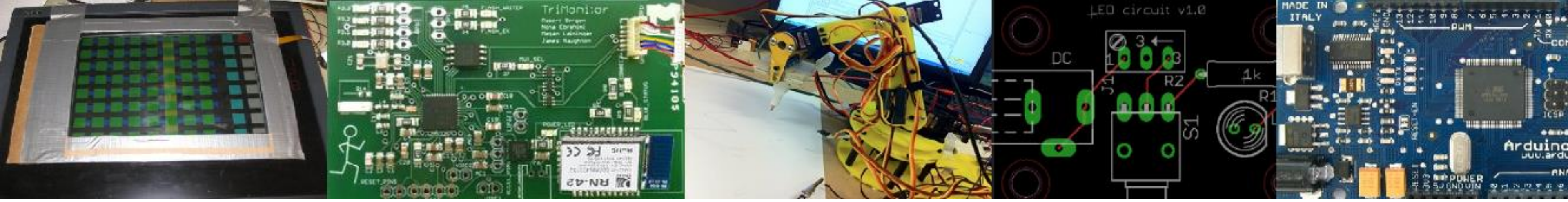
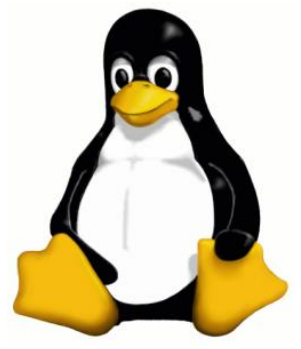
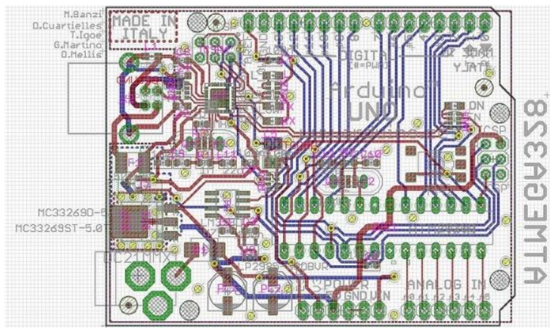


EECS 473 Advanced Embedded Systems



Lecture 1:

Class introduction
Design in this class
Creating software interfaces to hardware



Welcome

- This is a class on embedded systems. We will be…
  - Learning more about embedded systems
    - Lecture, labs, and, homework.
  - Gaining experience working as a group…
    - …but also working by yourself on researching and learning material without an instructor's guidance
  - Learning (more?) about the design cycle
    - Both theory in lecture and by doing a paper-to-prototype-to-project design.

Who am I?

- Dr. Mark Brehob
  - Prefer “Mark”, “Dr. Brehob” is okay too.
  - Full-time teacher (lecturer)
    - Been here for 20+ years and I’ve taught a wide variety of courses (100, 101, 203, 270, 281, 370, 373, 376, 452, 470, 473)
  - PhD is in the intersection of computer architecture and theoretical computer science as it relates to caches.
  - See http://web.eecs.umich.edu/~brehob/

Staff

- Senior Engineer
  - Matt Smith
    - Runs 270/373/473 labs

- GSIs:
  - Guthrie Tabios
  - Alan Tonkryk
  - Andrew (Andy) Zaloudek

On-line resources for the class

- Web site (primary location for course materials)
  - http://www.eecs.umich.edu/courses/eecs473/
  - Includes all labs, handouts, old exams, etc.
  - Links to Piazza and Gradescope pages at the top.
- Gradscope
  - Entry code: **WB3ZRW**
  - All assignments will be turned in there.
  - All graded assignments and exams will be posted there.
- Piazza
  - https://piazza.com/umich/fall2023/eecs473
  - Class forum. Please feel free to answer each others questions.
  - Link to join is also on the website.

Today

- Class intro
  - Grading, schedule, etc.
- A bit on design
  - Design requirements, engineering specifications, etc.
- Start on hardware interfaces
  - I've got a lot more slides than I expect to use
    - Hope to get through the breakout room stuff.
    - The Arudino stuff at the end is just reference.

Class Introduction

- Learning more about embedded systems
  - You’ll do 5 labs in 5 weeks
    - Labs 1 and 2 are microcontroller/rapid prototyping/interfacing
    - Lab 3 covers real-time operating systems (RTOS)
    - Lab 4 introduces embedded Linux and writing Linux device drivers
    - Lab 5 gives an introduction to PCB design
  - For the first 4 weeks, lecture will be mostly about supporting lab.
    - But will then focus on other issues including PCB power, embedded wireless, and DSP as (the) example of application-specific CPUs.
  - Homework
    - Give you more practice working with technical documents related to embedded systems
    - Review material learned in lecture

We will be…

- Gaining experience working as a group…
  - This is more-or-less the other side of your Engineering 100 experience.
    - We want you to use the technical knowledge you've learned in the last few years to make it through the whole engineering cycle with a team design.
  - Your labs will be in groups of 2
  - Your projects will be in groups of 4 to 5 (mostly 5)
    - Your group will make a schedule, create a budget, and divide up the work.

We will be…

- …but also working by yourself on researching and learning material without an instructor’s guidance
  - The project will be done without lecture to guide you.
    - » Each group will be doing something different and your group will be more expert than any of us on the topic (at least by the time you are done…)

We will be…

- Learning (more?) about the design cycle
  - Both theory in lecture and by doing a paper-to-prototype-to-project design.
  - Back to Engin 100 principles but with your engineering knowledge and 373 experience to provide context.

Prerequisites

- You must have a background in
  - Embedded design
    - Memory-mapped I/O, interrupts, serial bus interfacing, etc. (EECS 373)
  - Digital logic, C and assembly programming
    - Pointers, Verilog/VHDL, etc. (EECS 280, 270, 370)
  - And **either**
    - A solid programming background (EECS 281)
    - Or a reasonable circuits background (EECS 215)

Warning

- If you don't have 281
  - Lab 3 and 4 are going to be rough.
- If you don't have any circuits or electromagnetics background (EECS 215 or Physics 240)
  - About 2 weeks of lecture are going to be very difficult.
  - Come to office hours.
- If you've never taken an embedded systems class before
  - You're in the wrong place.
- Also, you'd ideally have some soldering experience including surface mount work. If not, we hope to have some lessons available in the next few weeks.

Class structure

- We will meet for 3 hours/week as a class
- Weekly labs for the first 5 weeks
  - Though the last lab is a bit shorter and different.
- During the first 4 weeks
  - we are going to “overstaff” the labs.
    - 2 of us in the lab.
  - And have about 10 hours/week of lab office hours.
- After week 4 we’ll spread out support more
  - Open lab
  - Schedulable times (“reserve a GSI”)

Office hours

Instructor

- **Mark Brehob**
  - Monday 4:00-5:30pm
    - 4632 Beyster
  - Thursday 5:00-6:00pm
    - In lab (2334 EECS)
  - I also will be available after lecture, generally in the hallway or outside (turn left as you leave the classroom)

Lab folks (in lab, 2334 EECS)

- **Alan**
  - Wednesday 9:30-10:30pm
  - Thursday 9:30-10:30pm
  - Friday 1-3pm
- **Guthrie**
  - Tuesdays 12-3pm
- **Andy**
  - Monday 12-3
  - Friday 12-1

We are double staffing lab during September. Once lab is over, the GSIs will be changing hours and adding “flex hours”.

