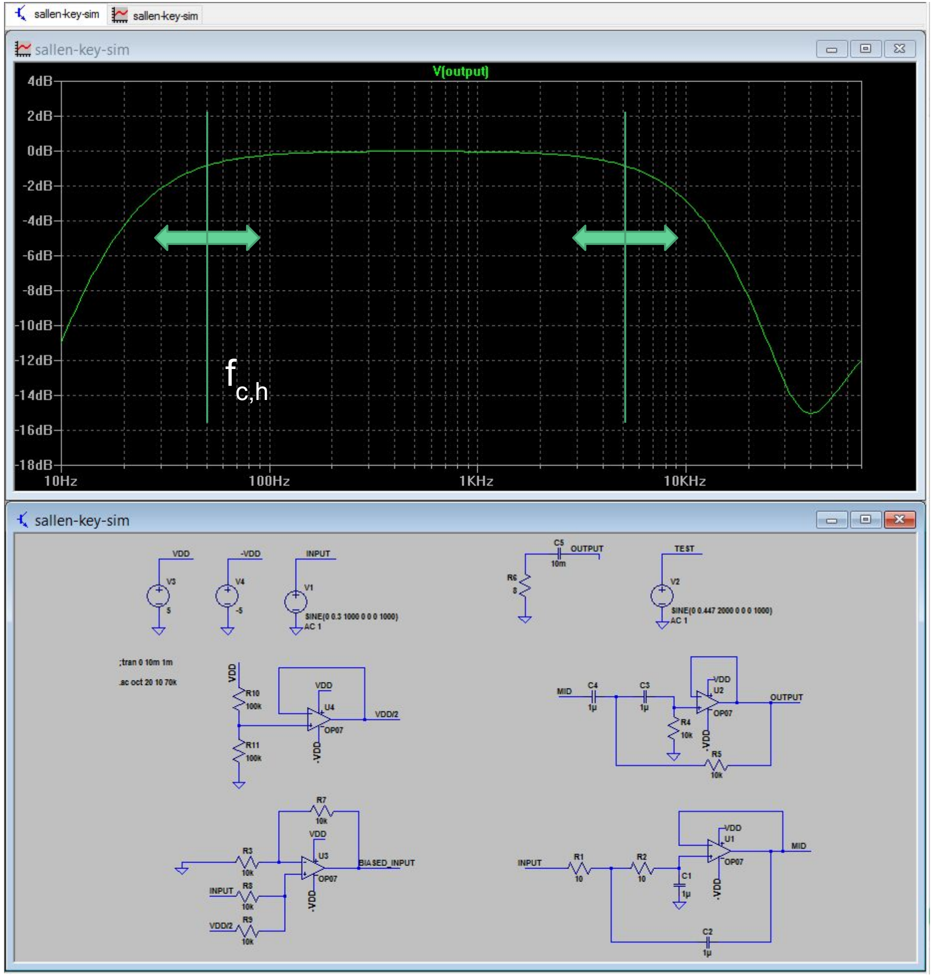
Shield Schematic Overview

- Potentiometers
- Digital Resistors
- Buttons
- Rail Inverter
- 2nd Order Filters



Analog Filtering

- Two Filters
  - One High Pass
  - One Low Pass
  - Back 2 Back
- Both ‘2nd Order'
  - ~12dB/decade
- Tunable Cutoff Frequencies



Digital vs Analog Potentiometers?

Digital Potentiometers

- Pros
  - Very, very high accuracy
  - Very, very high precision
- Cons
  - Have to be programmed
  - Low power rating

Analog Potentiometers

- Pros
  - Fun to use!
  - High power rating
- Cons
  - Low precision
  - Low accuracy

Our Goal:

Filter The Audio

Based On The

~~Position of the~~

~~Potentiometers.~~

The Button States.

Steps to Accomplish Goals:

1. Read The Button State
2. Adjust the Filter Cutoff Frequencies

Sub-Steps to Accomplish Goals:

1. Read The Buttons
   a. Read in as Digital Input.
2. Adjust the Filter Cutoff Frequency
   a. Learn relationship between digital resistors & our filters.
   b. Learn to talk to our filters.

Controlling the Filters

- Need to move filter cutoffs
- -> Need to Change Resistor Values
- Our Resistors are Microchip MCP4162s
  - Thanks Brady!
- So How Do We Learn How To Control These?

