Arduino: Beyond the Blink

Agenda

- Introductions
- Arduino Basics
- Getting into the IDE
- Blink!
- Arduino Peripherals
- Pitfalls
- Embedded Programming in a Nutshell
- Going Beyond Arduino

What is a Microcontroller?

??????????????????

What is a Microcontroller?

- It's a tiny computer on a Chip
  - Arduino has: 32 kb Flash, 8 kb RAM
  - Words are only 8 bits
- No operating system!

Binary Math Example

```
  0b00100010
+ 0b00010110
------------
  0b00111000
```

```
  0b0011001 00011000
+ 0b0100110 11110000
--------------------
  SO MANY BITS
  1
```

```
  0b0011001
+ 0b0100110
-----------
  0b1000000
```

```
  0b00011000
+ 0b11110000
------------
  0b00001000
  1
```

0b10000000 00001000

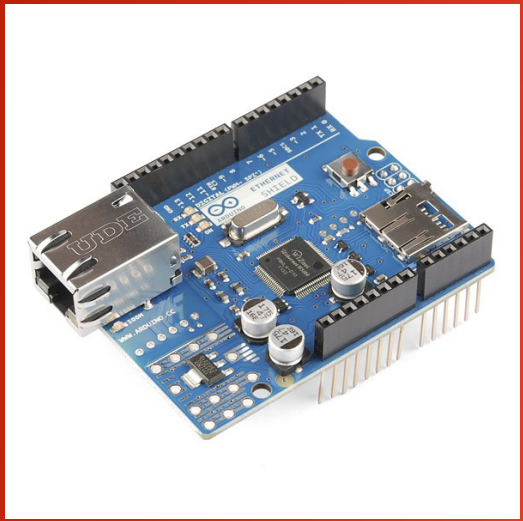
What can a Microcontroller do?

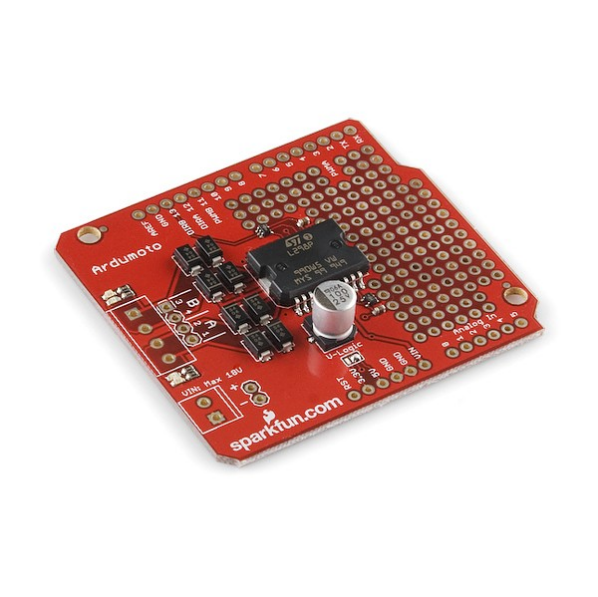
What Shouldn't a Microcontroller do?

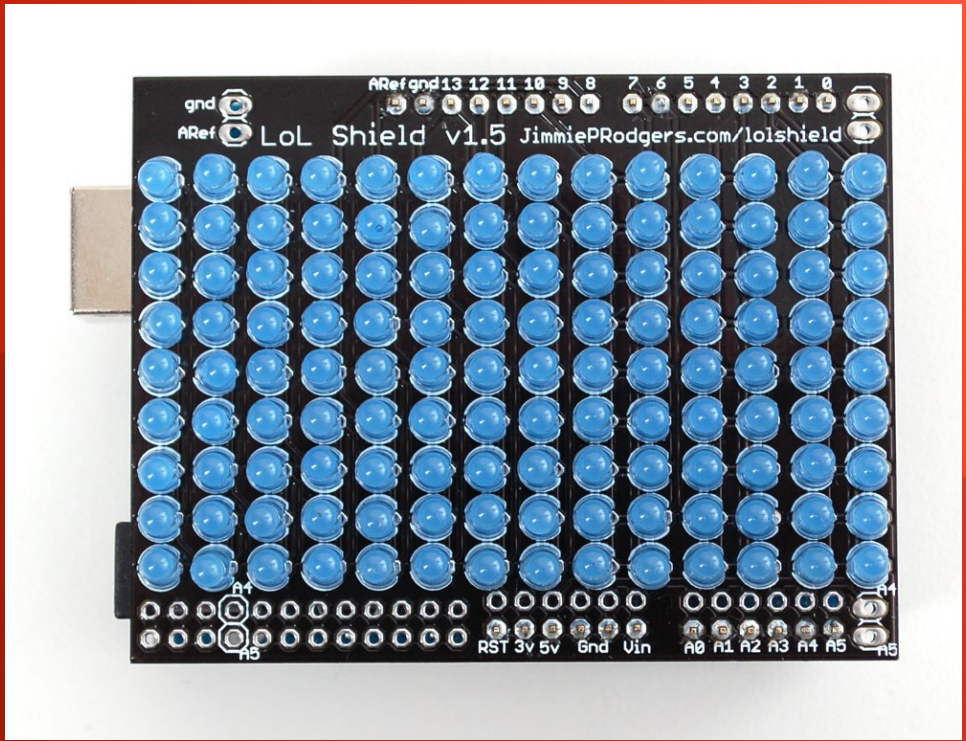
Training Wheels: Arduino Edition

- Handy USB output
  - No more pesky serial ports!
- Voltage Regulation
- Standard Daughterboard form factor

Shields!



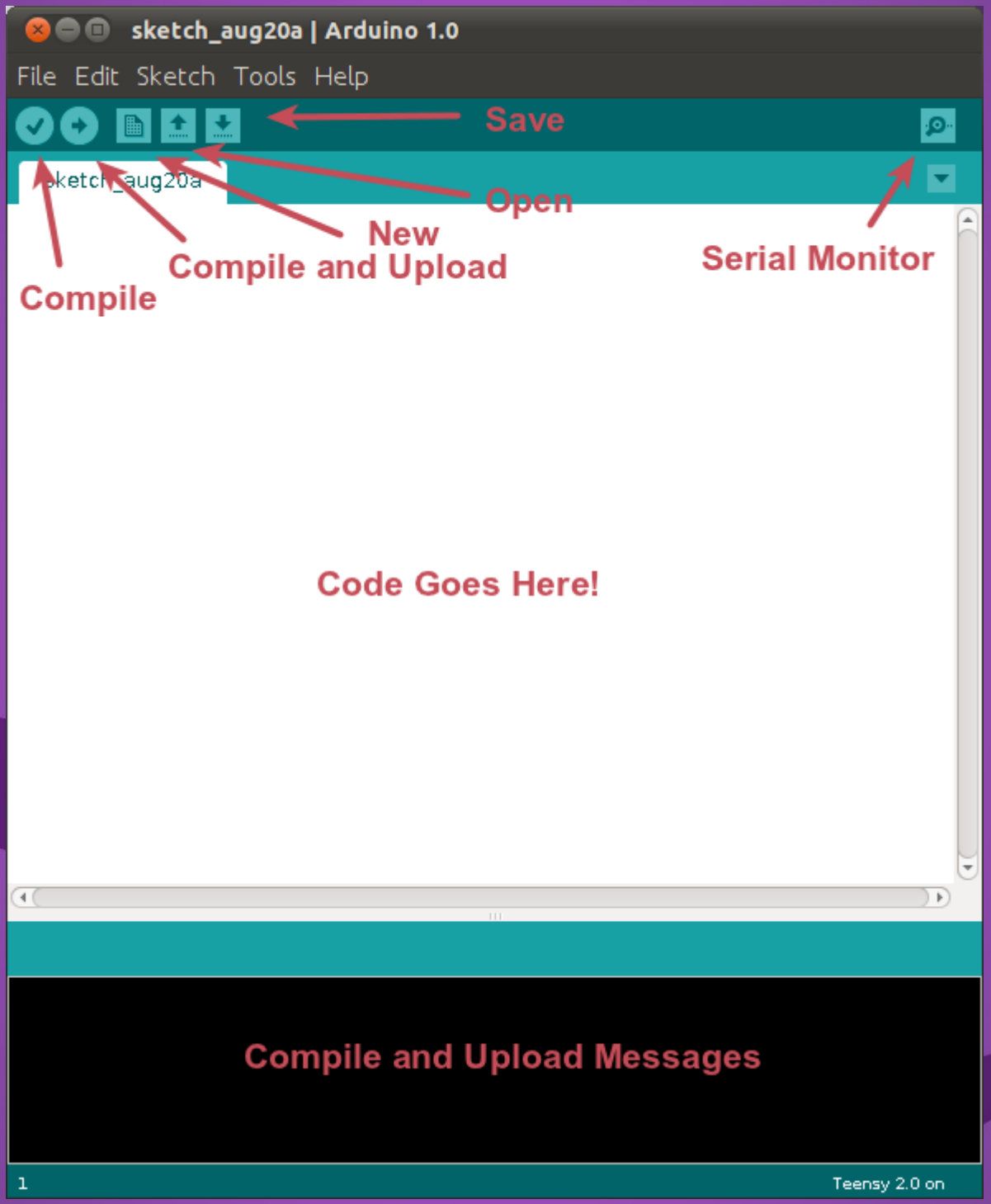

ArduMoto
sparkfun.com

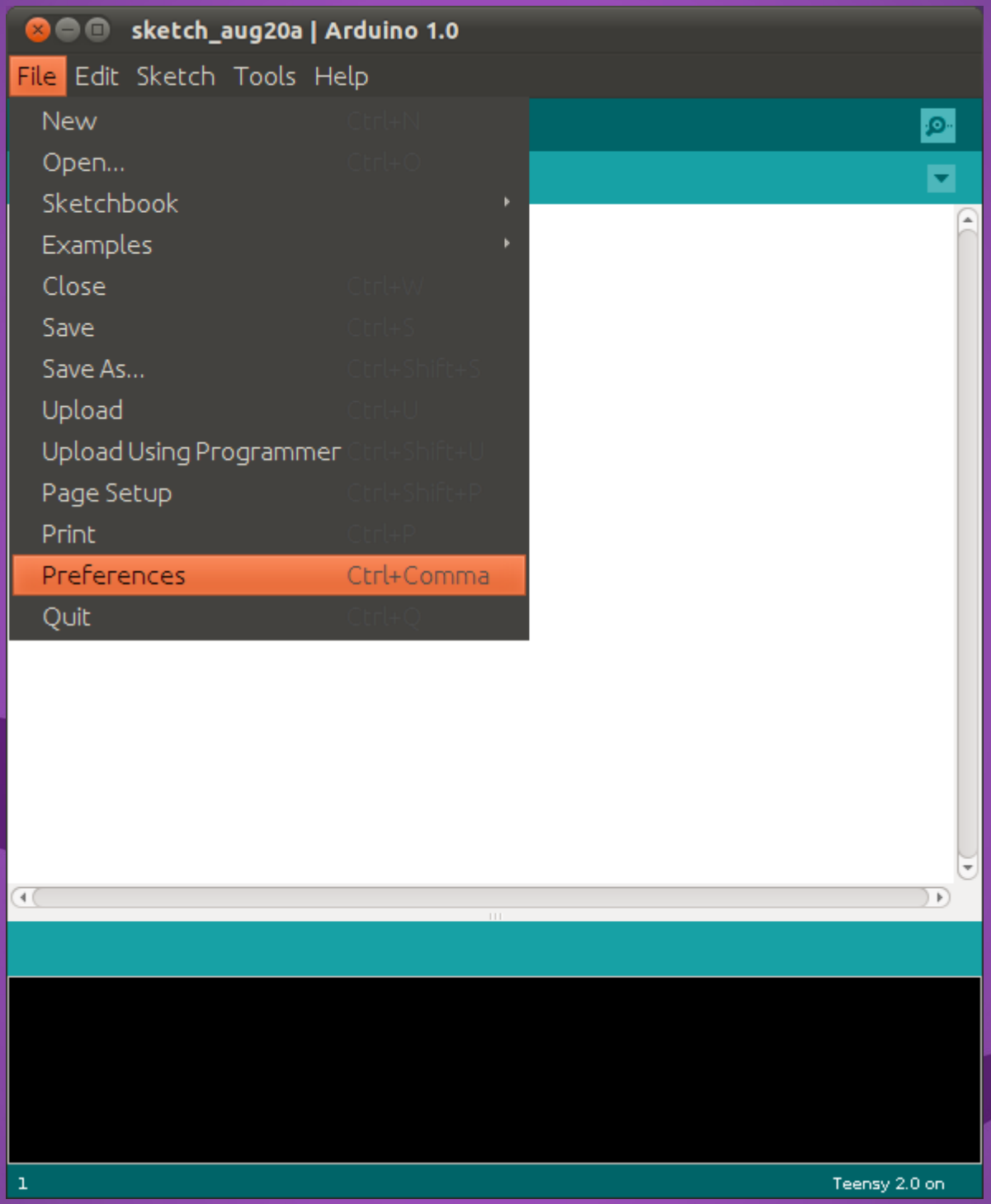

gnd
ARef
ARef gnd 13 12 11 10 9 8 7 6 5 4 3 2 1 0
LoL Shield v1.5 JimmiePRodgers.com/lolshield
A1
A5
RST 3v 5v Gnd Vin
A0 A1 A2 A3 A4 A5