Work/grades

- 20% Labs
- 5% Homework and guest speaker attendance
- 17% Midterm
- 18% Final exam
- 40% Final Project

Labs

- There are 5 labs
  - 2 Prototyping with Arduino, 1 RTOS, 1 Embedded Linux, 1 PCB
  - **Pre-labs** are done <u>individually</u> and are worth ~25% of the lab grade.
    - They are due before lab starts
  - In-labs and post-labs are done in groups of two.
    - They have two parts: a “sign-off” part and a “question” part.
    - The post-labs are just an extension of the “question” part of the in-lab
    - They are due before the start of the next lab.
  - Late labs lose 10% per school day late.
- Lab 5 is done entirely individually.
  - Can be done outside of the lab.

Project (1/2)

- You will work in groups of 4-5 on designing and building an embedded system of your choosing
  - Significant budget
    - ~$200/student
    - Sponsorship from Infineon!
  - There will be an emphasis on having a reliable system in place.
  - If you have an external group that wants something made and is willing to pay for it.
    - I'm open to discussion

Project (2/2)

- There will be a number of due dates (proposal, milestones, final project)
- There will be a significant degree of formalism in your reports and presentations.
- Your project will be presented at the CoE design expo on Thursday November 30th.
- ***<u>You have significant design freedom.</u>***
  - The only real restrictions are that it has to use a processor, be doable in the time given, be technically interesting, and do something useful or interesting.
    - We expect groups will make a PCB.
  - As you think of ideas, please feel free to run them past me.

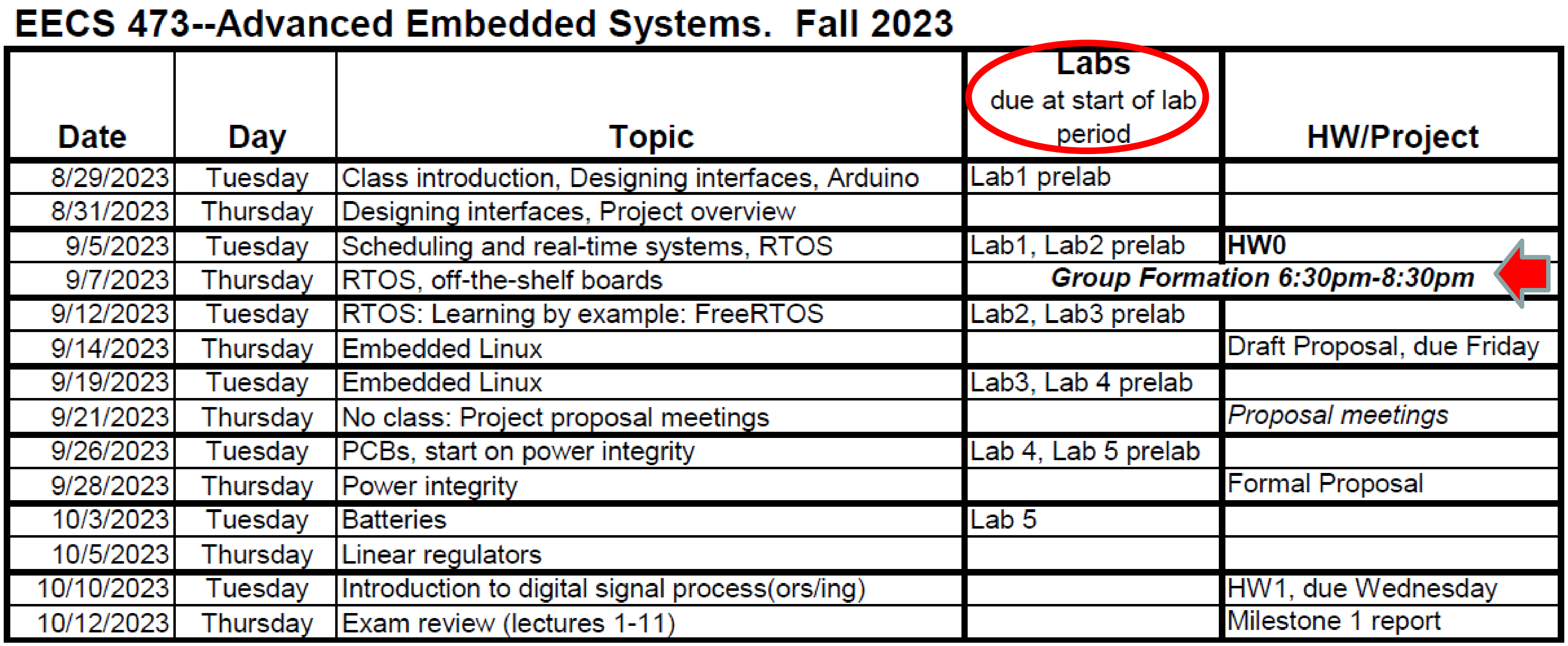
Exams

- A bit after the 5 labs are done, there will be an exam. It will cover the lab/class material up to that point.
  - Midterm is planned to be on ***Wednesday 10/18*** at 6pm
    - This may change, should know a few days after the Internet problems go away.

Homework

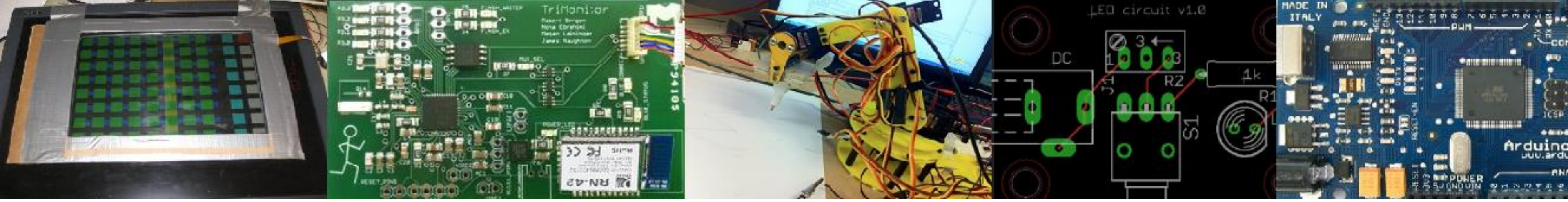
- You will have three homework assignments.
  - The first is to propose project ideas (HW0)
    - Already sent out
  - The other two will be used to reinforce classroom material or to give you a chance to drill down a bit farther on topics then we can in class/lab.