Interacting with Advanced Parts

How do we interact with Advanced Parts?

Read The Datasheet.

http://ww1.microchip.com/downloads/en/DeviceDoc/22059b.pdf

- This Datasheet is hard to digest:
    - For multiple parts, not all of which work the same way.
    - It’s long; 88 Pages
    - It includes a lot of information which you, as embedded software engineers don’t ‘need’ to know.
- Learning how to read Datasheets is a skill.
    - You will acquire this kind of skill in ECE 391, CS 431, and ECE 445.
- I did this for you.

MICROCHIP **MCP414X/416X/424X/426X**

7/8-Bit Single/Dual SPI Digital POT with Non-Volatile Memory

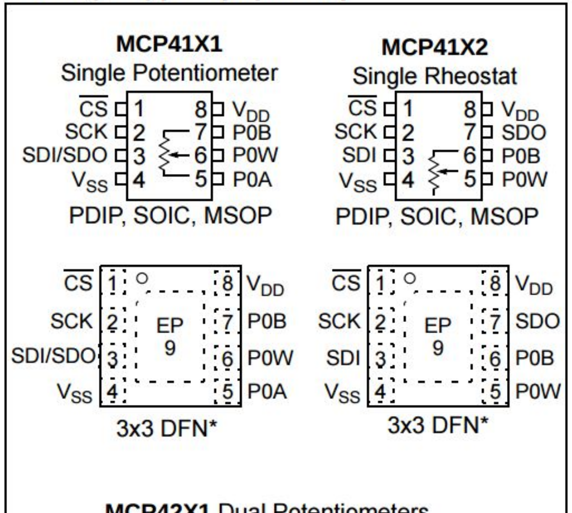
Features

- Single or Dual Resistor Network options
- Potentiometer or Rheostat configuration options
- Resistor Network Resolution
    - 7-bit: 128 Resistors (129 Steps)
    - 8-bit: 256 Resistors (257 Steps)
- R_{AB} Resistances options of:
    - 5 kΩ
    - 10 kΩ
    - 50 kΩ
    - 100 kΩ
- Zero-Scale to Full-Scale Wiper operation
- Low Wiper Resistance: 75Ω (typical)
- Low Tempco:
    - Absolute (Rheostat): 50 ppm typical (0°C to 70°C)
    - Ratiometric (Potentiometer): 15 ppm typical
- Non-volatile Memory
    - Automatic Recall of Saved Wiper Setting
    - WiperLock™ Technology
- SPI serial interface (10 MHz, modes 0,0 & 1,1)
    - High-Speed Read/Writes to wiper registers
    - Read/Write to Data EEPROM registers
    - Serially enabled EEPROM write protect
    - SDI/SDO multiplexing (MCP41X1 only)

Description

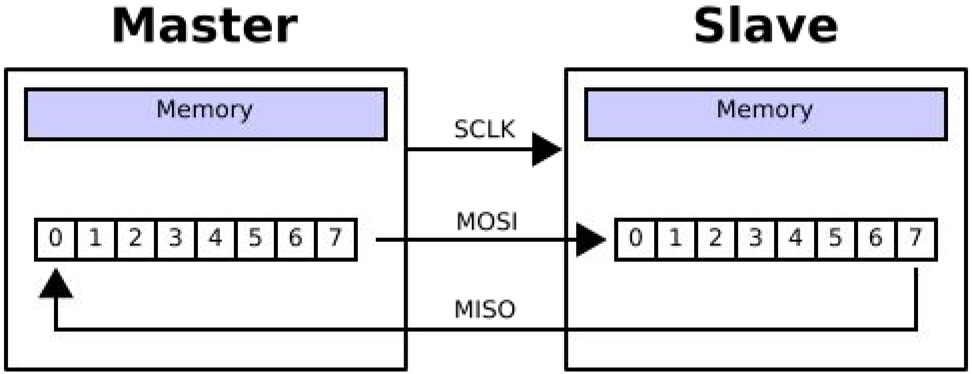
The MCP41XX and MCP42XX devices offer a wide range of product offerings using an SPI interface. WiperLock Technology allows application-specific calibration settings to be secured in the EEPROM.