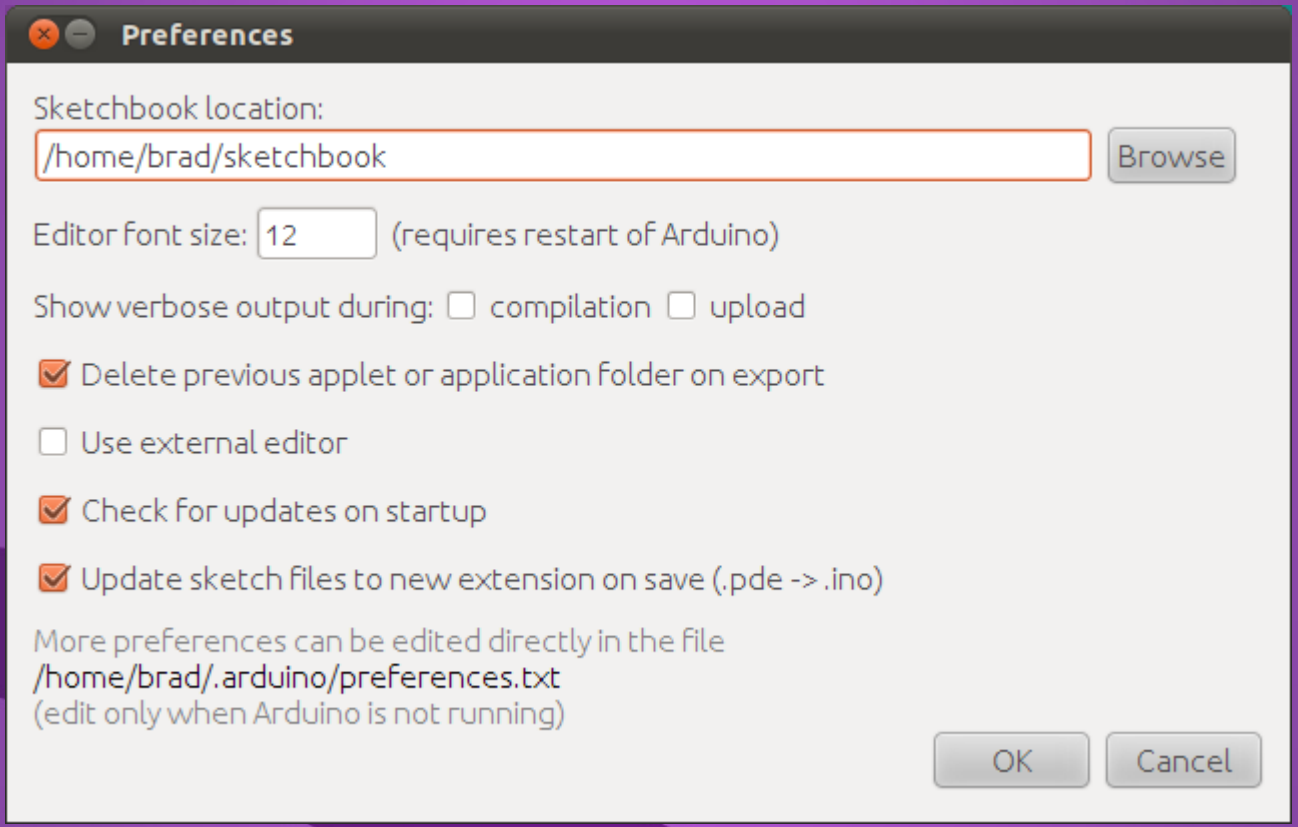
Libraries

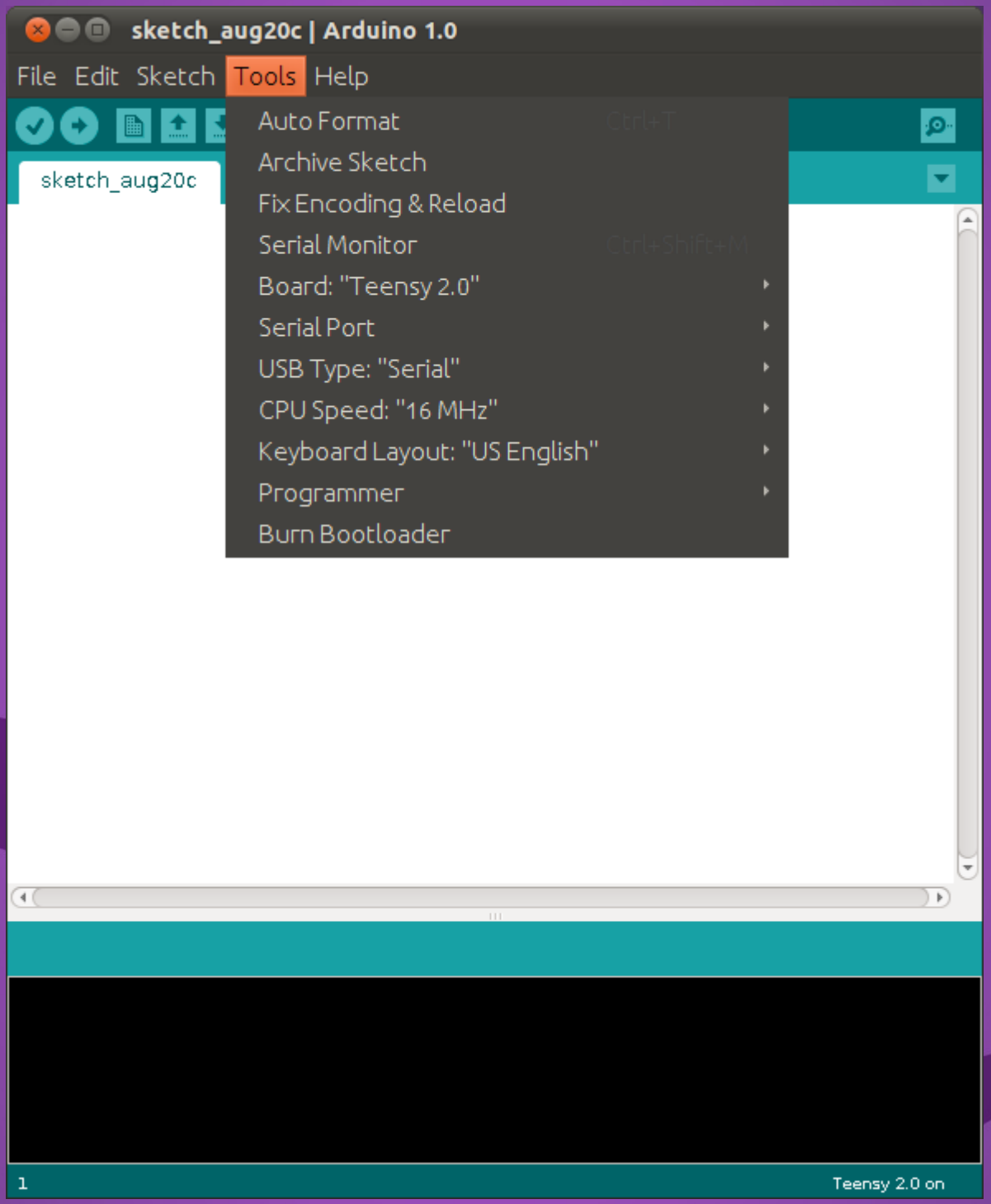
Code you don't have to write

Getting into the IDE


sketch_aug20a | Arduino 1.0
File Edit Sketch Tools Help
Save
Open
New
Compile and Upload
Compile
Serial Monitor
Code Goes Here!
Compile and Upload Messages
1
Teensy 2.0 on


sketch_aug20a | Arduino 1.0
File
Edit
Sketch
Tools
Help
New Ctrl+N
Open... Ctrl+O
Sketchbook
Examples
Close Ctrl+W
Save Ctrl+S
Save As... Ctrl+Shift+S
Upload Ctrl+U
Upload Using Programmer Ctrl+Shift+U
Page Setup Ctrl+Shift+P
Print Ctrl+P
Preferences Ctrl+Comma
Quit Ctrl+Q
1
Teensy 2.0 on


Preferences
Sketchbook location:
/home/brad/sketchbook
Browse
Editor font size: 12 (requires restart of Arduino)
Show verbose output during: compilation upload
Delete previous applet or application folder on export
Use external editor
Check for updates on startup
Update sketch files to new extension on save (.pde -> .ino)
More preferences can be edited directly in the file
/home/brad/.arduino/preferences.txt
(edit only when Arduino is not running)
OK
Cancel


sketch_aug20c | Arduino 1.0
File
Edit
Sketch
Tools
Help
sketch_aug20c
Auto Format
Ctrl+T
Archive Sketch
Fix Encoding & Reload
Serial Monitor
Ctrl+Shift+M
Board: "Teensy 2.0"
Serial Port
USB Type: "Serial"
CPU Speed: "16 MHz"
Keyboard Layout: "US English"
Programmer
Burn Bootloader
1
Teensy 2.0 on

About the IDE

- What Language?
  - C++, kinda. C, really.
- Advantages?
  - No Makefiles!
  - Automatic Dependency Gathering
  - Upload code quick and easy
  - Easy access to libraries

Basic C

- Casting and Types
- Bitwise Operations
- Macros
- Comments
- Semicolon

To move beyond the blink, you must first learn to blink.

Arduino Zen Master

Blink | Arduino 1.0

File Edit Sketch Tools Help

Blink §

```
/*
  Blink
  Turns on an LED on for one second, then off for one second, repeatedly.

  This example code is in the public domain.
 */

void setup() {
  // initialize the digital pin as an output.
  // Pin 13: Arduino has an LED connected on pin 13
  pinMode(13, OUTPUT);
}

void loop() {
  digitalWrite(13, HIGH);   // set the LED on
  delay(1000);              // wait for a second
  digitalWrite(13, LOW);    // set the LED off
  delay(1000);              // wait for a second
}
```

18 Teensy 2.0 on

Peripherals

Peripherals

- GPIO
- Timers
- I2C
- SPI
- Serial
- ADC
- PWM

GPIO

- Can use all pins as GPIO
- Operate as switch (high or low), 0 or 5 volts
- Must declare as output or input
- Inputs can be pulled up
- Can only output 20mA per pin
- Can input much more (open drain)

GPIO

- PinMode(Pin, Mode)
- DigitalRead(Pin)
- DigitalWrite(Pin, Value)

Timers

- Predictably, used to time things
- Configurable Counters
- Useful for:
  - Counting Events
  - Timing Events
  - PWM
  - Occasionally activating something
  - Waiting

Timers

- Delay(Milliseconds)
- DelayMicroseconds(Microseconds)
- Millis()
- Micros()
- AnalogWrite(pin, value)

Measuring Time: Loop-de-loop

- Millis(), Micros(), Delay() and DelayMicroseconds() can all loop!
- If you're trying to measure time, just use the Time library!
- Otherwise, cast to int64_t, if your number suddenly goes lower than it was before, you've looped over!

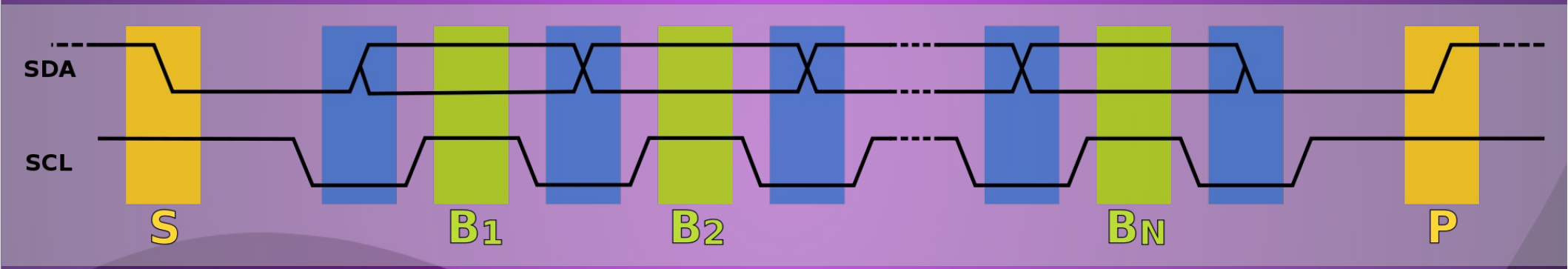
Timer Libraries

- Timer1 uses Timer1 instead of Timer0
  - More PWM control, doesn't interfere with time operations
- Timer
  - Modify PWM, basic scheduling operations, more time functions

I2C

- Digital Communications
- 2 Wires
- Limited to about 3 feet in distance
- 400 kHz Bit Rate
- Tons of Devices
- Addressable! Up to 127 devices on bus!

I2C



I2C

<START>+

<7 bit Address>+

<R/W bit>+

<DATA>+

<STOP>

Wiring Library

- Is terrible
- Wire.begin()
- Wire.beginTransmission(7 bit addr)
- Wire.write(data)
- Wire.endTransmission()

Wiring Library

- Wire.requestFrom(addr)
- Wire.available()
- Wire.read()

Wiring Library

- Wire.begin(addr)
- Wire.onReceive(function)
- Wire.onRequest(function)

SPI

- More serial data transfer!
- 3 Lines (minimum)
  - Clock
  - Data In
  - Data Out
  - (Chip Select)
  - (Reset)
  - (IRQ)
- Not individually addressable (well, chip select)

SPI

- Data read at the same time it is transmitted
- Devices are extremely diverse!
- Different messages, sometimes different endianness
- Will need to write custom stuff for each device
  - Unless there's a library!
- Arduino is always a master device
- Arduino provides SPI access primitives

SPI

- SPI.begin()
- SPI.end()
- SPI.setBitOrder(LSBFIRST/MSBFIRST)
- SPI.setClockDivider(SPI_CLOCK_DIV2/4/8/16/32/64/128)
- SPI.setDataMode(SPI_MODE0/1/2/3)
- SPI.transfer(byte)

Serial

- Arduino has one
- You use it for USB
- Not all bauds are created equally
- Serial.print(string)
- Serial.println(string)

Arduino Baud Rates

http://www.wormfood.net/avrbaudcalc.php

16 Mhz		
Bit Rate	**UBRR**	**% of error**
300	3332	0.0
600	1666	0.0
1200	832	0.0
2400	416	0.1
4800	207	0.2
9600	103	0.2
14400	68	0.6
19200	51	0.2
28800	34	0.8
38400	25	0.2
57600	16	2.1
76800	12	0.2
115200	8	3.7
230400	3	7.8

New Soft Serial

- Not great, but the only option for bonus serial ports
- Uses a lot of CPU!
- Don't interrupt it!
- Uses timers
- Same interface as Serial

Analog to Digital

- analogRead(Pin)
- Involves math to convert to voltages
- Watch your battery!
- Don't read too fast!

Example ADC Math

- Aref = 5v
- 1023 = 5v, 0 = 0v
- X/1024 * 5 = input voltage

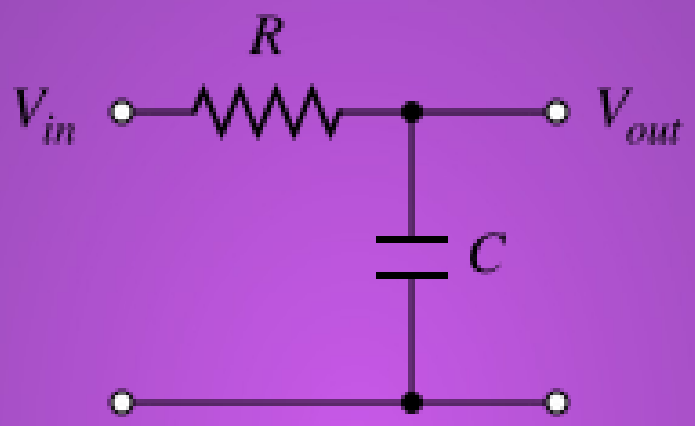
PWM

- Analog out
  - But not really!
- Shares timer with millis and micros
- Star next to PWM pins
- Use Timer1 library for better control!

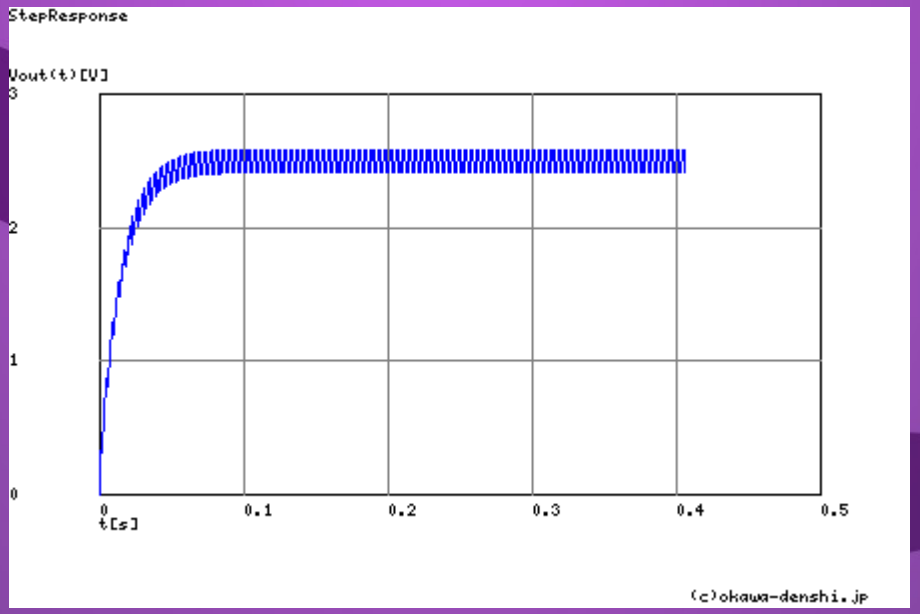
“Analog“ out

- PWM pulses very rapidly
  - Arduino at ~900kHz
- Duty cycle varies from 0-100%
- Fine for dimming LEDs, but what about real analog out?