Schedule



EECS 473--Advanced Embedded Systems. Fall 2023

Date	Day	Topic	Labs due at start of lab period	HW/Project
8/29/2023	Tuesday	Class introduction, Designing interfaces, Arduino	Lab1 prelab	
8/31/2023	Thursday	Designing interfaces, Project overview		
9/5/2023	Tuesday	Scheduling and real-time systems, RTOS	Lab1, Lab2 prelab	**HW0**
9/7/2023	Thursday	RTOS, off-the-shelf boards	***Group Formation 6:30pm-8:30pm***	
9/12/2023	Tuesday	RTOS: Learning by example: FreeRTOS	Lab2, Lab3 prelab	
9/14/2023	Thursday	Embedded Linux		Draft Proposal, due Friday
9/19/2023	Tuesday	Embedded Linux	Lab3, Lab 4 prelab	
9/21/2023	Thursday	No class: Project proposal meetings		*Proposal meetings*
9/26/2023	Tuesday	PCBs, start on power integrity	Lab 4, Lab 5 prelab	
9/28/2023	Thursday	Power integrity		Formal Proposal
10/3/2023	Tuesday	Batteries	Lab 5	
10/5/2023	Thursday	Linear regulators		
10/10/2023	Tuesday	Introduction to digital signal process(ors/ing)		HW1, due Wednesday
10/12/2023	Thursday	Exam review (lectures 1-11)		Milestone 1 report



A (Very Brief) Introduction to Design

Thinking about how to think about building things.

(yes, it’s that abstract, but also critical)

What is the design process?

- Unlike the material in 95% of your engineering classes, this question is a matter of opinion.
  - There are tons of books on the topic, and they all use different words and emphasize different things.
  - That said, they (almost) all have similar ideas.

The stages of design

- Where design ends is debated.
  - Pretty much everyone agrees that identifying a problem to be solved is the first step.
    - Though some have some pretty significant steps to be taken here (requirements gathering, user surveys, marketing analysis etc.)
  - But is the last step:
    - Handing off the design to be manufactured?
    - Dealing with manufacturing issues?
    - Supporting users of the design?
    - Dealing with end-of-life issues?

Design stages in this class (1/4)

- Identifying the purpose*
  - Identifying a problem
- Design requirements
  - What characteristics does the device need?
    - This should be things like “light-weight” or “easy to use” rather than “less than 8oz” or “iPhone-like interface”

*There is often a research step between this step and developing the design requirements. What are current solutions/workarounds? What do people really want? What is doable in this space?

Design stages in this class (2/4)

- Engineering specification
  - The design requirements turned into measurable outcomes.
    - “8 oz or less”
    - “New users can start measurement in less than 10 seconds on the average”
    - “48-hour battery life in the worst case”

Design stages in this class (3/4)

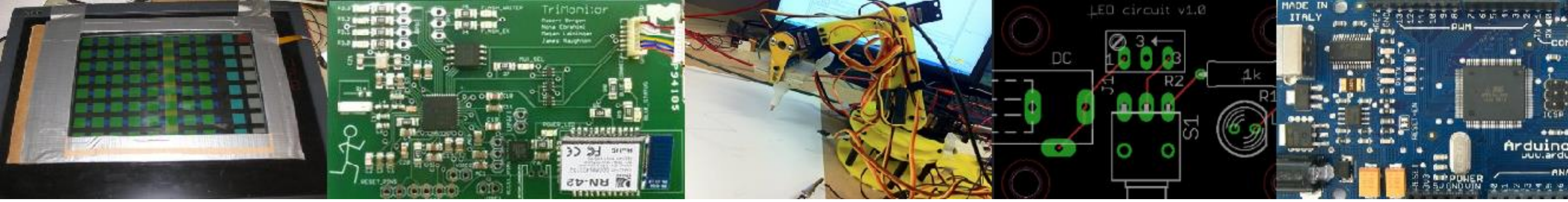
- Work out possible solutions
  - Identify a few ways to solve this problem
- Pick a solution
  - Find the one you like best.
- Prototyping
  - Building a prototyped device
    - Likely not the right form-factor etc.
    - Probably on a breadboard, but mechanical issues also need to be addressed.
  - Let's you see what's doable.
    - Also gives you a testbed to develop your solution

Design stages in this class (4/4)

- Implement your design
  - For us this involves ordering and assembling a PCB and getting your software up and running.
- Test and debug
  - Get everything working
  - Test to see if engineering requirements are met.

Design and this class

- This class is about getting a useful design done.
  - Following the steps of the design process helps a lot more than you might think.
    - Though we have such a time crunch that they will have to overlap a bit.



Creating good interfaces to hardware

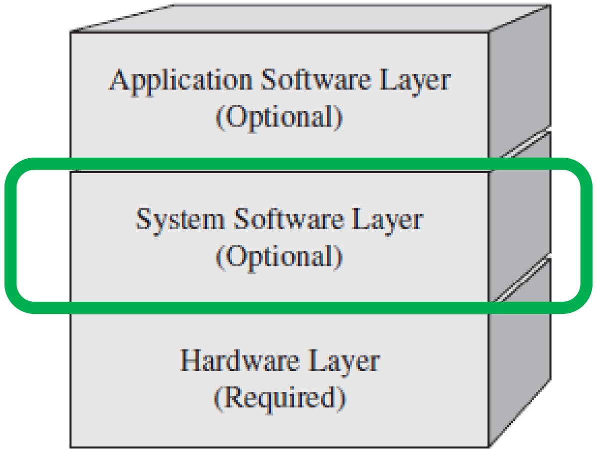


Figure 1-1
Embedded Systems Model.

Figure from *Embedded Systems Architecture* by Tammy Noergaard

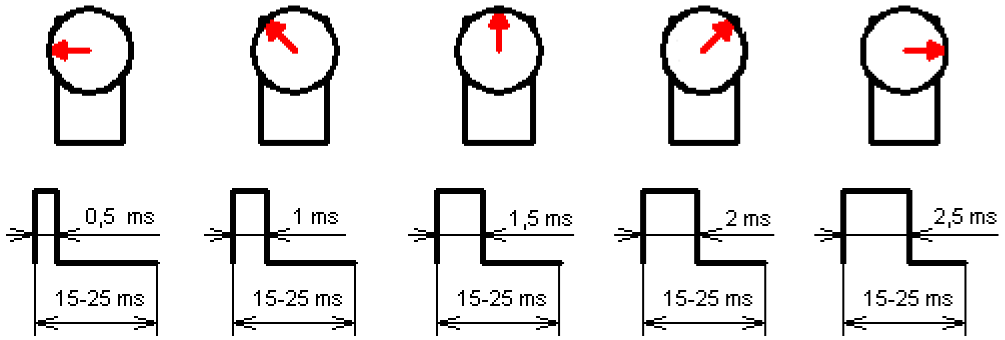
What is a hardware interface?

- A hardware interface is a set of functions, macros, or other programming constructs which allows the programmer to not worry about how exactly the hardware works.
  - It creates a level of abstraction so the programmer only needs to think about a subset of the problem.
  - This is creates a nice boundary where high-level code can be handed off.
    - Also extremely useful **even** when you are the only programmer!