Package Types (top view)



Talking to SPI Parts

- Master and Slave Devices
  - Can have multiple slaves on the same SCLK/MOSI/MISO lines.
  - Each slave has a chip select which we can use to tell it to listen.
- Kind of a loosely defined protocol.



- Reading the Datasheet:
  - MSB First

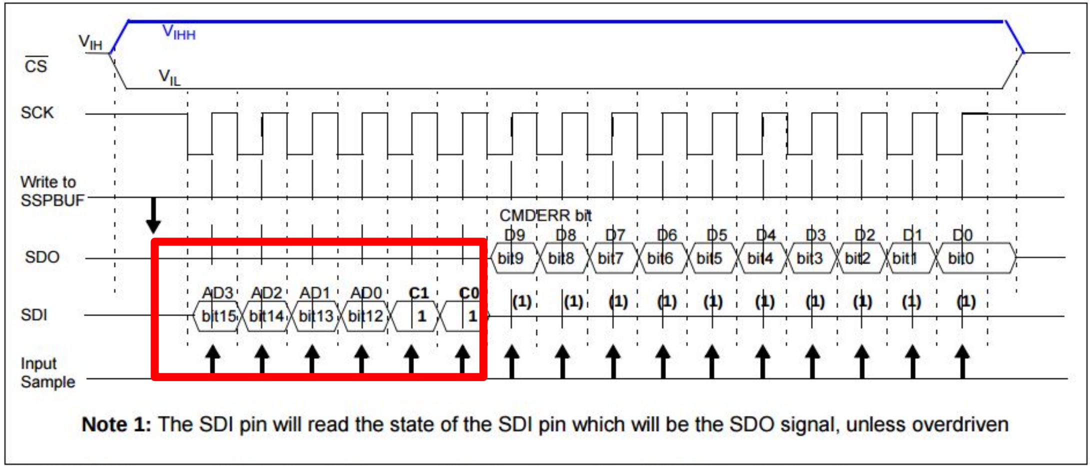


FIGURE 6-5: *16-Bit Read Command for Devices with SDI/SDO multiplexed - SPI Waveform (Mode 1,1).*

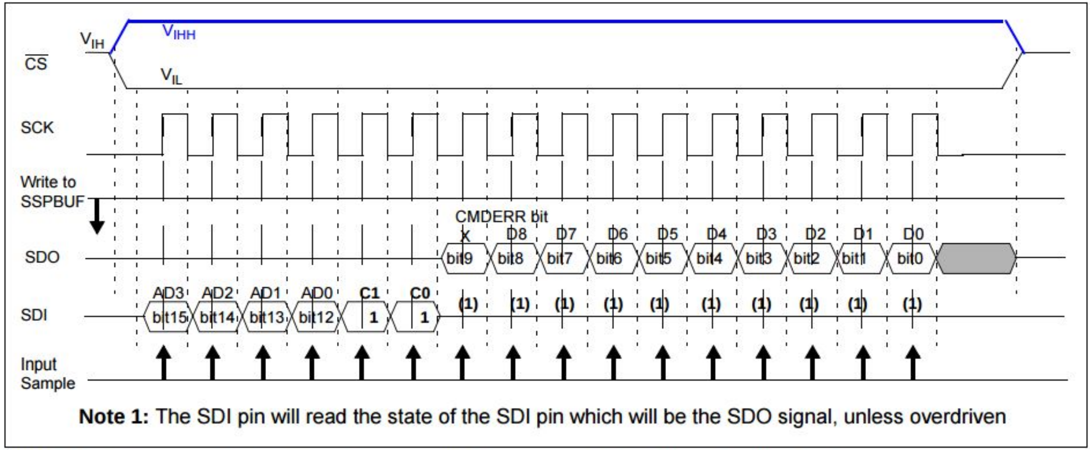


FIGURE 6-6: *16-Bit Read Command for Devices with SDI/SDO multiplexed - SPI Waveform (Mode 0,0).*

- Reading the Datasheet:
  - MSB First
  - Mode 11
  - 10MHz operation

TABLE 1-1: SPI REQUIREMENTS (MODE = 11)