Low-Pass Filter



R = 15kOhms, C = 2.2uf



Interrupts

- Interrupt your program when something important happens
- Allows the Arduino to react quickly to external stimuli
- Can be Internal or External

Internal Interrupts

- Called when something happens to a peripheral
- Generally, libraries take care of this for you!
- Example: Wiring Library

External Interrupts

- Generic, outside your Arduino
- INT0 = D2, INT1 = D3
- attachInterrupt(pin, function, mode)
- Mode = LOW, CHANGE, RISING, FALLING
- Function = void doSomething()

External Interrupts

- Pin Change interrupts
- Operate on banks of pins
- More complicated
- If you really need more interrupts, use the third-party pinchangint library

Arduino Problems

- Arduino 1.0 incompatibilities
- Automatic Dependency Generation
- Adding Libraries
- Crummy Libraries
- Failure Conditions
- Walled Gardens
- Code Space

Basic Embedded C

- Structure
- Println Debugging
- Interrupts
- Anatomy of a Function Call
- Volatile
- Structuring your code
- Libraries
- Other Tricks

Structure of a Program

Blink | Arduino 1.0

File Edit Sketch Tools Help

Blink

```
/*
  Blink
  Turns on an LED on for one second, then off for one second, repeatedly.

  This example code is in the public domain.
 */

void setup() {
  // initialize the digital pin as an output.
  // Pin 13: Arduino has an LED connected on pin 13
  // Pin 11: Teensy 2.0 has the LED on pin 11
  // Pin 6: Teensy++ 2.0 has the LED on pin 6
  pinMode(13, OUTPUT);
}

void loop() {
  digitalWrite(13, HIGH);   // set the LED on
  delay(1000);              // wait for a second
  digitalWrite(13, LOW);    // set the LED off
  delay(1000);              // wait for a second
}
```

1 Teensy 2.0 on

Println Debugging

Blink | Arduino 1.0

File Edit Sketch Tools Help

Blink §

```
/*
  Blink
  Turns on an LED on for one second, then off for one second, repeatedly.

  This example code is in the public domain.
 */

void setup() {
  // initialize the digital pin as an output.
  // Pin 13: Arduino has an LED connected on pin 13
  // Pin 11: Teensy 2.0 has the LED on pin 11
  // Pin 6: Teensy++ 2.0 has the LED on pin 6
  pinMode(13, OUTPUT);
  Serial.begin(9600);
  Serial.println("Pins and Serial Initialized");
}

void loop() {
  digitalWrite(13, HIGH);   // set the LED on
  Serial.println("LED ON");
  delay(1000);              // wait for a second
  digitalWrite(13, LOW);    // set the LED off
  Serial.println("LED OFF");
  delay(1000);              // wait for a second
}
```

Println Debugging!

13 Teensy 2.0 on

Interrupts

- Keep these short!
- Don't allow other interrupts to trigger!
- Don't call a lot of other functions!

Function Calls

- Don't recurse!
- Stack Overflow

Volatile

- Alters the way compilers detect changes in variable
- Use it when variables aren't changing when they should

Globals

- Avoid them
- Pass around state variable instead

Switch-Case

- Multiple choices?
- Use me instead of if-else-if chains!

Beyond Arduino

- Arduino + Pure C = Beyond!
- AVR – Arduinowheel = Beyond!
- Other Microcontrollers = Beyond!

Arduino + Pure C

- AVR studio or AVR-GCC are both great!
- AVR LIBC is well documented!
- Tons of libraries
- AVRFreaks.net
- AVRDude

AVR - Arduinowheels

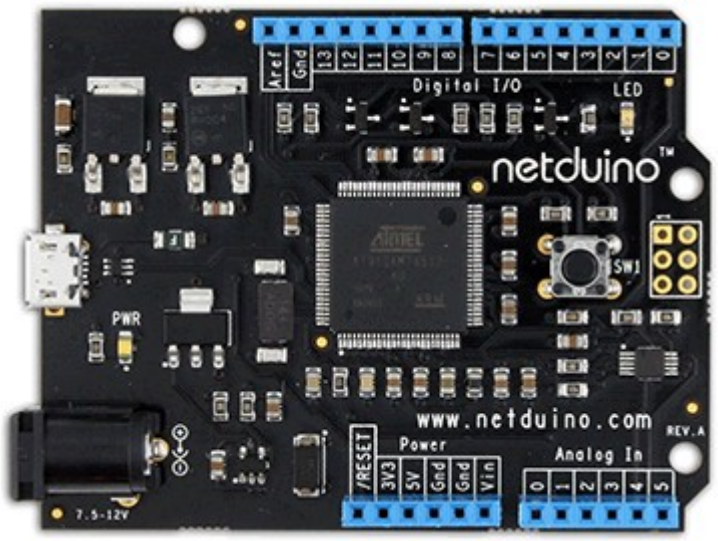
- Just need an Atmega328, voltage supply, and an AVR programmers
- Programmers start at $20
- A nice programmer is $50

Other Microcontrollers

- Tons of choices!
- All the same principles apply!

NetDuino

- Program in C#
- Super easy!
- Lots of code space!
- Not as great for robots



LeafLabs Maple

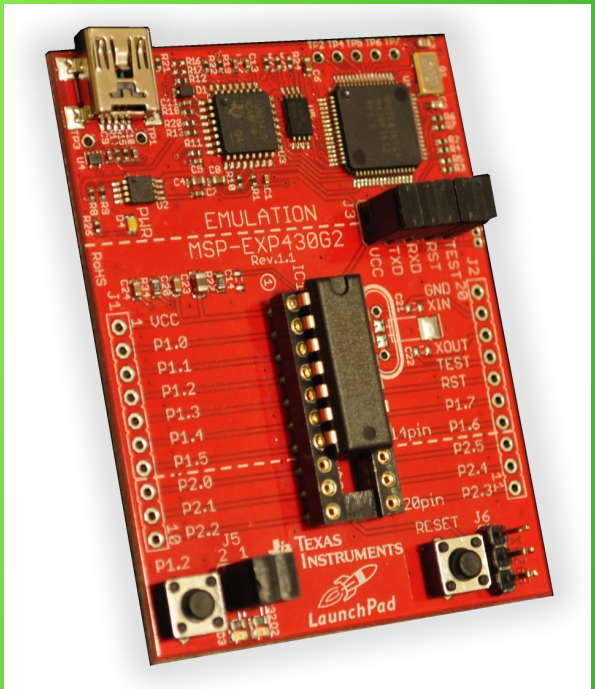
- ARM Chip
- Arduino-like environment
- FAST!
- More IO!

PICs

- Before Arduino, these were **the** hobbyist chip
- Cheap!
- There's a million of them!
- Tons of libraries and support

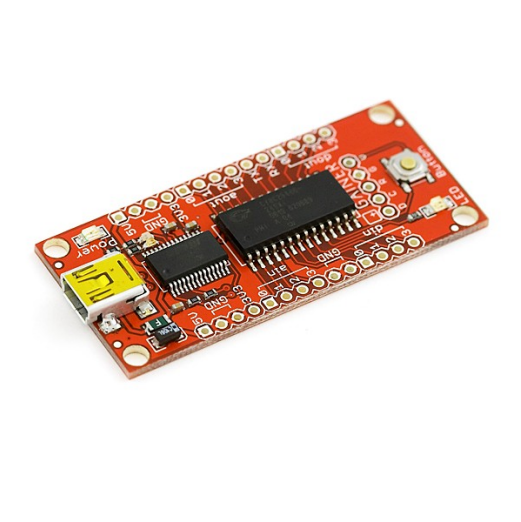
MSP430

- Cheap dev boards
- Super, super low power
- Reasonable community
- Flexible peripheral set



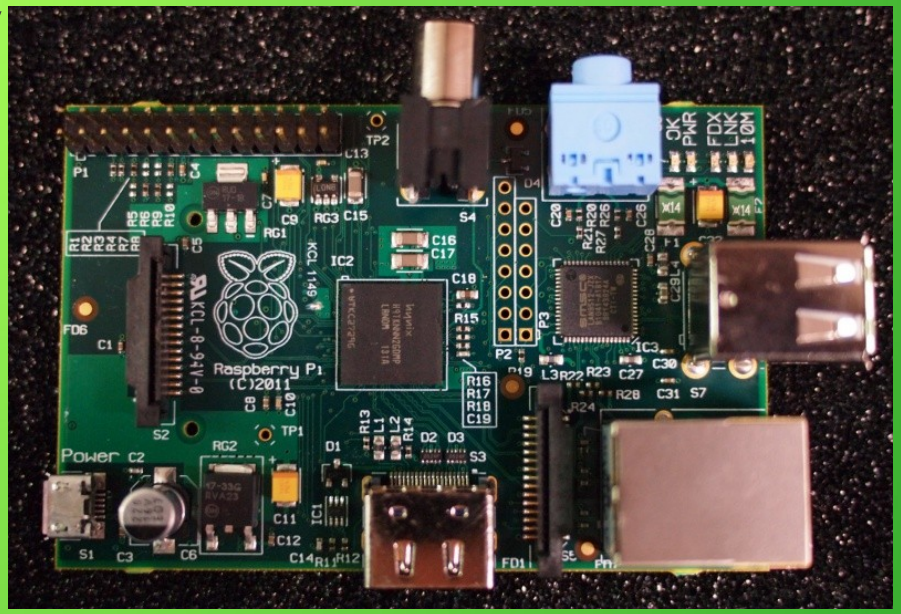
PSoC

- Mixed Signal Microcontroller
- Difficult to develop for
- It is a mystery



RasPi

- Cheap!
- Linux!
- Not so good for robots
- Closed and Proprietary



Other ARM

- Seriously, more than I could ever describe
- If you're to this point, you're definitely “Beyond“

END OF LINE

Arduino: Beyond the Blink

Agenda

- Introductions
- Arduino Basics
- Getting into the IDE
- Blink!
- Arduino Peripherals
- Pitfalls
- Embedded Programming in a Nutshell
- Going Beyond Arduino

Here are the topics we're going to cover.

•Class will be about 4 hours, try to take a decent break every hour. Get up at any time, do what you need to, you're not in school.

•Feel free to stay after. Feel free to ask questions whenever.

•What experience does everyone else already have with Arduino?

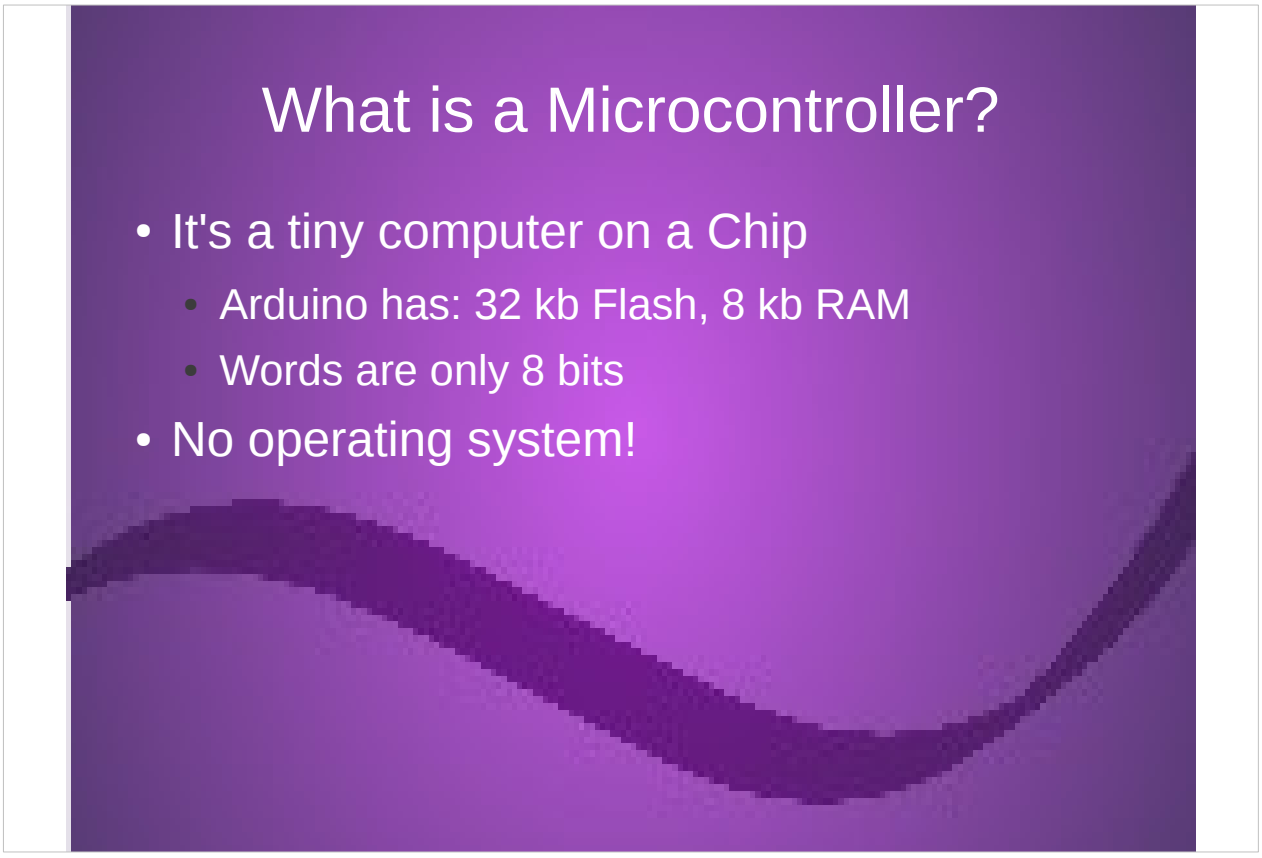
•My experience is this:

- M. Eng thesis on developing secure RTU
- Lead engineer, balloon program
- Other workshops
- Honeywell
- Other events

•Going to try something new

- Take sticky notes, write down project ideas!
- Any project idea!
- They can be completely insane
- They don't have to be things you actually want to do.
- These will be anonymous
- LVL1 is an engineer collective, so we're going to make use of it!
- Please collaborate on project ideas!
- The more, the better!

Can anyone describe what a microcontroller is? Anyone at all!



A microcontroller is a very tiny computer on a chip. The Arduino has only 32 kilobytes of flash, and 8 kilobytes of RAM.

- Flash used for program, RAM for variables
- 8 bit words
  - Largest number can natively represent is 255
  - Handle larger numbers with a Carry Bit
- One program only!
- Direct access to IO
- Very tight control loops

Binary Math Example

```
  0b00100010
+ 0b00010110
------------
  0b00111000
```

```
  0b0011001 00011000
+ 0b0100110 11110000
--------------------
  SO MANY BITS
  1
```

```
  0b0011001
+ 0b0100110
-----------
  0b1000000
```

```
  0b00011000
+ 0b11110000
------------
  0b00001000
  1
```

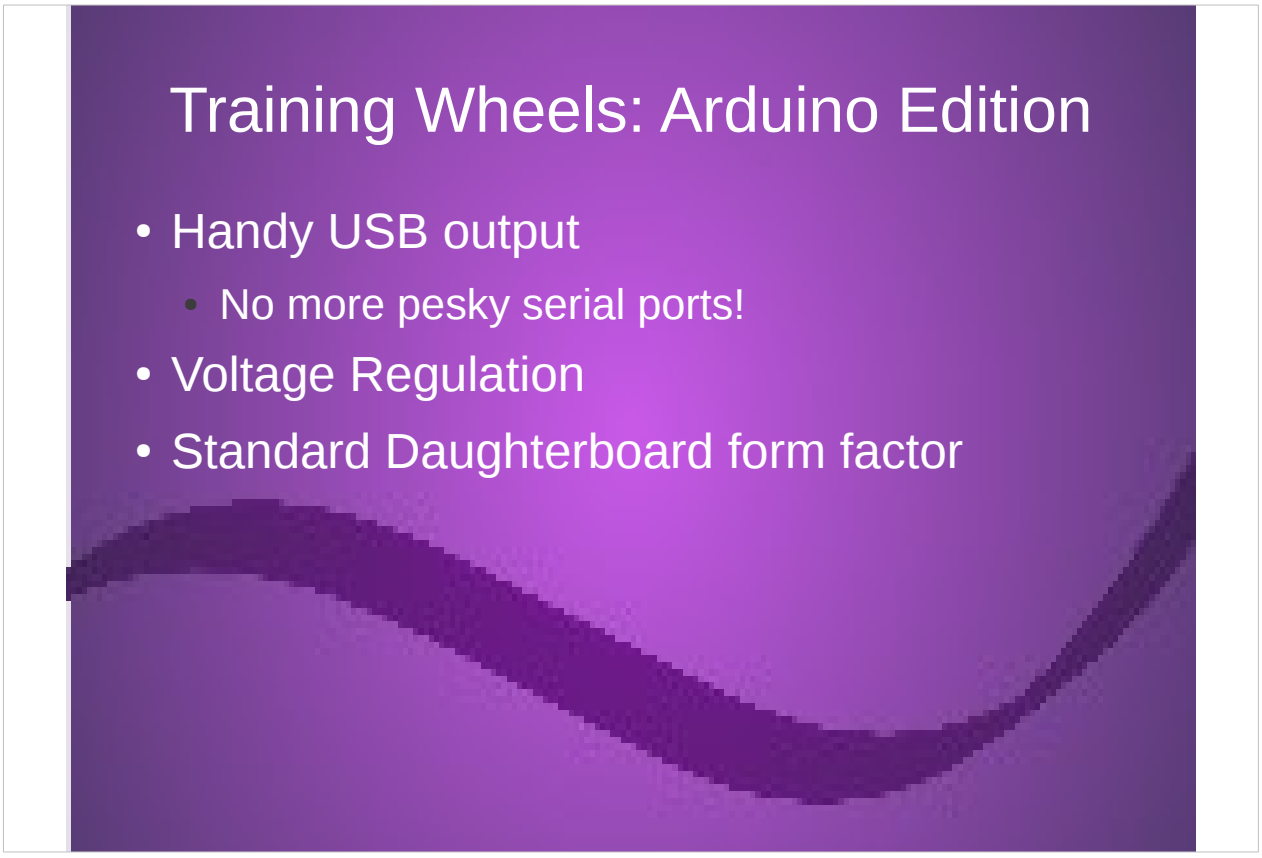
0b10000000 00001000

What can a Microcontroller do?