Servos

- Pulse Width Modulation (PWM)
- We Provide:

+5VDC
PWM Signal
Ground



We Get:

0-180 Degree
Range of Motion

Talking directly to the servo

- In fact we probably aren’t talking to the servo.
  - Instead there is (hopefully) a timer that supports PWM.
    - We can specify a period by writing a register
      - or more likely a series of registers (prescalar, clock select, etc.)
    - We can specify the duty cycle in a similar way
      - Generally a single register where the duty cycle is in terms of clock ticks
    - We probably need to configure the timer to do PWM

What should the interface be for a servo?

- What I want you to do is to discuss:
  - What basic functions you want.
  - What the interface to those functions should be like.
  - Try to get a formal description of as much as you can.
  - You will have about 5 minutes.
  - Do it yourself—no web searches.

Discuss ideas

Things change…

- Might get a new servo
  - So period and duty cycle might be different
- Might get a new processor
  - So timer configuration might change.
- Might need additional functionality
  - Perhaps want to include stepper motors

…but in 373 we didn’t…

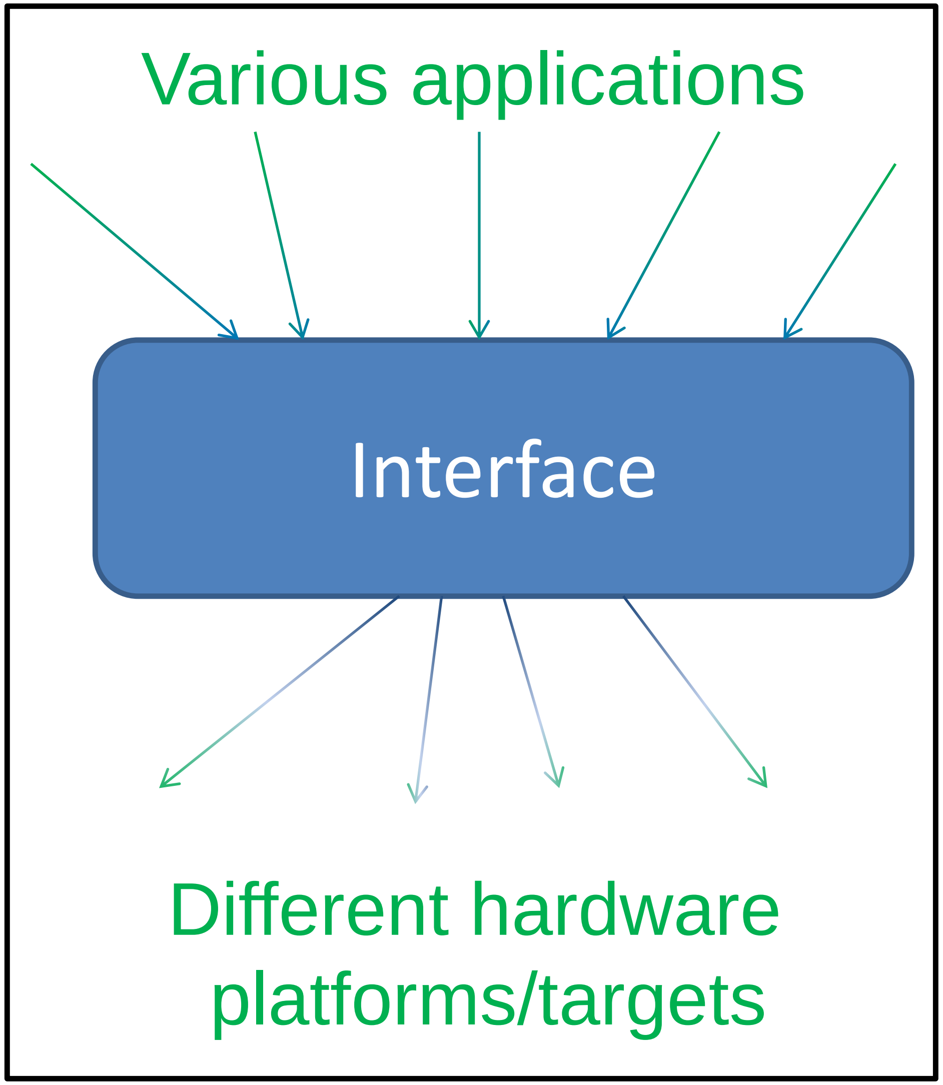
- Most of you didn’t create any meaningful interfaces in 373*
  - Exposed the low-level details to the programmer
    - After all you were the programmer and interface design takes time.
      - Plus you often don’t yet know what you’re going to need.
  - This makes it easy to do boneheaded things.
    - Wrong MMIO address, lots of replicated code, etc.
  - It also makes it hard to write good code.
    - You are worrying about too many things at once.
    - Keep worrying about things like bounds checking when interface should do that.

*Though some did. Often the more successful groups had well-defined and reasonably well-documented interfaces.

Stepping back

- Can think of an interface as a single way to talk to a class of hardware devices.
  - Each application uses the interface.
  - Each target "just" needs to support the interface.
  - What is the alternative?

- Now that we have a sense of what an interface is, let us look at what makes a good one.



Creating interfaces to hardware

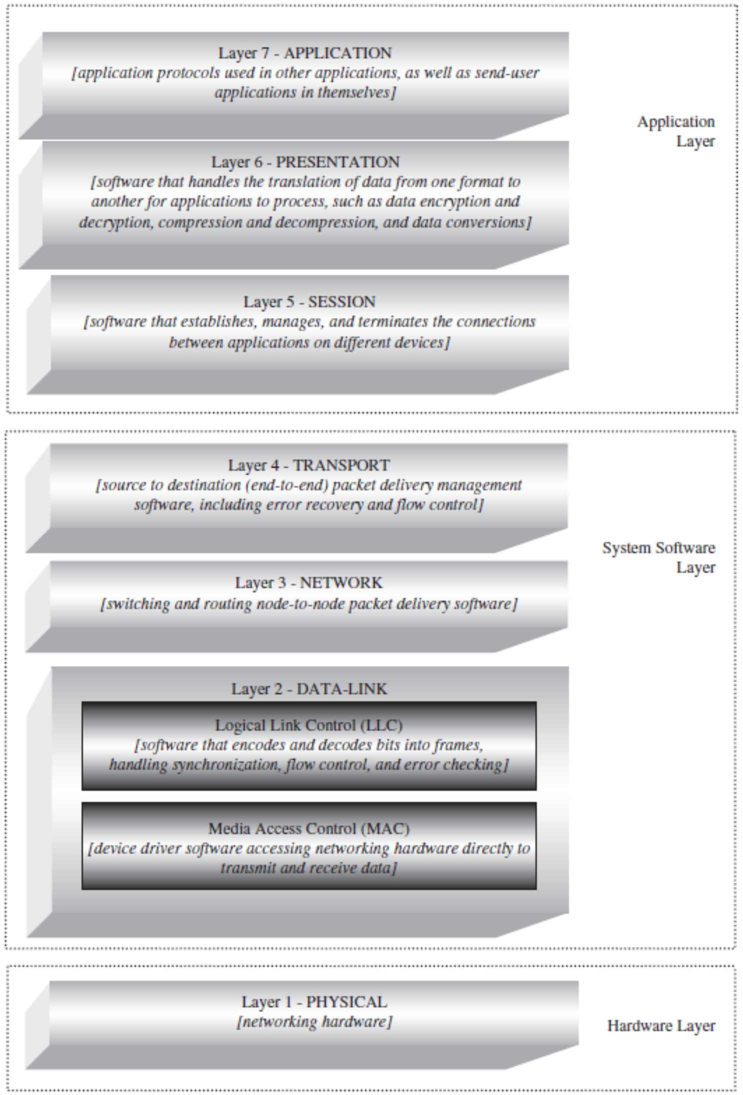
- A good hardware interface has three main goals:
  - It is easy to understand and use (useable)
  - It is efficient
  - It is portable to other hardware platforms.
- Those three things are often at odds.
  - And sometimes one matters a lot more than the others.
    - If the plan is to only use one hardware platform, portability matters little (though it matters a bit, as often plans change!)
      - Examples?
    - If the plan is that *you* are the only one who will use it, easy is less important.
      - *That* plan changes more than any IME
      - And easy is still powerful even if you are the only user.