#	Characteristic	Symbol	Min	Max	Units	Conditions
	SCK Input Frequency	F_{SCK}	—	10	MHz	V_{DD} = 2.7V to 5.5V
			—	1	MHz	V_{DD} = 1.8V to 2.7V
70	$\overline{CS}$ Active (V_{IL} or V_{IHH}) to SCK↑ input	TcsA2scH	60	—	ns	
71	SCK input high time	TscH	45	—	ns	V_{DD} = 2.7V to 5.5V
			500	—	ns	V_{DD} = 1.8V to 2.7V
72	SCK input low time	TscL	45	—	ns	V_{DD} = 2.7V to 5.5V
			500	—	ns	V_{DD} = 1.8V to 2.7V
73	Setup time of SDI input to SCK↑ edge	TDIV2scH	10	—	ns	
74	Hold time of SDI input from SCK↑ edge	TscH2DIL	20	—	ns	
77	$\overline{CS}$ Inactive (V_{IH}) to SDO output hi-impedance	TcsH2DOZ	—	50	ns	**Note 1**
80	SDO data output valid after SCK↓ edge	TscL2DOV	—	70	ns	V_{DD} = 2.7V to 5.5V
				170	ns	V_{DD} = 1.8V to 2.7V
83	$\overline{CS}$ Inactive (V_{IH}) after SCK↑ edge	TscH2csI	100	—	ns	V_{DD} = 2.7V to 5.5V
			1		ms	V_{DD} = 1.8V to 2.7V
84	Hold time of $\overline{CS}$ Inactive (V_{IH}) to $\overline{CS}$ Active (V_{IL} or V_{IHH})	TcsA2csI	50	—	ns	

Note 1: This specification by design.

- Reading the Datasheet:
  - MSB First
  - Mode 11
  - 10MHz operation
  - Write to Address 0x00
  - Send Command 00

TABLE 7-1: COMMAND BIT OVERVIEW

C1:C0 Bit States	Command	# of Bits	Operates on Volatile/ Non-Volatile memory
11	Read Data	16-Bits	Both
00	Write Data	16-Bits	Both
01	Increment [(1)]	8-Bits	Volatile Only
10	Decrement [(1)]	8-Bits	Volatile Only

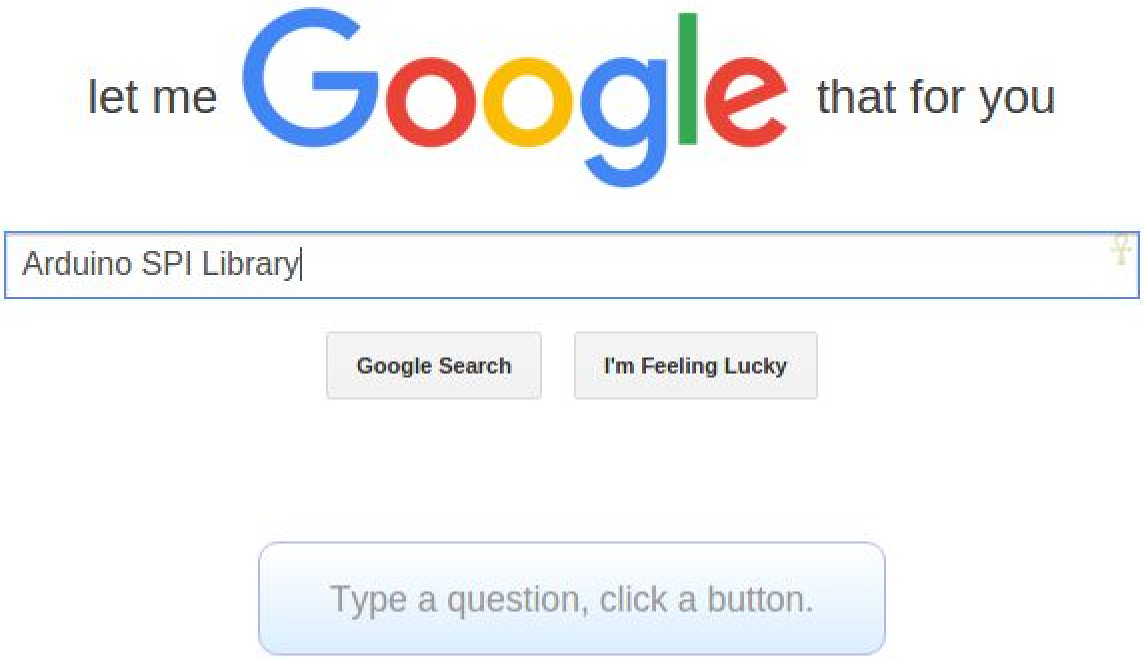
Note 1: High Voltage Increment and Decrement commands on select non-volatile memory locations enable/disable WiperLock Technology and the software Write Protect feature.