- Whatever it damn well pleases! It's a computer!
- Works slowly
- Lots of data is a problem (remember, 32kb, 8kb)
- One crazy guy made an arduino run linux
  - By emulating an ARM chip
  - Take 8 hours to start up

What Shouldn't a Microcontroller do?

- Not for super simple tasks
  - Blinking things
  - One shots
  - Easy logic (motor driving, pwm)
  - Analog stuff
- Not for things with a lot of data
- Community tends to look down on trivial solutions with arduino
- When in doubt, doing something is better than doing nothing



What projects are you guys into? Here's how arduino helps.

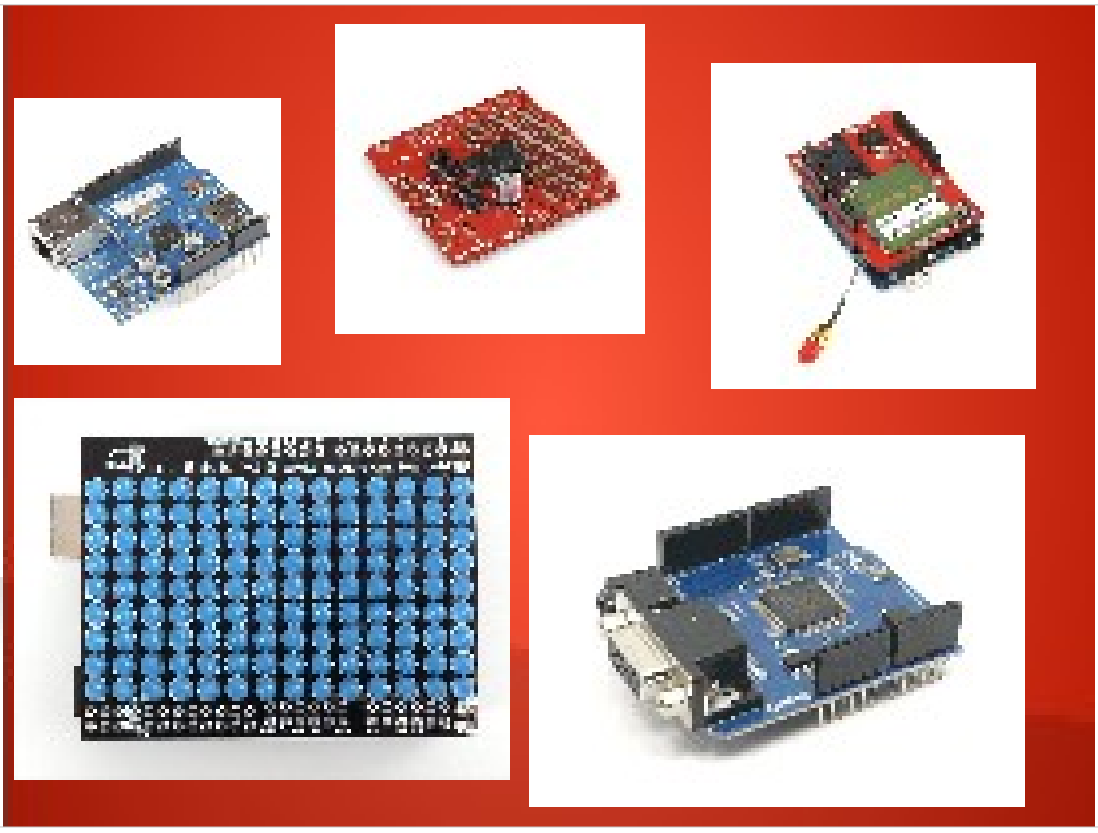
- FTDI acts as serial to USB bridge
  - Means no serial with USB at the same time
  - Counterexample: Teensy
  - Others?

Shields!

What are they?

- Standard way to attach hardware
- Stacking daughterboards
- Abstracts cool stuff to simple IO ports