What makes an interface easy to understand and use?

What does an interface have to do with efficiency?

- On the silly side:
  - One could imagine our servo interface having only a “turn 1 degree” function (direction specified)
    - Covers all functionality
    - But big turns require a lot of code to run.
- On the less silly side
  - If we use angles (in degrees) as the basis for the interface, that is going to require some math in the interface itself to convert to a register value.
    - Perhaps the programmer could skip a lot of that.
  - Other places we might see inefficiency for our servo?

Look at Open Systems Interconnect (OSI)


Layer 7 - APPLICATION
[application protocols used in other applications, as well as send-user applications in themselves]
Layer 6 - PRESENTATION
[software that handles the translation of data from one format to another for applications to process, such as data encryption and decryption, compression and decompression, and data conversions]
Layer 5 - SESSION
[software that establishes, manages, and terminates the connections between applications on different devices]
Application Layer
Layer 4 - TRANSPORT
[source to destination (end-to-end) packet delivery management software, including error recovery and flow control]
Layer 3 - NETWORK
[switching and routing node-to-node packet delivery software]
Layer 2 - DATA-LINK
Logical Link Control (LLC)
[software that encodes and decodes bits into frames, handling synchronization, flow control, and error checking]
Media Access Control (MAC)
[device driver software accessing networking hardware directly to transmit and receive data]
System Software Layer
Layer 1 - PHYSICAL
[networking hardware]
Hardware Layer

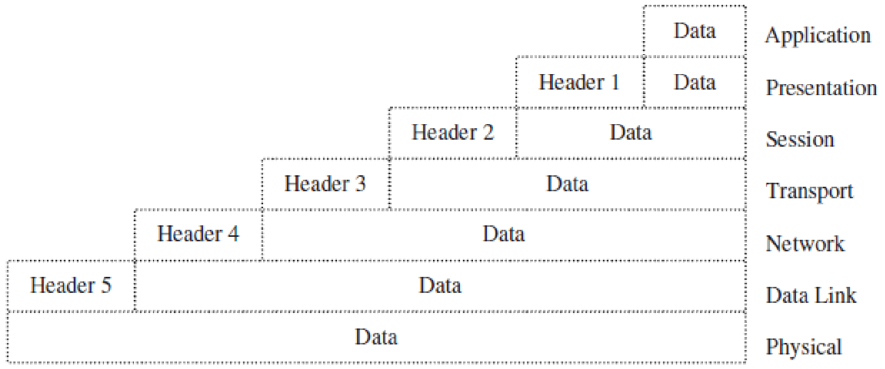


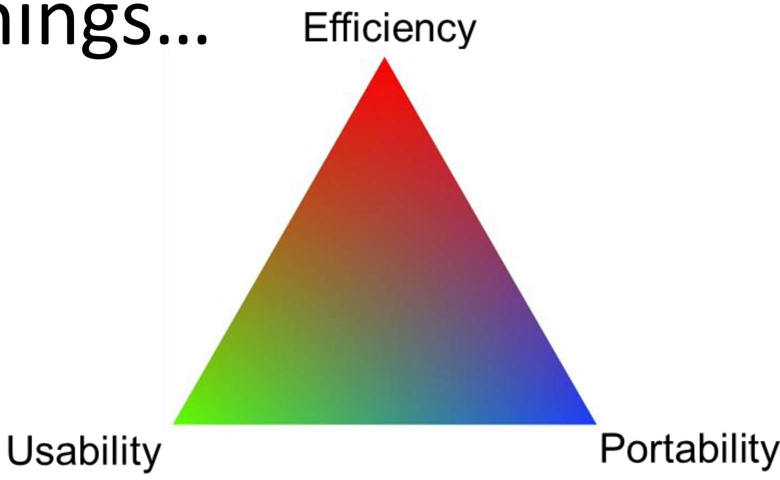
Figure 2-24
Header diagram.

This degree of layering is considered largely necessary, but it clearly creates a lot of overhead (inefficiencies)

Figures from *Embedded Systems Architecture* by Tammy Noergaard

So what makes hardware-interface design difficult?

- Mainly the three competing requirements of usability, efficiency and portability.
  - As discussed, they often fight with each other.
- But there are a few other things…



Software in parallel with hardware (1/2)

- When developing an embedded system, it is often necessary to select hardware and start on software in parallel.
  - Not just because of time constraints. Hardware side needs a solid understanding of software's needs before picking a processor
    - Memory and CPU requirements etc.
    - Certain peripherals might greatly reduce CPU needs
      - e.g. Don't need to bit-bang for a servo if you have PWM support. But if CPU otherwise idle, bit-banging might be okay.
    - Might want/need more than one processor
      - More on that later…

Software in parallel with hardware (2/2)

- Developing software, or even specifying interfaces, while selecting hardware is tricky.
  - Often some of the *functionality* will be impacted by hardware/component choices.
    - Can buttons detect level of pressure? Do we have multi-touch support? Does our processor have enough SPI busses that we can support a second ADC?
- But it’s common to do software design before hardware is selected.
  - Scheduling pressure
  - HR issues (software people are otherwise idle)
  - Hardware people need answers from software folks

Terminology

- There is a lot of terminology wrapped around hardware interfacing.
  - Terms like: HAL (hardware abstraction layer), Middleware, and Device Driver are all related to hardware interfacing.
    - And it's not unusual to see different people use those terms differently.
  - We'll try to take a look at that terminology later on.
    - I'll generally use "device driver" or "system software layer"

Summary

- It's a good idea to have at least one layer between applications and hardware.
  - Hardware interface
- Hardware interfaces have three main goals
  - Efficiency
  - Usability
  - Portability
- And those three things are often at odds.