TABLE 4-1: MEMORY MAP

Address	Function	Memory Type
00h	Volatile Wiper 0	RAM
01h	Volatile Wiper 1	RAM
02h	Non-Volatile Wiper 0	EEPROM
03h	Non-Volatile Wiper 1	EEPROM
04h	Volatile TCON Register	RAM
05h	Status Register	RAM
06h	Data EEPROM	EEPROM
07h	Data EEPROM	EEPROM
08h	Data EEPROM	EEPROM
09h	Data EEPROM	EEPROM
0Ah	Data EEPROM	EEPROM
0Bh	Data EEPROM	EEPROM
0Ch	Data EEPROM	EEPROM
0Dh	Data EEPROM	EEPROM
0Eh	Data EEPROM	EEPROM
0Fh	Data EEPROM	EEPROM

So now we
implement SPI, right?

Rule of Software: Don’t Reinvent the Wheel



Arduino SPI Library

- Standard, shipping library
- Good documentation by Arduino Standards

https://www.arduino.cc/en/Reference/SPI

Reference Language | Libraries | Comparison | Changes

SPI library

This library allows you to communicate with SPI devices, with the Arduino as the master device.

A Brief Introduction to the Serial Peripheral Interface (SPI)

Serial Peripheral Interface (SPI) is a synchronous serial data protocol used by microcontrollers for communicating with one or more peripheral devices quickly over short distances. It can also be used for communication between two microcontrollers.

With an SPI connection there is always one master device (usually a microcontroller) which controls the peripheral devices. Typically there are three lines common to all the devices:

- **MISO** (Master In Slave Out) - The Slave line for sending data to the master,
- **MOSI** (Master Out Slave In) - The Master line for sending data to the peripherals,
- **SCK** (Serial Clock) - The clock pulses which synchronize data transmission generated by the master

and one line specific for every device:

- **SS** (Slave Select) - the pin on each device that the master can use to enable and disable specific devices.

Functions

- SPISettings
- begin()
- end()
- beginTransaction()
- endTransaction()
- setBitOrder()
- setClockDivider()
- setDataMode()
- transfer()
- usingInterrupt()
- Due Extended SPI usage

See also

- shiftOut()
- shiftIn()

Aside: Device Communication Standards

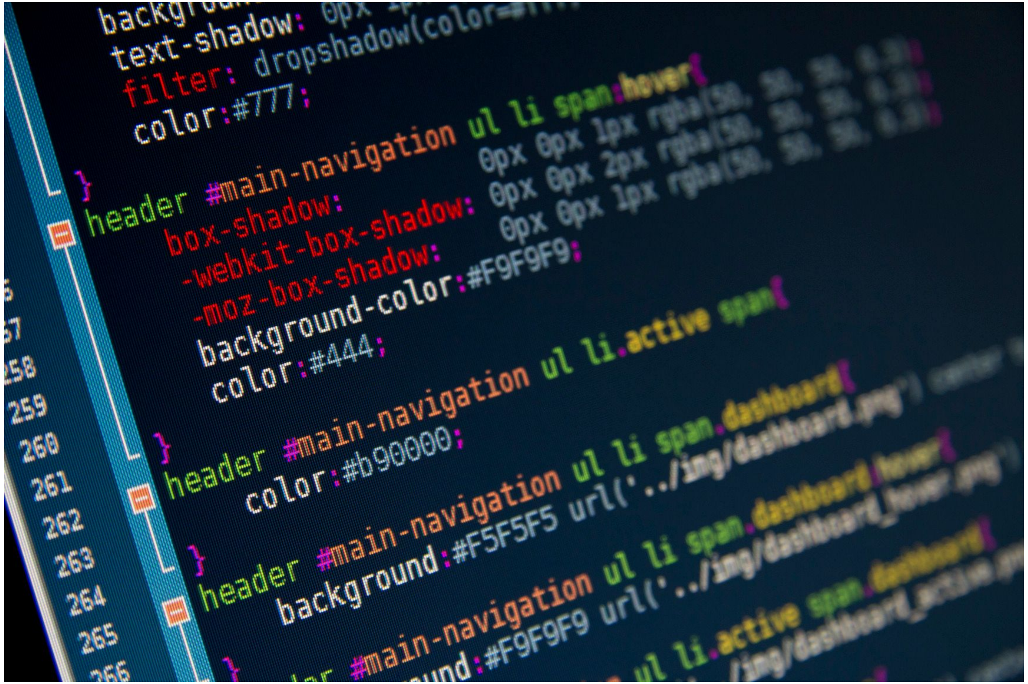
- There are a lot of them.
- You generally should not invent your own communication protocol.
- You generally should not have to write your own communication library.