What projects? Mention useful shields

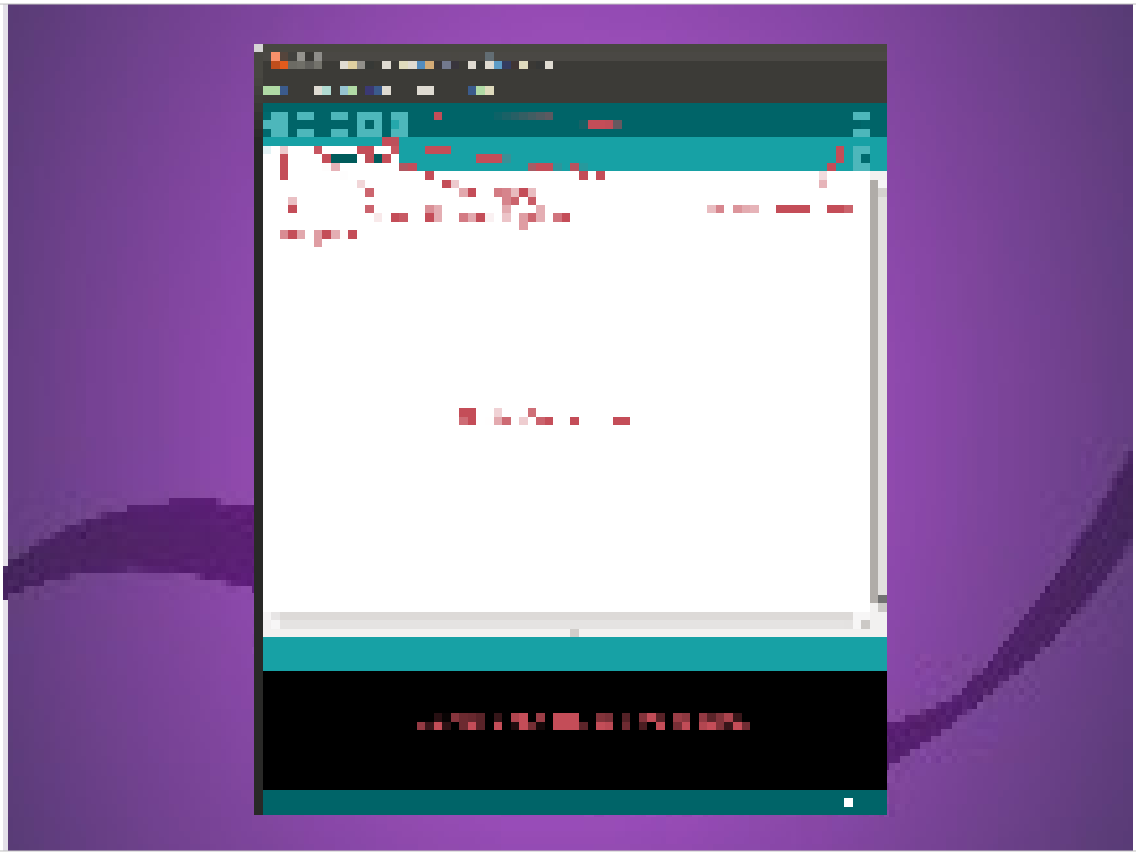


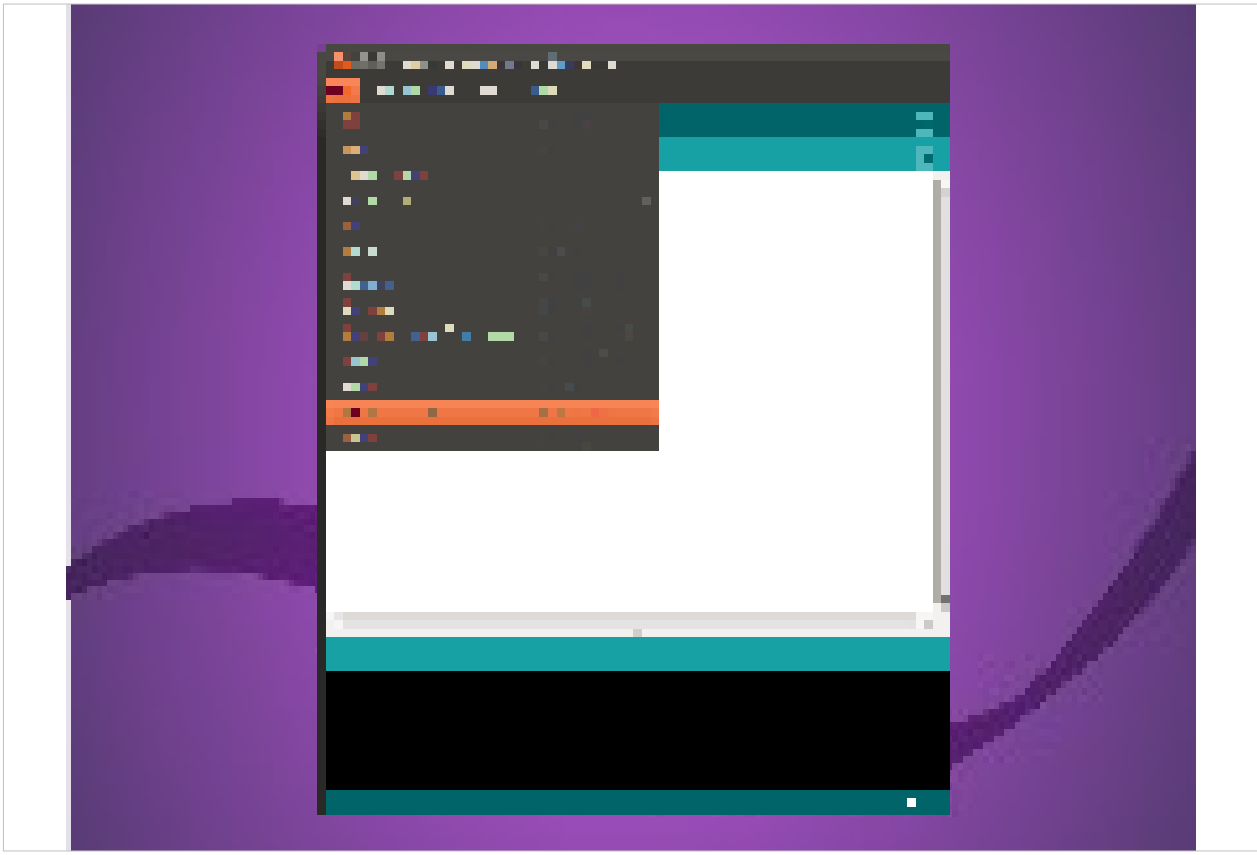
Libraries

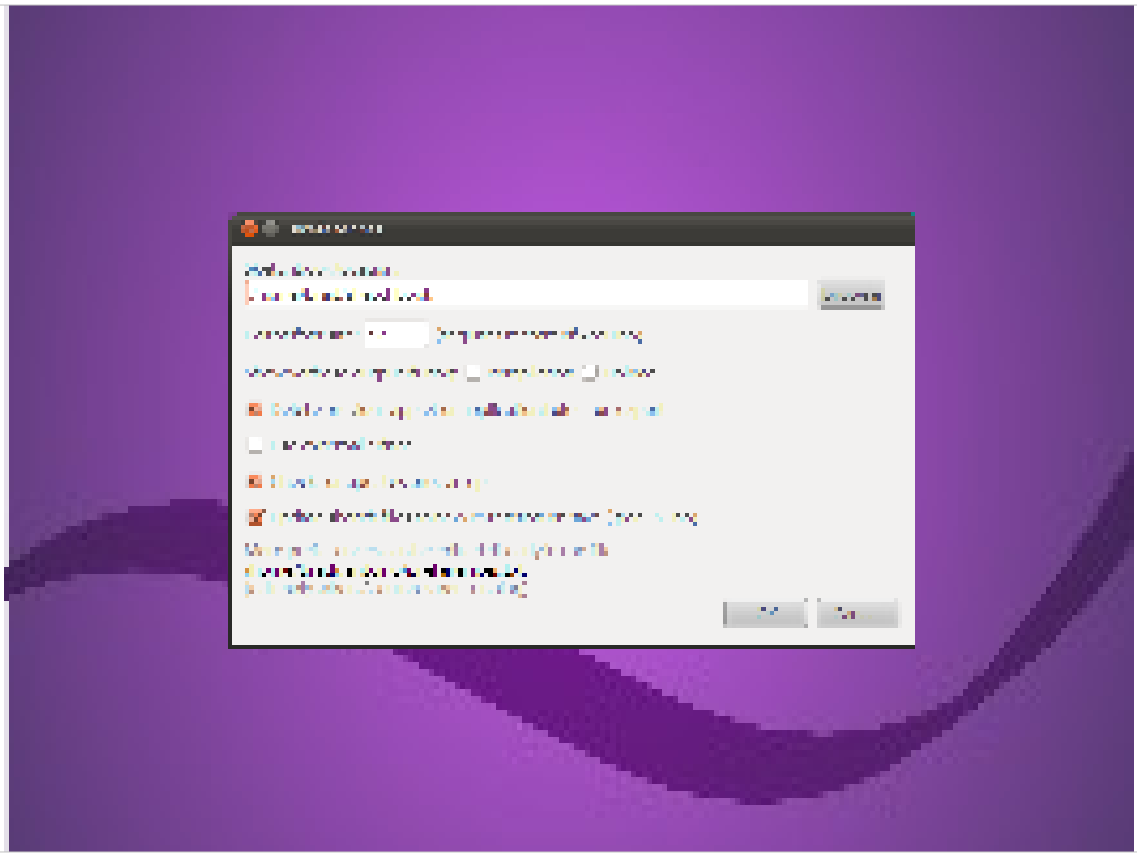
Code you don't have to write

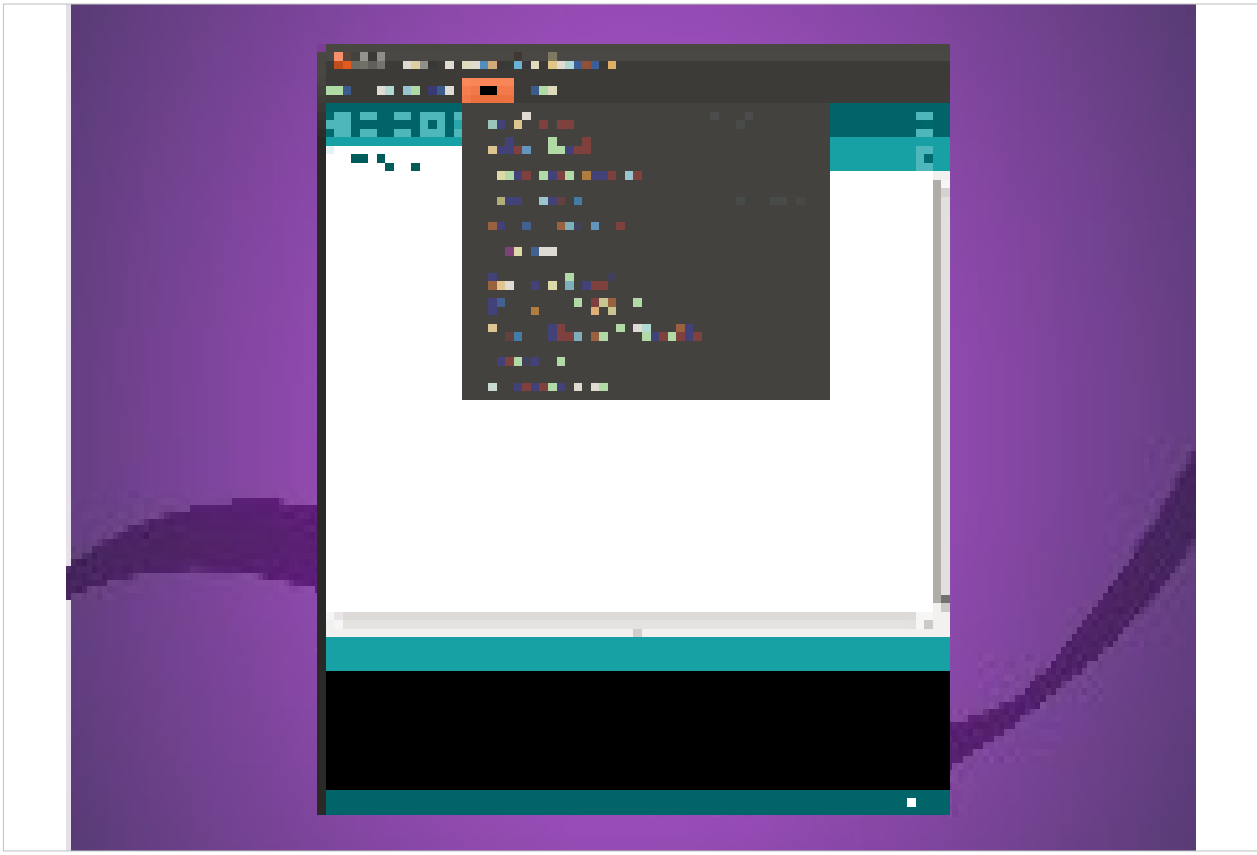
- Designed for a specific task
- Designed for specific hardware
- Abstracts complex operations into simple stuff
- What projects?

Getting into the IDE









About the IDE

- What Language?
  - C++, kinda. C, really.
- Advantages?
  - No Makefiles!
  - Automatic Dependency Gathering
  - Upload code quick and easy
  - Easy access to libraries

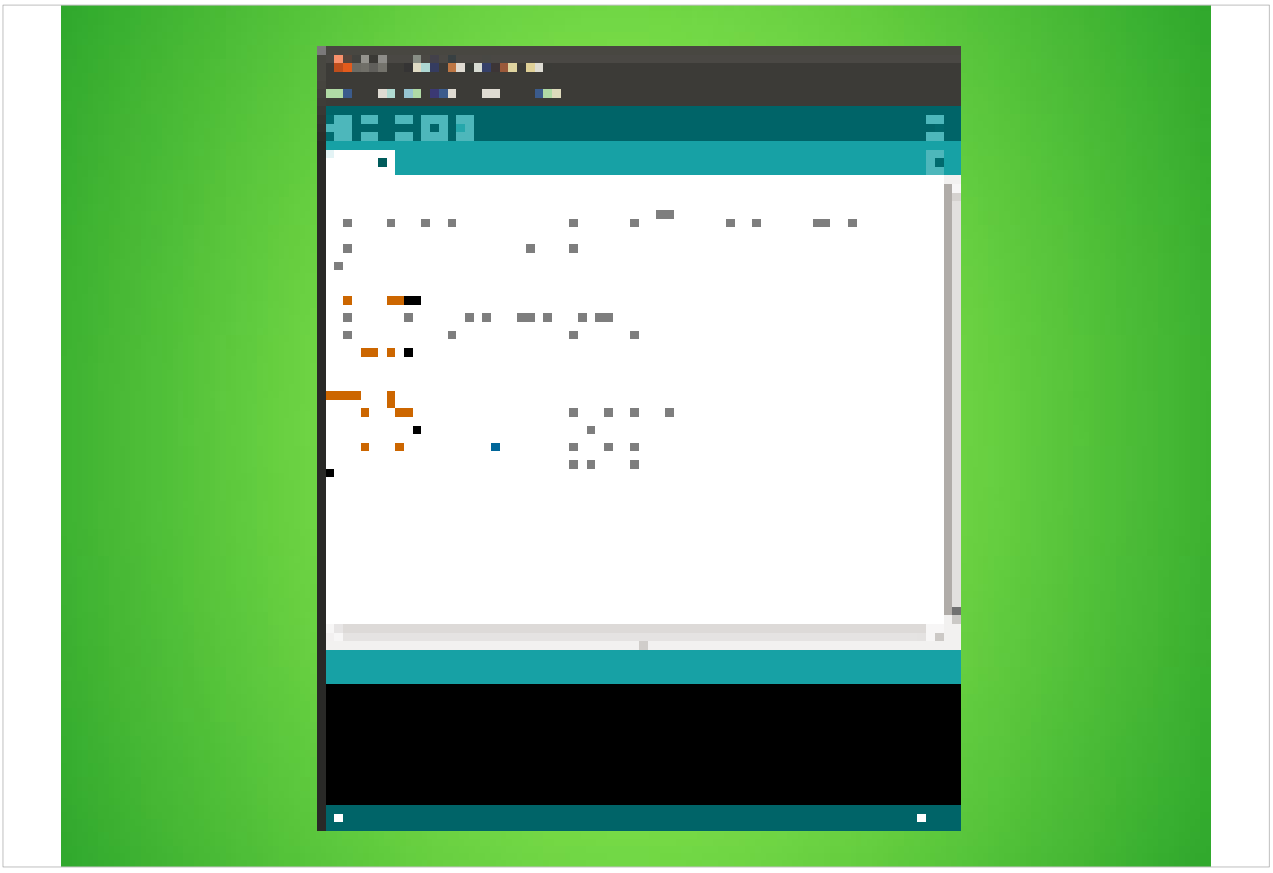
- C++ not really
  - No dynamic memory
    - Why? Stack and Heap
  - No templating
  - No metaprogramming
  - Limited data structures
  - No standard libraries
  - How many have C/C++ Experience?
  - Java?
  - Others?
- Direct access to pins hidden
- Includes some libs without help
- Arduino reset whenever serial port opened
-

Basic C

- Casting and Types
- Bitwise Operations
- Macros
- Comments
- Semicolon

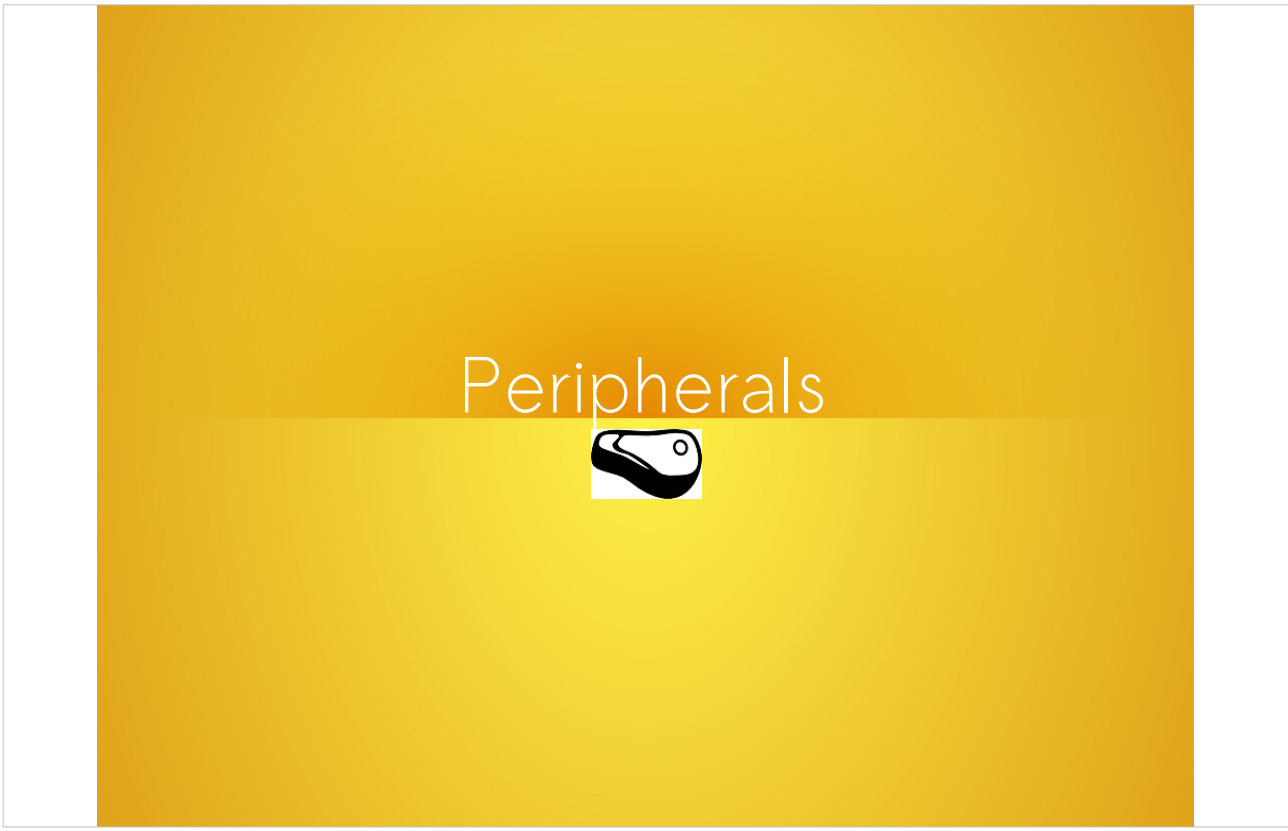
To move beyond the blink, you must first learn to blink.

Arduino Zen Master



Explain basic structure

- Setup
  - Stuff that happens before the rest.
  - Only happens once.
- Loop
  - Necessary because the microcontroller has to do **something** always.
- PinMode
  - Pins can be inputs, output, or input_pullup
- DigitalWrite (high or low)
- Delay (waits)



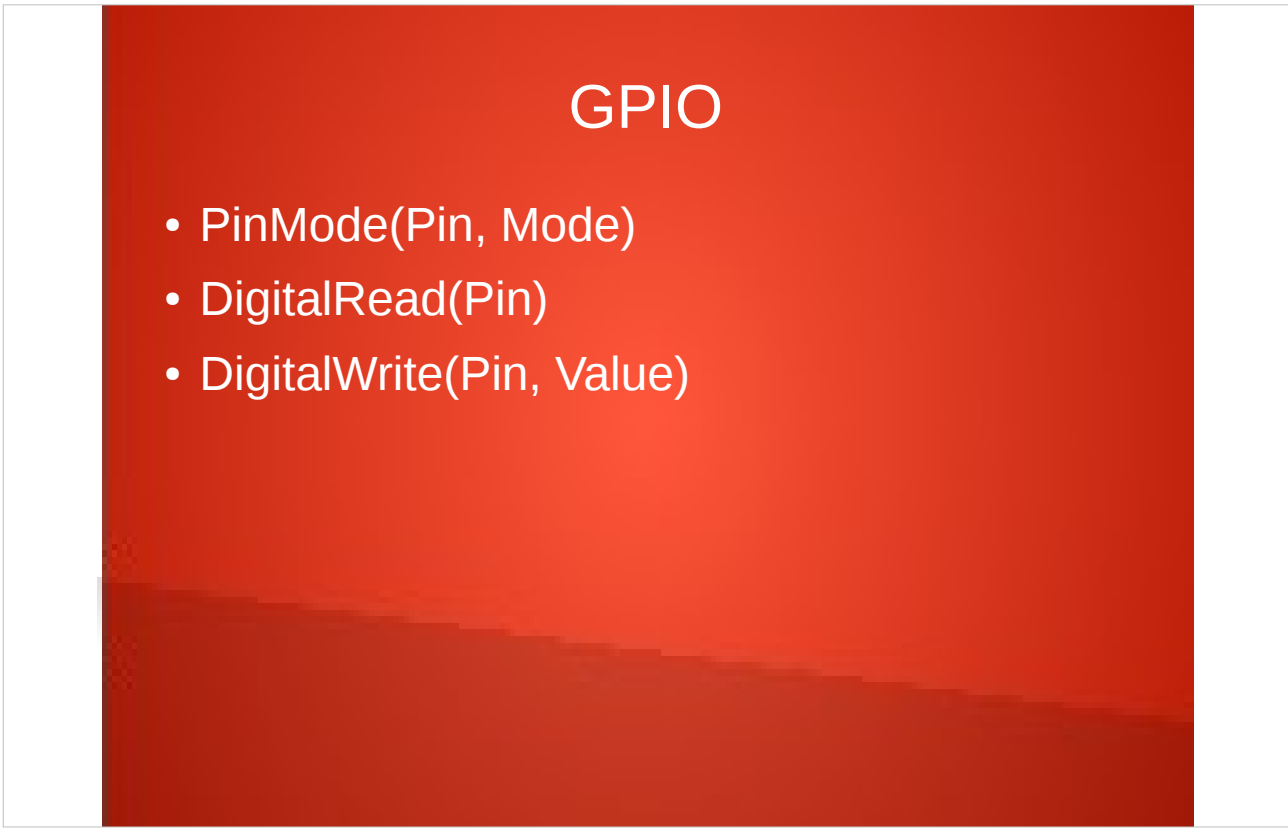
This is a big huge chunky section, there aren't really any good ways to break it up, so here it is.

Peripherals

- GPIO
- Timers
- I2C
- SPI
- Serial
- ADC
- PWM

GPIO

- Can use all pins as GPIO
- Operate as switch (high or low), 0 or 5 volts
- Must declare as output or input
- Inputs can be pulled up
- Can only output 20mA per pin
- Can input much more (open drain)



Pins are either a number (for digital pins) or the pin name (A0, etc for analog)

Mode is either OUTPUT, INPUT or INPUT_PULLUP

Read will return the value, high or low.

Write will write the value, high or low.

Timers

- Predictably, used to time things
- Configurable Counters
- Useful for:
  - Counting Events
  - Timing Events
  - PWM
  - Occasionally activating something
  - Waiting

Timers are Up Counters. The Arduino has three. Two are 8 bit, one is 16 bit. Their speed is set as some divisor of the main clock.

- Used for PWM output
- Arduino uses timer0 for everything.
- Millis and Micros use timer to tell time since startup
- Millis takes about 51 days to loop over
- PWM is analogWrite
- Not actually analog! Just PWM!
- Arduino uses Timer0 for everything!