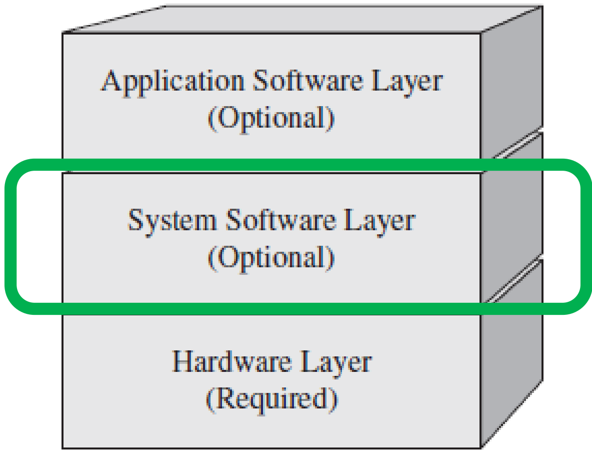
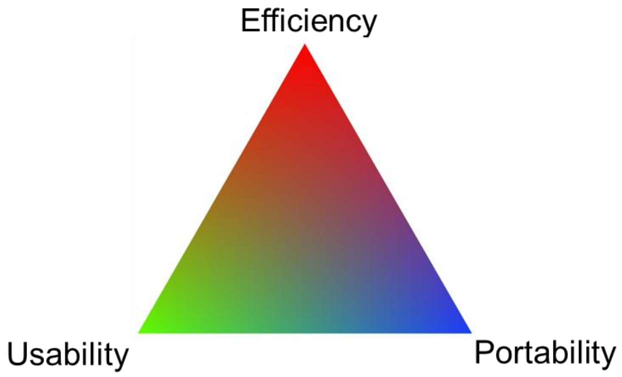
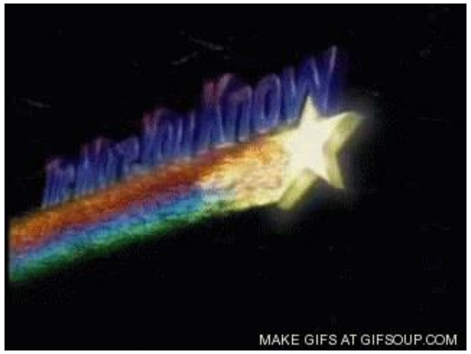

Application Software Layer (Optional)
System Software Layer (Optional)
Hardware Layer (Required)

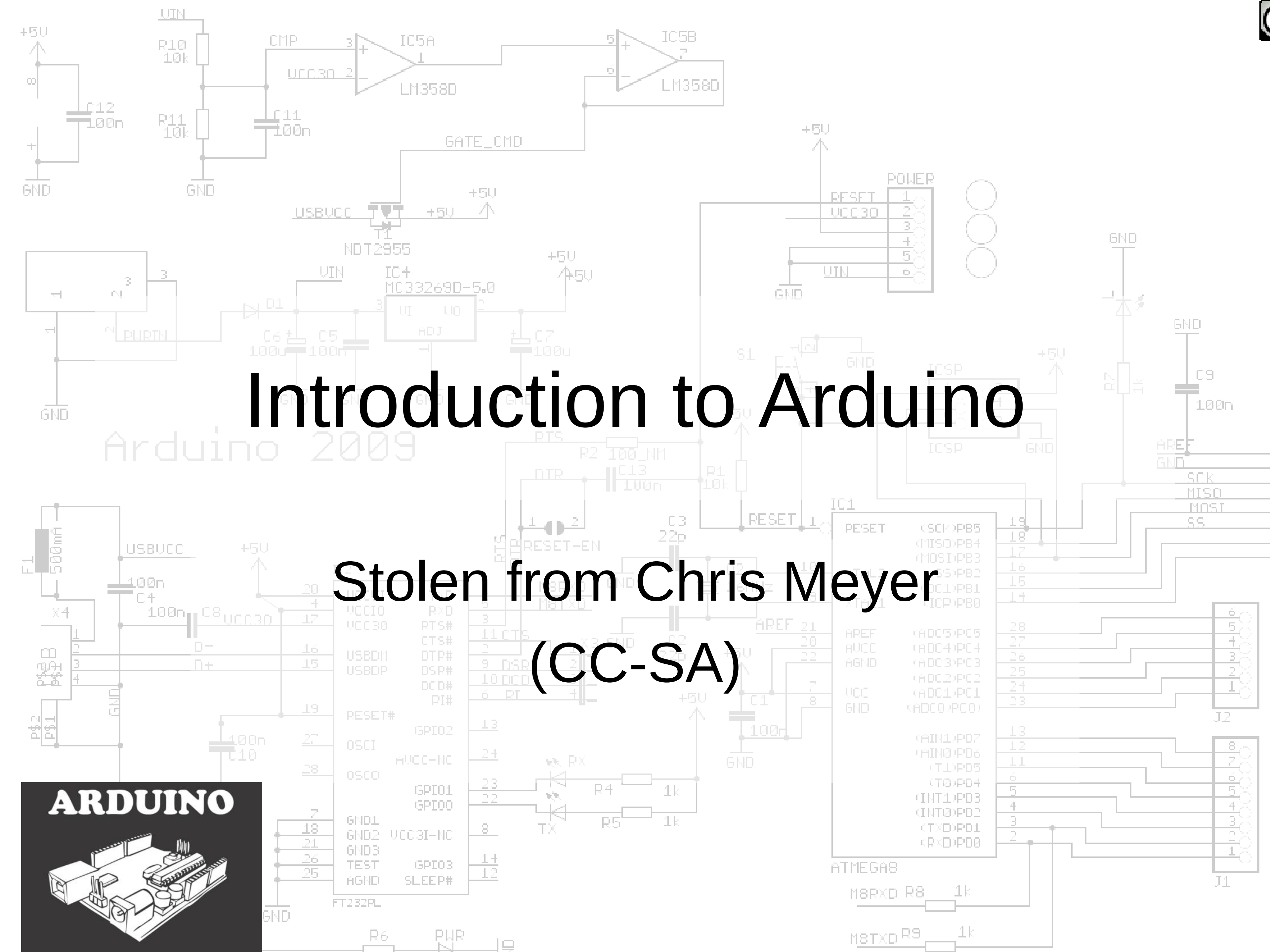
Figure 1-1
Embedded Systems Model.



Hardware/software co-design

- While a bit outside of the scope of this class, this is a good time to discuss hardware/software co-design
  - This phrase generically means "designing software and hardware a the same time"
  - But often used to describe automatic methods or tools to do the partitioning.
    - Might end up with an ISA/architecture for a CPU and application-specific code to run on it for example
      - HP has been doing this for printers for a while apparently
    - Might end up with three processors and code "done for you" once the problem is specified.
  - There is a lot of literature dating back decades on the topic.