- SPI
- SSI
- CANN
- I2C
- I2S
- the list goes on and on and on and on

Writing Good Software

What Is Good Software?

1. Works
2. Readable
3. As Simple as Possible
4. Modular
5. Extensible



CSS is almost never good code.

Why Do We Write Good Software?

1. Money
2. Pride
3. Kindness to Ourselves & Others

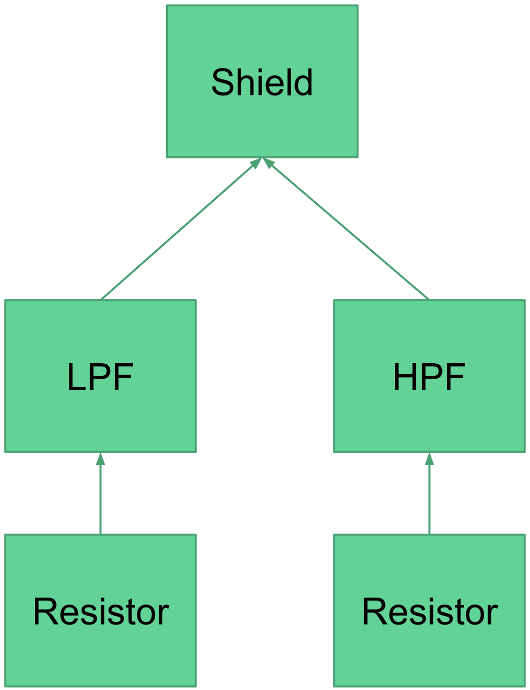
Rule of Software: Adhere to A Style

https://www.arduino.cc/en/Reference/APIStyleGuide

Arduino: Not C++

Arduino Uses OOP

- Object Oriented Programming
- Every Physical Thing should also be a Software Thing
- You'll learn about it in CS 225



Let’s Talk About the Code

Appendix + Errata:
Using the Potentiometers

The PCB Shields have a small error which we didn't have time to fix.

Correctly Attaching The Pots

Ironically, all the “hard” parts of the board work fine.

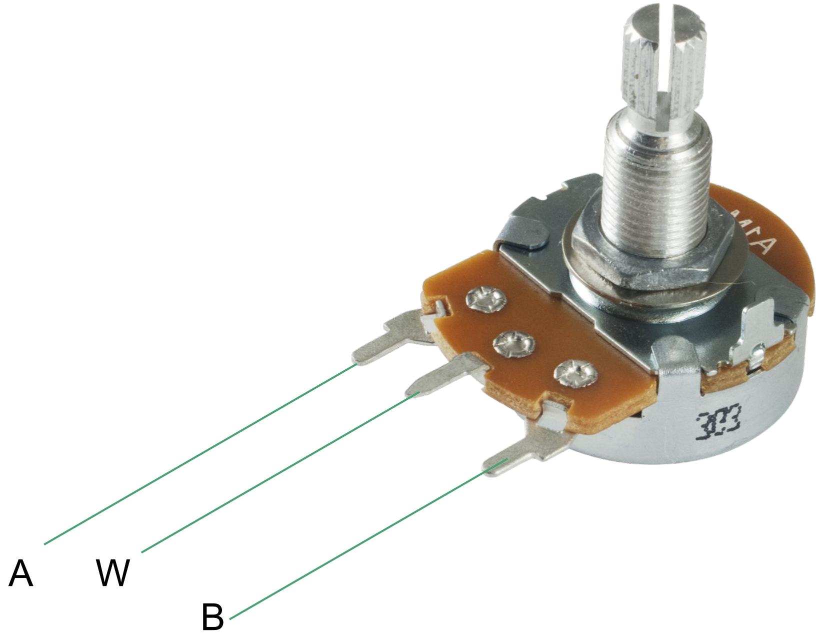
It’s the easiest part that got messed up