Timers

- Delay(Milliseconds)
- DelayMicroseconds(Microseconds)
- Millis()
- Micros()
- AnalogWrite(pin, value)

- Delay causes program to do nothing for the specified number of Milliseconds (or microseconds). Not clockwork accurate, but good for short delays!
- Millis and micros measure time since startup
  - Millis loops at 51 days, micros at 70 minutes
- AnalogWrite outputs PWM
  - Only used on pins with asterisk

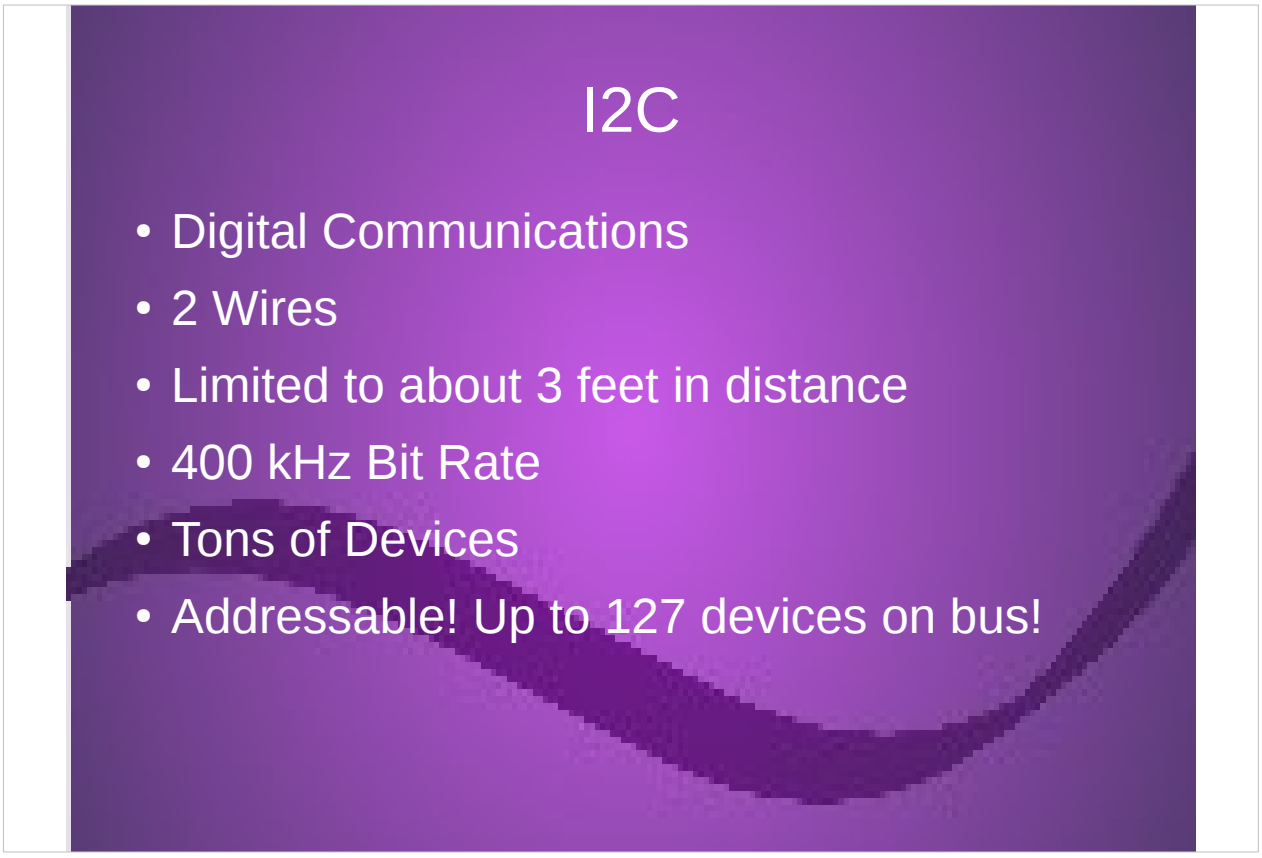
Measuring Time: Loop-de-loop

- Millis(), Micros(), Delay() and DelayMicroseconds() can all loop!
- If you're trying to measure time, just use the Time library!
- Otherwise, cast to int64_t, if your number suddenly goes lower than it was before, you've looped over!

Must measure often enough to ensure that you don't loop over more than once!

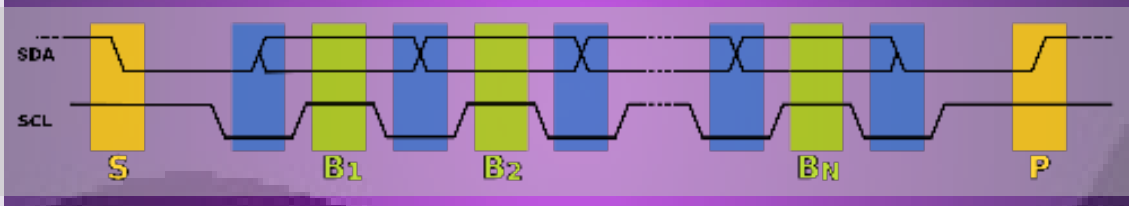
Timer Libraries

- Timer1 uses Timer1 instead of Timer0
  - More PWM control, doesn't interfere with time operations
- Timer
  - Modify PWM, basic scheduling operations, more time functions



- Limited by bus capacitence
  - Can be helped by being careful
  - Using nice wire
- Master/Slave bus
  - Arduino is almost always master
  - Can be multiple masters
- Devices include EEPROMs, Temp Sensors, Accelerometers, Motor Drivers, LED Drivers, IO Multiplexors, etc.

I2C



I2C

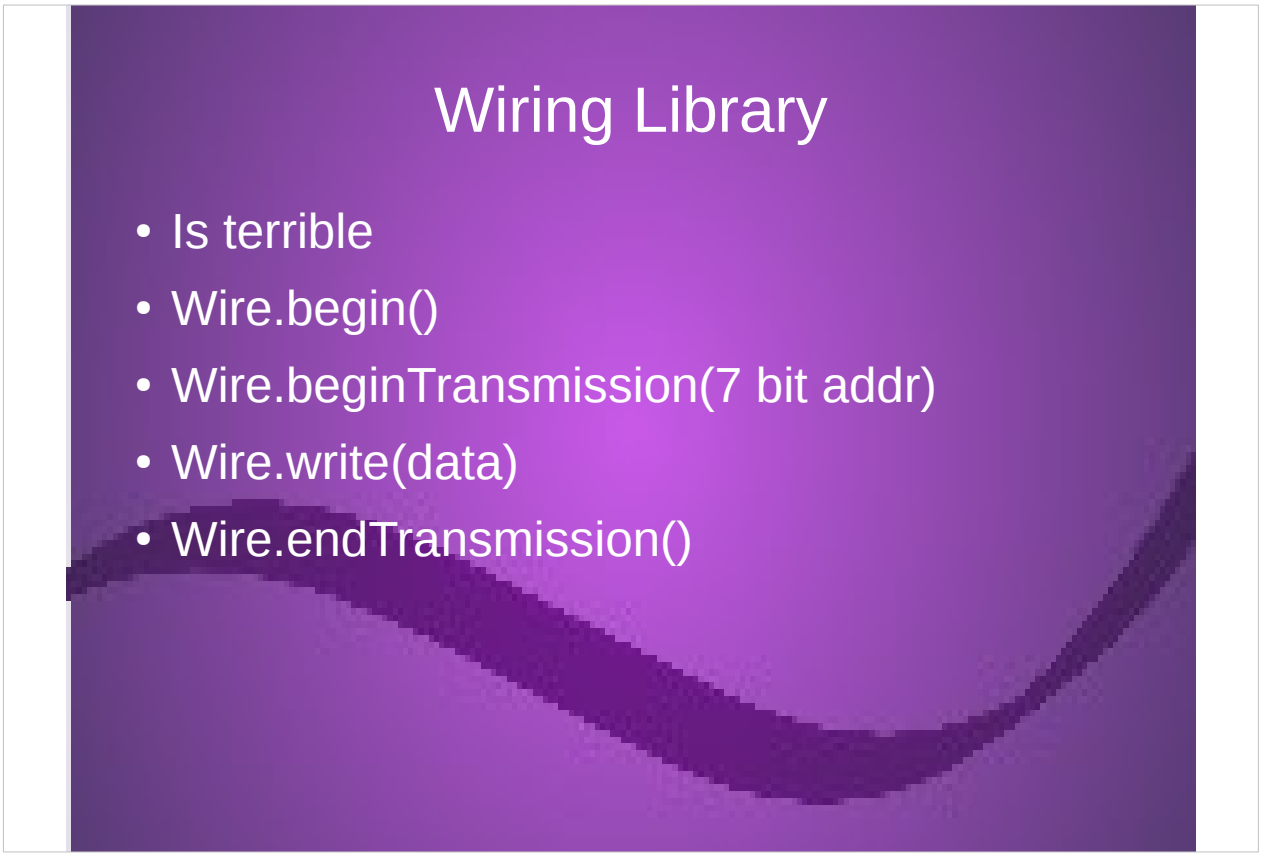
<START>+

<7 bit Address>+

<R/W bit>+

<DATA>+

<STOP>



The wiring library is terrible. It will crash, it isn't intuitive, you can't change settings, and it will freeze your arduino.

There is nothing better right now.

To join the bus as a master, call Wire.begin.

To send data to a device, call Wire.beginTransmission with the 7 bit address (this is often shifted down one bit from the address in the data sheet for the device)

Next, call Wire.write with the data you want to send. Finally, call Wire.endTransmission. The data isn't actually sent until you call endTransmission

To read data froma device, call Wire.requestFrom with the 7 bit address.

In a loop (checking wire.available before each read), call wire.read to read your data.

In order to join as a slave, call Wire.begin with your desired 7 bit address. You can still act as a master in slave mode!

Write functions that take an int and return void. The int will be the amount of data received (for on receive). Write a void function for onRequest. Use wire.send and wire.read.

SPI

- More serial data transfer!
- 3 Lines (minimum)
  - Clock
  - Data In
  - Data Out
  - (Chip Select)
  - (Reset)
  - (IRQ)
- Not individually addressable (well, chip select)

Can be much faster! Up to 25mhz data rate!

SPI

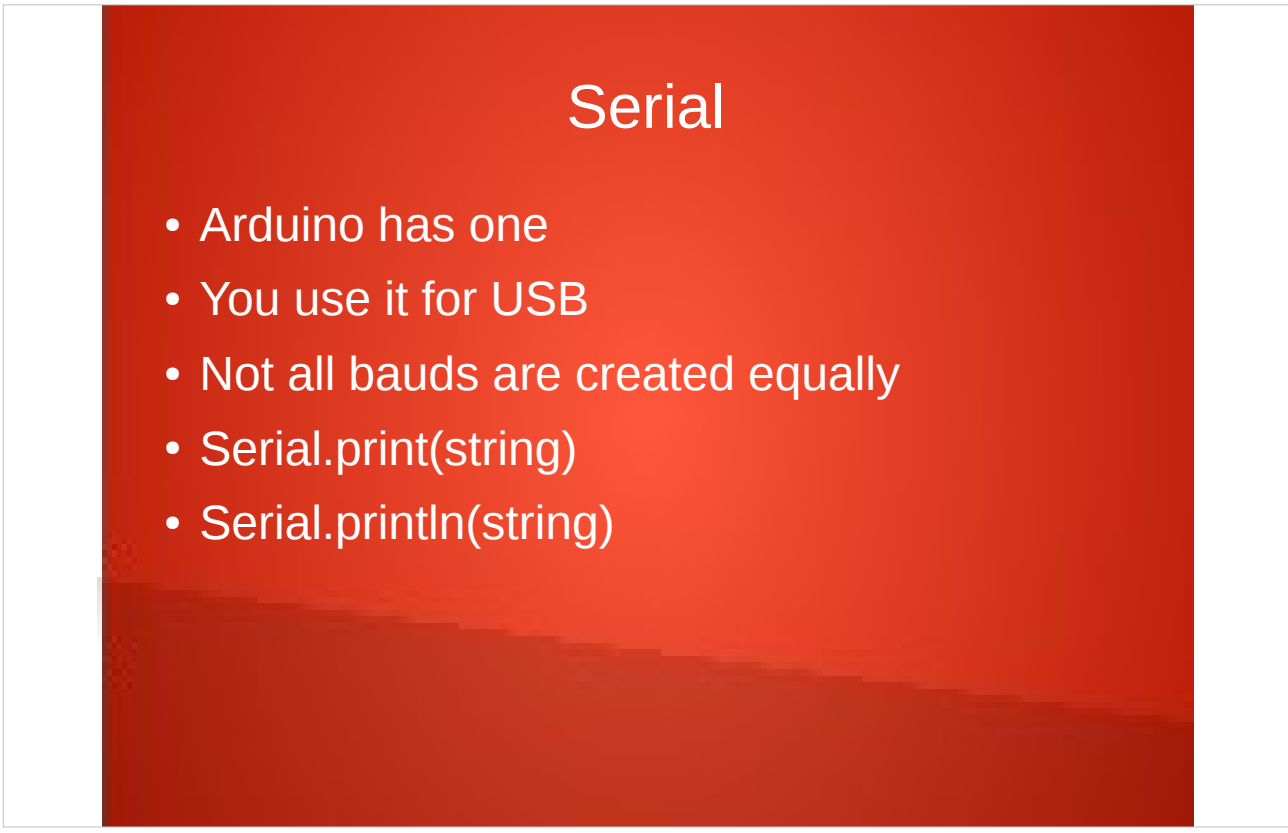
- Data read at the same time it is transmitted
- Devices are extremely diverse!
- Different messages, sometimes different endianness
- Will need to write custom stuff for each device
  - Unless there's a library!
- Arduino is always a master device
- Arduino provides SPI access primitives

Data is always read at the same time it is written.
To read, write nonsense.
Most devices, this can make sense.

SPI

- SPI.begin()
- SPI.end()
- SPI.setBitOrder(LSBFIRST/MSBFIRST)
- SPI.setClockDivider(SPI_CLOCK_DIV2/4/8/16/32/64/128)
- SPI.setDataMode(SPI_MODE0/1/2/3)
- SPI.transfer(byte)

Begin – attach to bus
End – detatch from bust
SetBitOrder – Endianess
ClockDivider
DataMode – sets polarity of bits (1 = high or low) and what clock edge is used (falling or rising)
Transfer- send the data! Returns bytes read in



The Arduino has one serial port, and it's used exclusively by the USB port to get code on the board, and to communicate with the PC. If you want to use the serial port for something else, you CAN'T do println debugging.

Soft Serial is an option.

Other Arduino-like boards have more serial.

You cannot select arbitrary baud rates, becuase they must be a divisor of the main clock.

Arduino Baud Rates

http://www.wormfood.net/avrbaudcalc.php

16 Mhz		
Bit Rate	U.BRR	% of error
300	3332	0.0
600	1666	0.0
1200	832	0.0
2400	416	0.1
4800	207	0.2
9600	103	0.2
14400	68	0.6
19200	51	0.2
28800	34	0.8
38400	25	0.2
57600	16	2.1
76800	12	0.2
115200	8	3.7
230400	3	7.8

Here's the table of possible baud rates for a 16 mhz AVR

New Soft Serial

- Not great, but the only option for bonus serial ports
- Uses a lot of CPU!
- Don't interrupt it!
- Uses timers
- Same interface as Serial

NewSoftSerial is an option for extra serial ports. Simply declare input and output pins, along with baud rate. Uses interrupts and timers to send serial data, but it uses a LOT of CPU. Outside of the declaration, you can use it just like other serial ports.

Analog to Digital

- analogRead(Pin)
- Involves math to convert to voltages
- Watch your battery!
- Don't read too fast!