Introduction to Arduino

Stolen from Chris Meyer

(CC-SA)

Arduino

ARRRR, like a pirate /

/ DWEE, just say "do we“ fast /

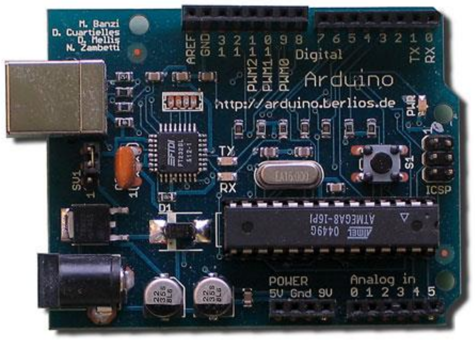
/ NO, as in no.

"ARRR-DWEE-NO"

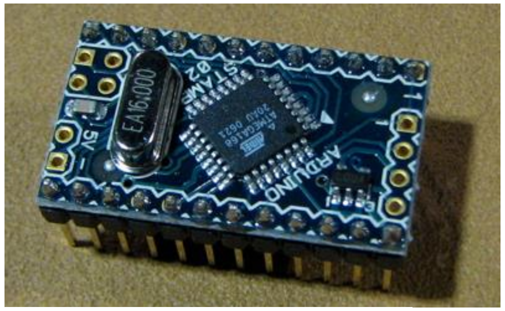
http://www.arduino.cc/cgi-bin/yabb2/YaBB.pl?num=1191602549%3Bstart=all

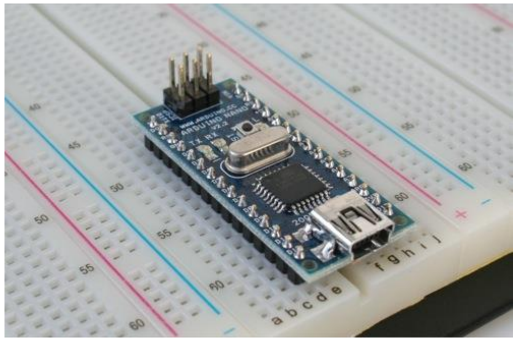
What is Arduino?

- Open Source Hardware Development Platform
- USB Programmable Microcontroller (MCU)



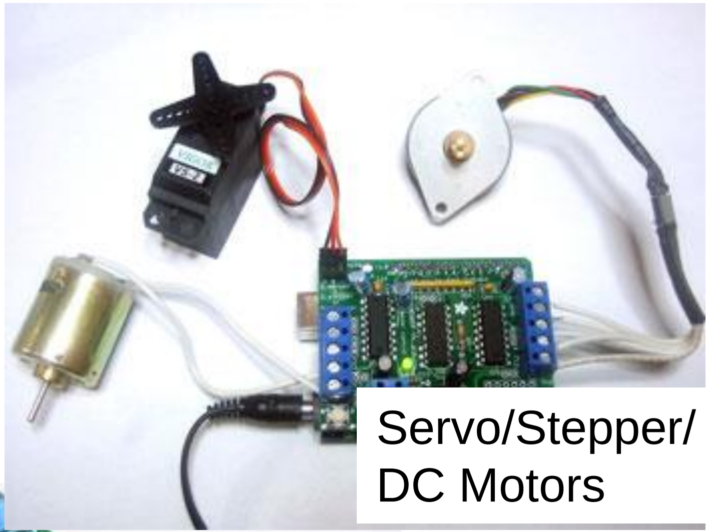
$30 Investment!

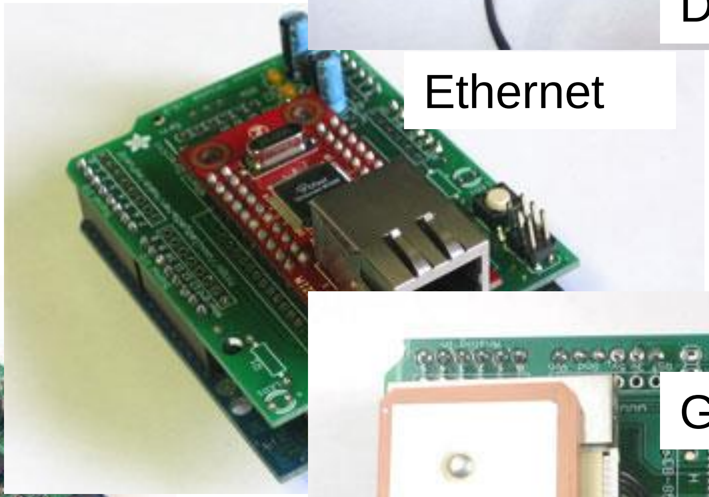


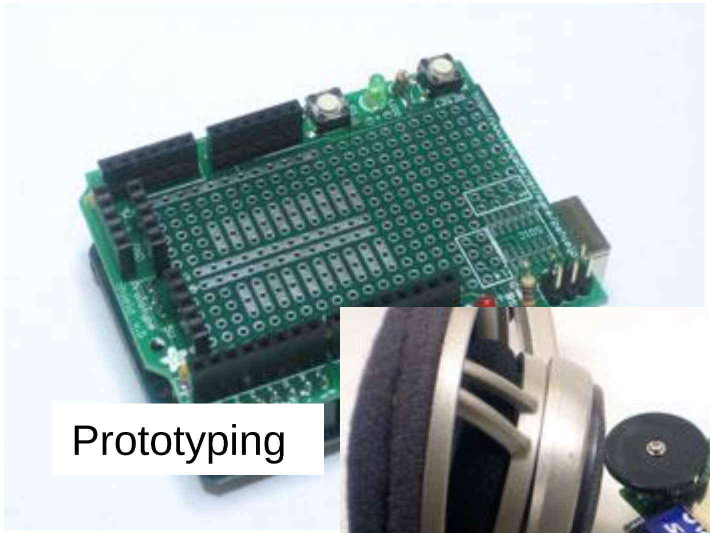


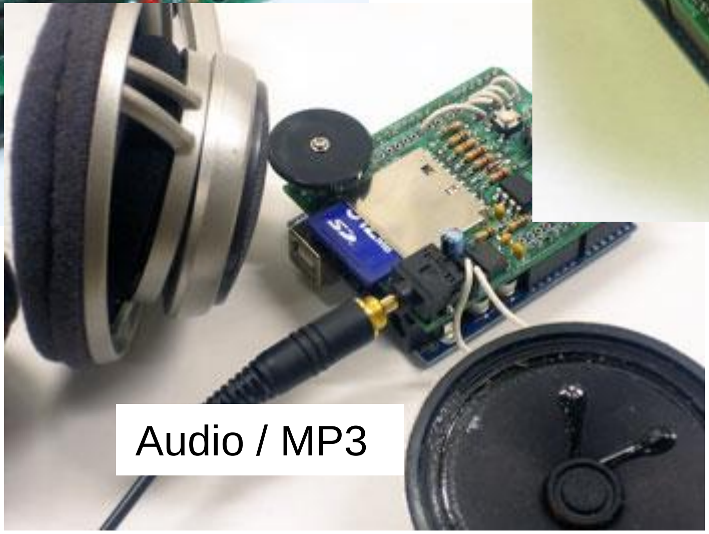
Shields?