I mislabeled the “W” and “B” pins

The fix is simple, just rewire manually

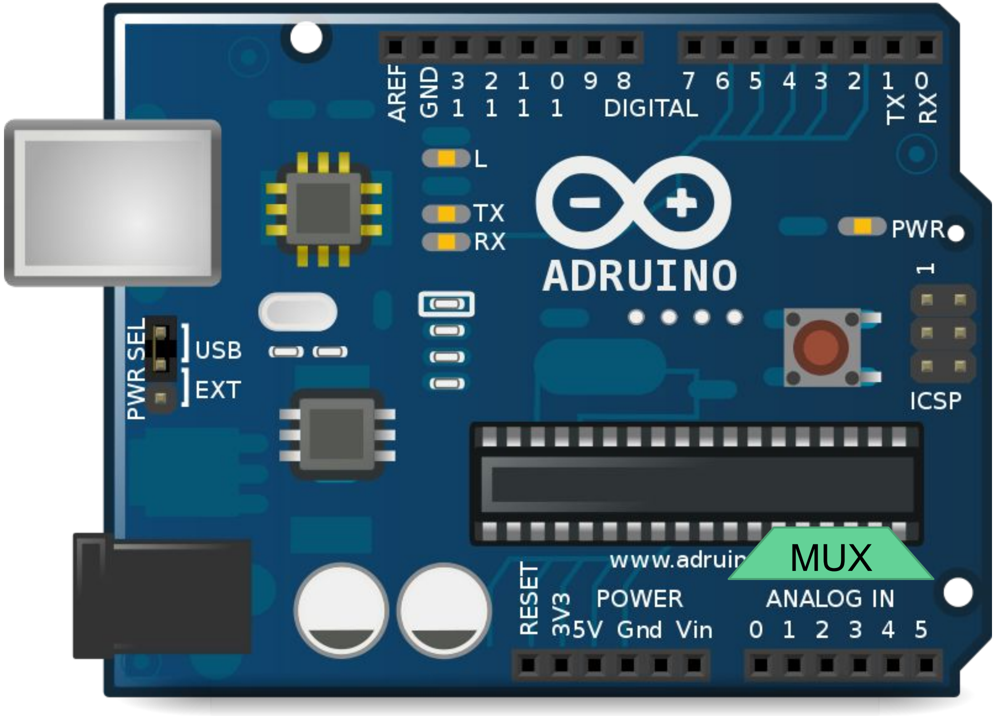
Simple for one board, obnoxious for fixing 30 of them

Rework instructions online by end of today



Arduino ADC

- 10 Bit
  - Values Range from 0 to 2^10 - 1
  - Can be converted to voltage using math.
- Despite # of Analog Pins, only one ADC onboard
  - Don’t sample too fast; avoid noise.
- `int analogRead(pin){};`



Reading a Potentiometer

- The potentiometers on your shields are/were on pins A2, A3
- We'll adapt a sketch to learn how to do this.

```
/* Analog Read to LED
 * ------------------
 *
 * turns on and off a light emitting diode(LED) connected to digital
 * pin 13. The amount of time the LED will be on and off depends on
 * the value obtained by analogRead(). In the easiest case we connect
 * a potentiometer to analog pin 2.
 *
 * Created 1 December 2005
 * copyleft 2005 DojoDave <http://www.0j0.org>
 * http://arduino.berlios.de
 *
 */

int potPin = 2;    // select the input pin for the potentiometer
int ledPin = 13;   // select the pin for the LED
int val = 0;       // variable to store the value coming from the sensor

void setup() {
  pinMode(ledPin, OUTPUT);  // declare the ledPin as an OUTPUT
}

void loop() {
  val = analogRead(potPin);    // read the value from the sensor
  digitalWrite(ledPin, HIGH); // turn the ledPin on
  delay(val);                  // stop the program for some time
  digitalWrite(ledPin, LOW);   // turn the ledPin off
  delay(val);                  // stop the program for some time
}
```

arduino.cc/en/Tutorial/Potentiometer