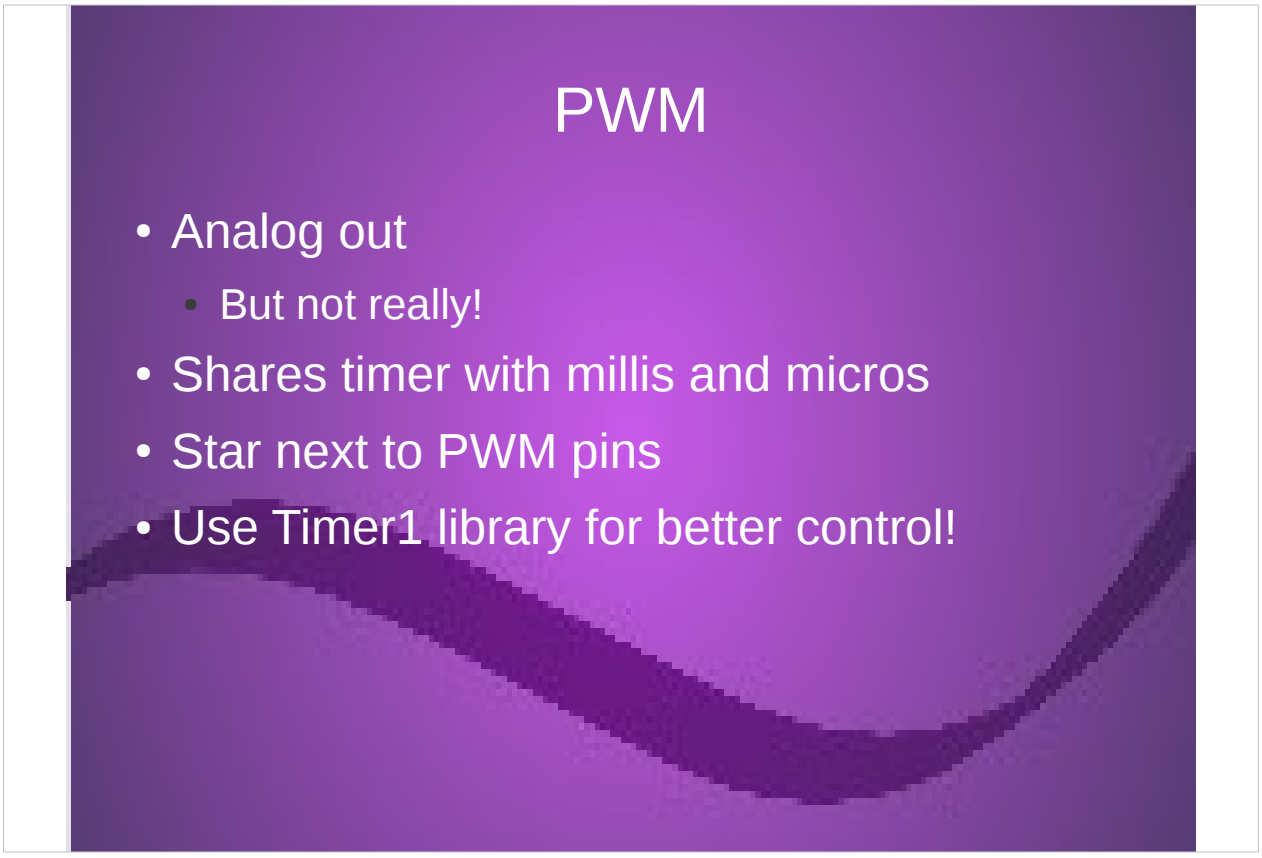
Just use analogRead! Pinmode has to be input, but otherwise, go for it!

If your battery drifts below 5v, your calculations for input voltage will be off.

Don't read faster than about 10kHz!

Example ADC Math

- Aref = 5v
- 1023 = 5v, 0 = 0v
- X/1024 * 5 = input voltage

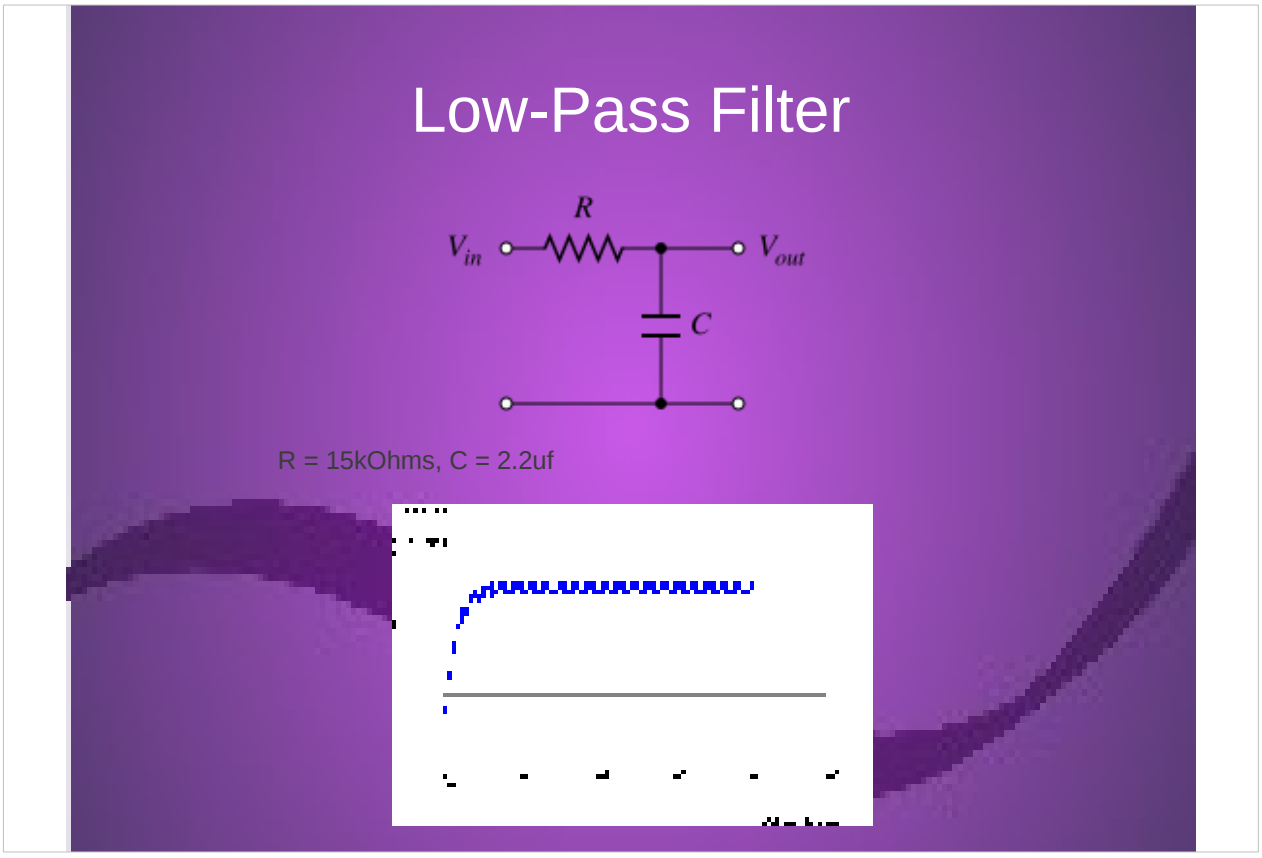


PWM is really limited on the arduino, just uses the analogWrite(pin, value) command.

Great for LEDs and Servos.

“Analog“ out

- PWM pulses very rapidly
  - Arduino at ~900kHz
- Duty cycle varies from 0-100%
- Fine for dimming LEDs, but what about real analog out?



Low pass filter is basically like a slowly leaking electricty tank for your signal. You fill it at a certain rate (the duty cycle of the PWM), and it drains at another rate. If you fill it faster, the tank rises (your output).

It's allowing the low-frequency (DC) components to pass, while the full force of the PWM swings are buffered.

For this, you sacrifice some response time.

Interrupts

- Interrupt your program when something important happens
- Allows the Arduino to react quickly to external stimuli
- Can be Internal or External

Internal Interrupts

- Called when something happens to a peripheral
- Generally, libraries take care of this for you!
- Example: Wiring Library

The onReceive command: This is called when some incoming data interrupts your microcontroller, and allows you to do something with it immediately!

External Interrupts

- Generic, outside your Arduino
- INT0 = D2, INT1 = D3
- attachInterrupt(pin, function, mode)
- Mode = LOW, CHANGE, RISING, FALLING
- Function = void doSomething()

External Interrupts

- Pin Change interrupts
- Operate on banks of pins
- More complicated
- If you really need more interrupts, use the third-party pinchangint library

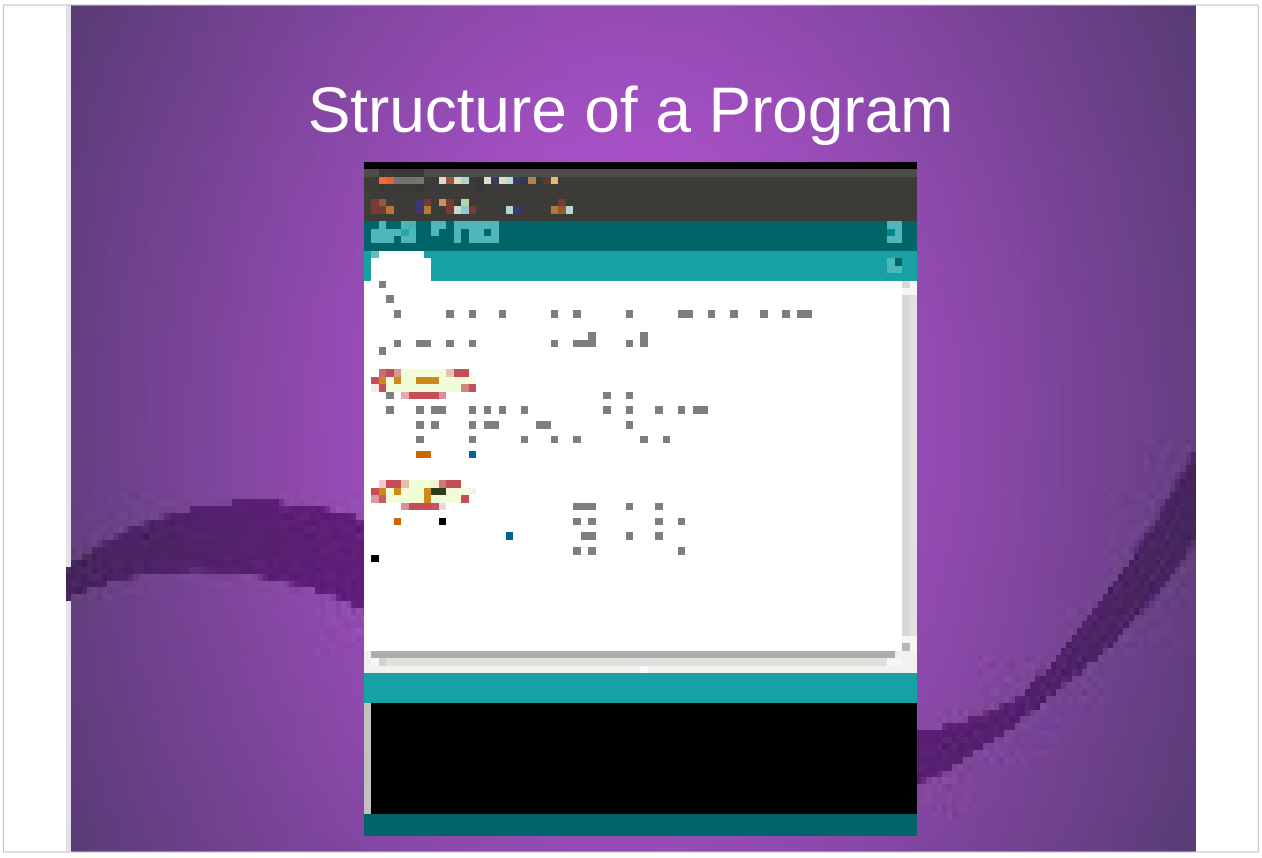
Arduino Problems

- Arduino 1.0 incompatibilities
- Automatic Dependency Generation
- Adding Libraries
- Crummy Libraries
- Failure Conditions
- Walled Gardens
- Code Space

•Arduino-land isn't all fairies and cocoa puffs
•Arduino recently released 1.0, and a lot of libraries changed, rendering a lot of old software incompatible with the new stuff. If you're getting a lot of esoteric errors, this is probably why!
•Automatic dependency generation sometimes fails. No good way to fix this!
•Can't use precompiled libraries from Atmel!
•Difficult to add libraries!
•There are lots of terrible libraries!
  - Almost anything third party
  - Even some first part (wiring, looking at you!)
•Walled Gardens of code can be difficult to break out of-- lots of really specialized stuff that will lead you down wrong paths
•No good libraries for sleeping or power saving.
•No way to really tell when bad errors occur: printf debugging is the main debugging. Lots of symptoms for different problems, hard to chase them down sometimes. No OS to tell you where things went pear shaped. Look at RAM use and nested function calls
•Errors can be very bad, and result in exploding things, or ramming motors into things, or fire!

Basic Embedded C

- Structure
- Println Debugging
- Interrupts
- Anatomy of a Function Call
- Volatile
- Structuring your code
- Libraries
- Other Tricks

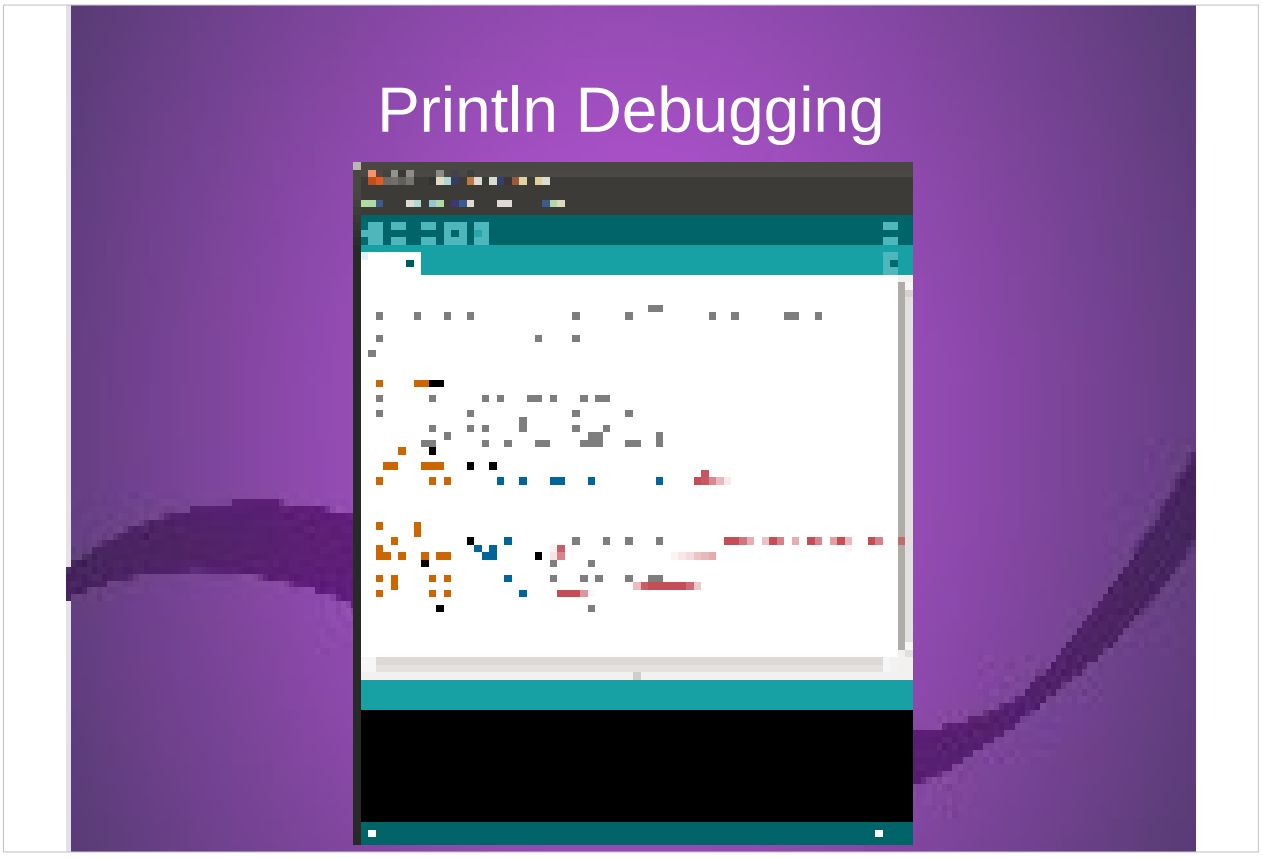


A simple embedded program has two main parts: Initialization and Loop code.

Initialization code starts everything up the right way and in the right order (so that your machine doesn't break itself right away, for example!). Pin modes are set, devices are initizlized, and you move into a loop.

The loop is what your program does all the time. An embedded program must never exit, because it has nothing to do once it exits! Your desktop does: It keeps listening to the network, displaying your desktop, etc. Your embedded program is more bare metal than that! You have to specifically tell your program to do nothing if you want it to do nothing! Typically, you set it up to do a bunch of background maintenance tasks, then you let interrupts handle the rest. If you don't want to, or can't use interrupts for something, then this loop can be a polling loop: Checking often to see if some condition is met.

Anyone have any ideas how this might work?

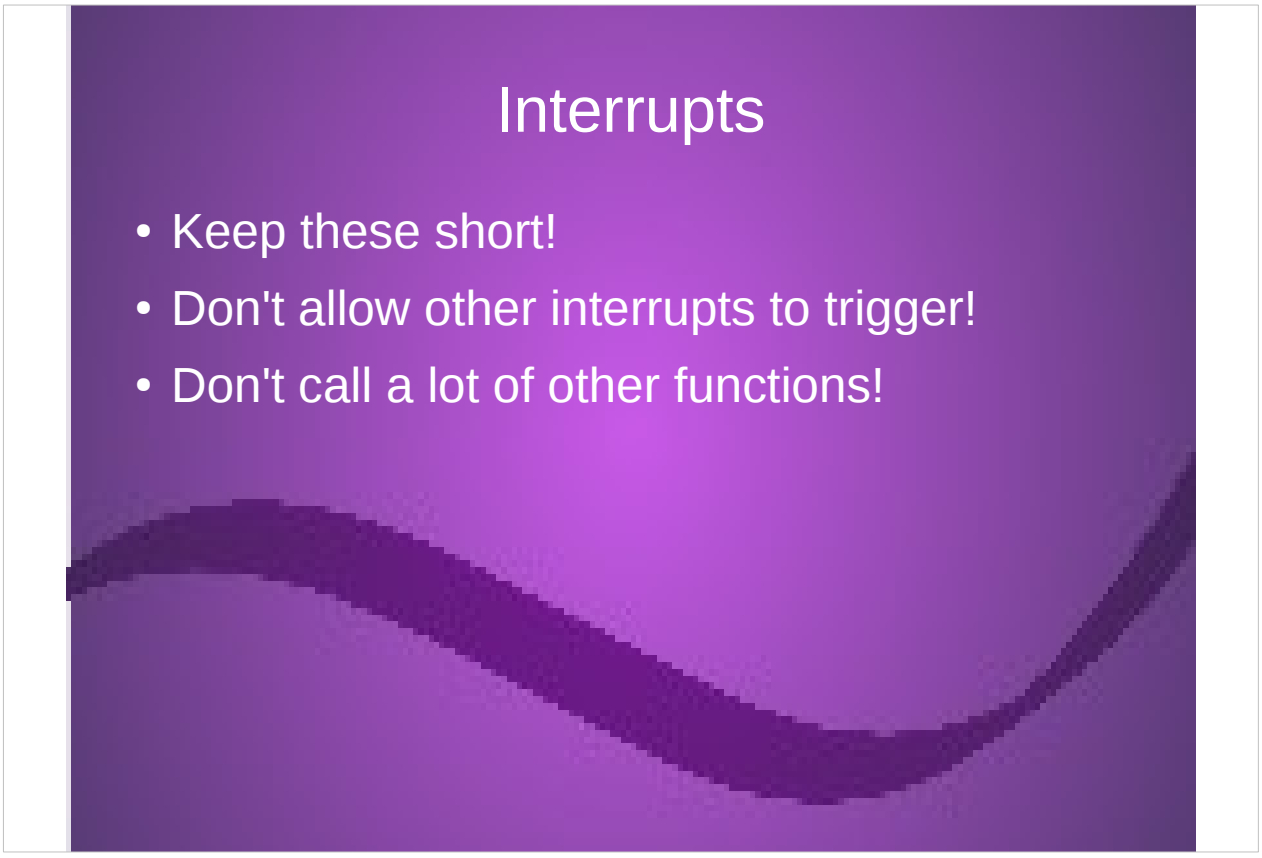


Println debugging is just what it sounds like: Using a bunch of Serial.println statements to know what your code is doing. You can't always do this, though.

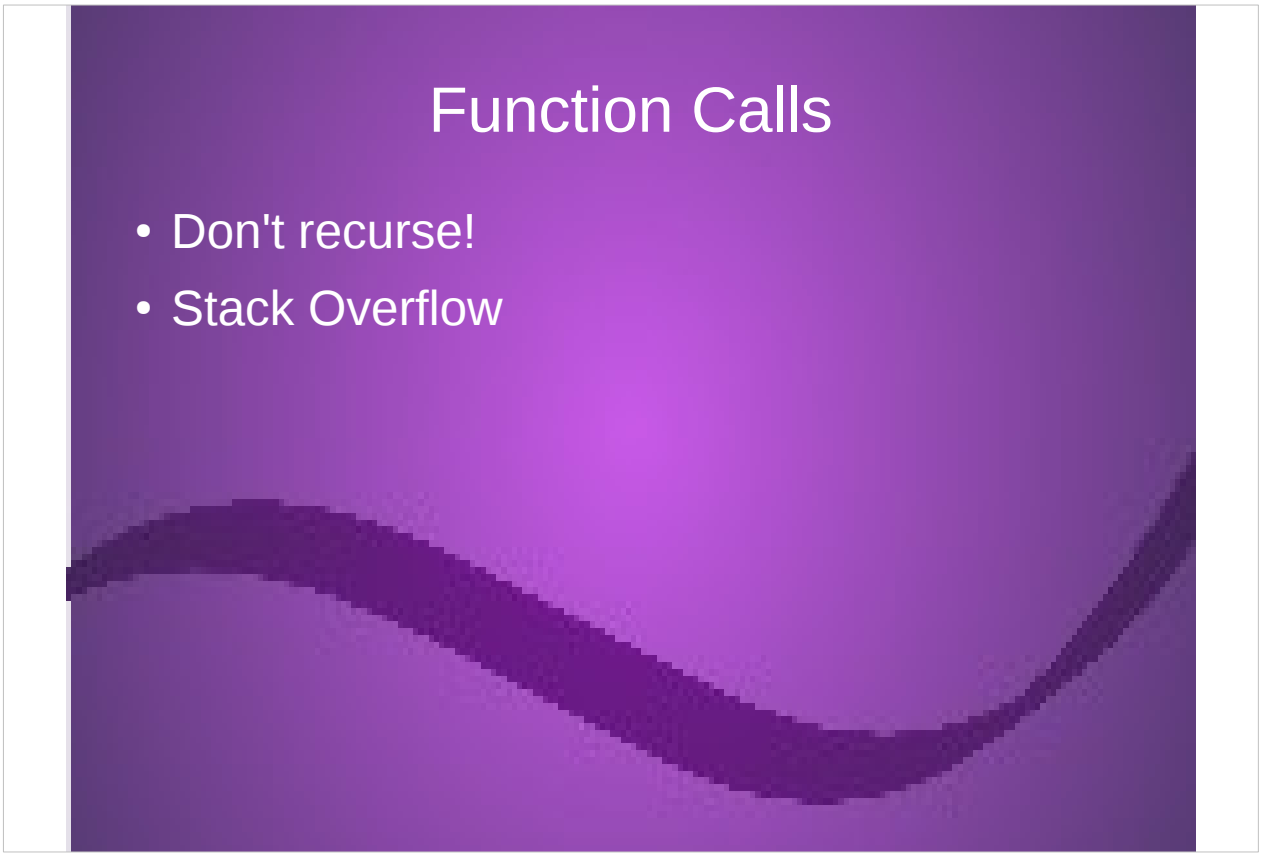
What situations might not be appropriate for Println debugging?

Interrupts! Doing things fast! Println is slow!

Tight code programs, strings take up a lot of space!

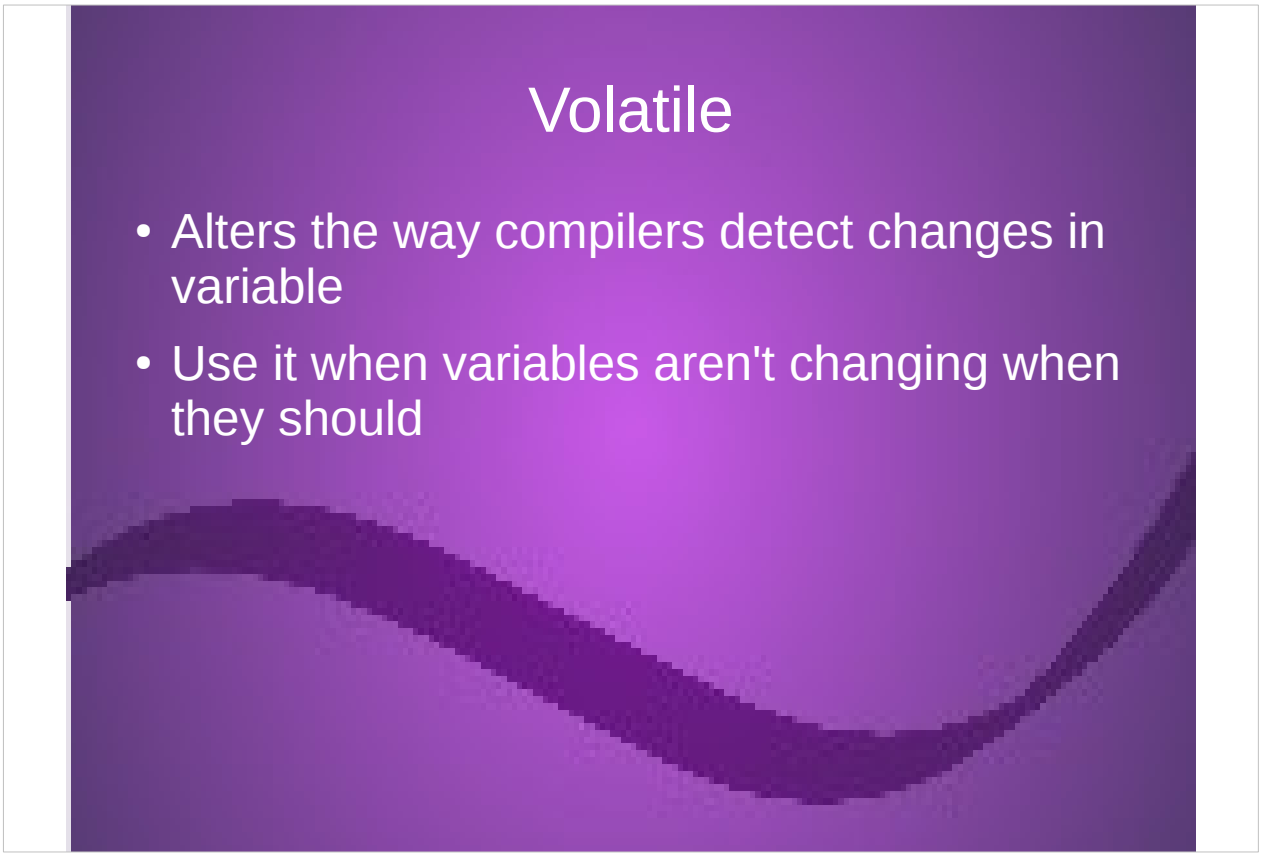


Interrupts literally stop the processor in its tracks! It saves its state (which takes some memory!) and does something else for a while. If you let multiple interrupts occur at once, you can run out of memory! Same thing can happen if you spend a lot of time there, or nest function call. Keep 'em short!



Don't use recursive functions, or, if you do, put a hard cap on the recursion depth. Recursive functions can easily use all the memory (since each function call has a tiny memory overhead).

Stack overflows are bad, because they jump your program into an unknown, unrecoverable state! This can catch things on fire, or destroy things.



C compilers attempt to predict when variables can change. Normally, it's pretty good, but sometimes, hardware changes a variable unpredictably. Use volatile when you want to tell the compiler, „this variable could change at any time“.

Globals are to be avoided. Sometimes they're useful, and can be tracked easily. Other times, you lose track of where and what can change them from one state to the next, and then you're screwed.

I break this rule all the time, but try not to.

Switch-Case

- Multiple choices?
- Use me instead of if-else-if chains!

Beyond Arduino

- Arduino + Pure C = Beyond!
- AVR – Arduinowheel = Beyond!
- Other Microcontrollers = Beyond!

Arduino + Pure C

- AVR studio or AVR-GCC are both great!
- AVR LIBC is well documented!
- Tons of libraries
- AVRFreaks.net
- AVRDude

If you want to cram more stuff on the arduino, or you need tigher timing requirements, or you're tired of the limitations of the arduino IDE, you can program any arduino board in pure C with AVR-GCC (or AVR studio) using AVR Libc

Windows users – download winavr

AVRdude has a command line switch for uploading to an arduino! Almost as easy as the ide!

Need to write makefiles :(

AVR - Arduinowheels

- Just need an Atmega328, voltage supply, and an AVR programmers
- Programmers start at $20
- A nice programmer is $50

Without the arduinowheels, you can diversify your power requirements. Need something super low power? Take off all the wasteful regulators!

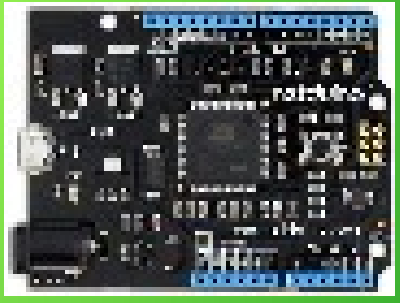
You will need a programmer to get C code on the chip, but they're cheap!

Other Microcontrollers

- Tons of choices!
- All the same principles apply!

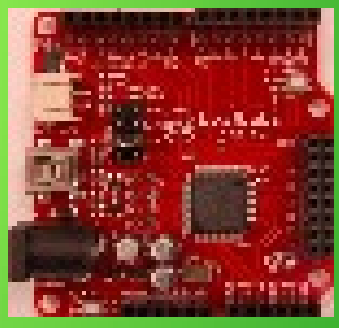
NetDuino

- Program in C#
- Super easy!
- Lots of code space!
- Not as great for robots



LeafLabs Maple

- ARM Chip
- Arduino-like environment
- FAST!
- More IO!



PICs

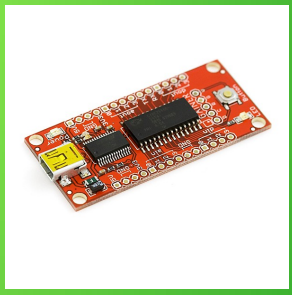
- Before Arduino, these were **the** hobbyist chip
- Cheap!
- There's a million of them!
- Tons of libraries and support

MSP430

- Cheap dev boards
- Super, super low power
- Reasonable community
- Flexible peripheral set

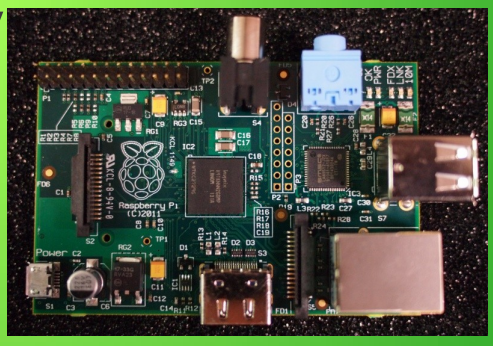
PSoC

- Mixed Signal Microcontroller
- Difficult to develop for
- It is a mystery



RasPi

- Cheap!
- Linux!
- Not so good for robots
- Closed and Proprietary



Other ARM

- Seriously, more than I could ever describe
- If you're to this point, you're definitely “Beyond“

END OF LINE