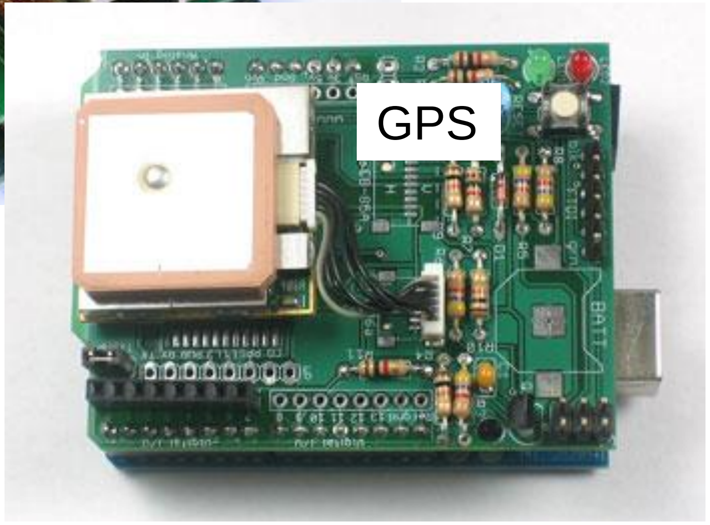
- Shields break-out/wire-up additional components to MCU



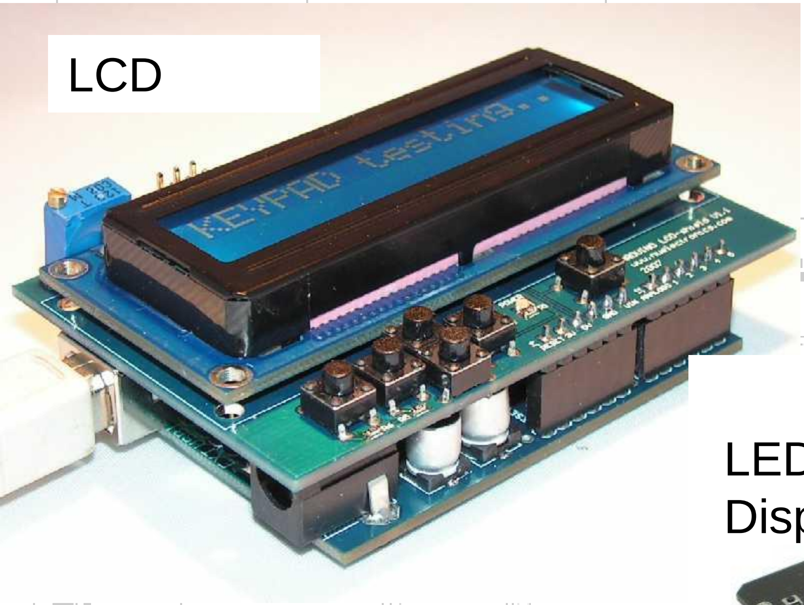


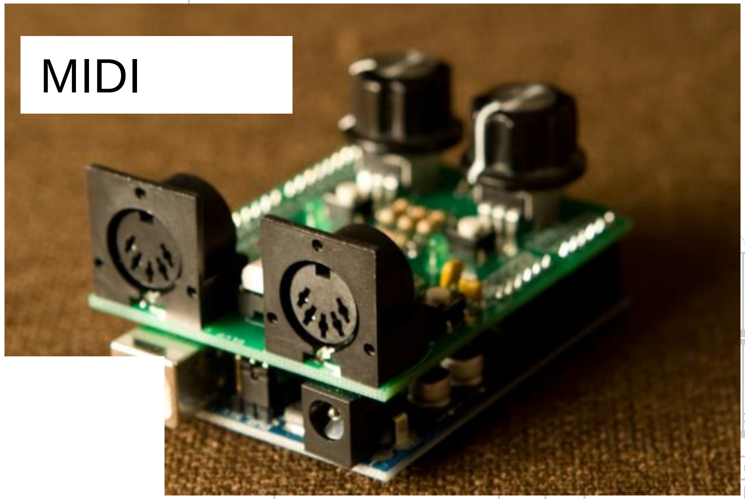




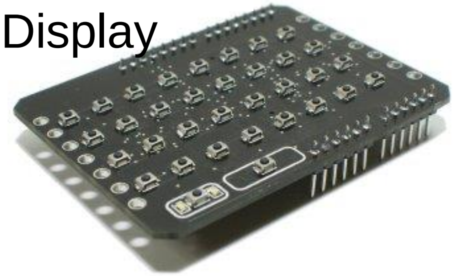


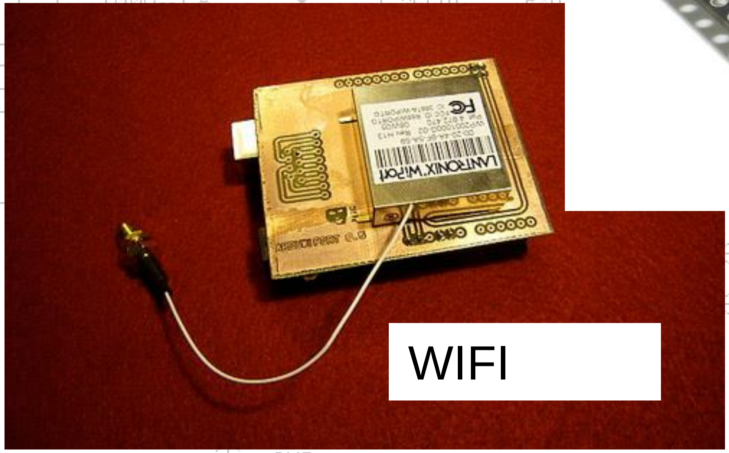
More Shields!

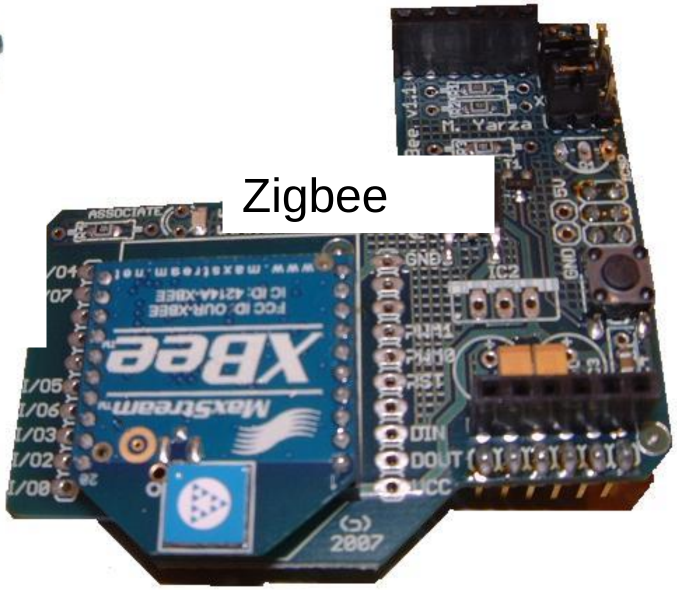

LCD


MIDI

LED Display

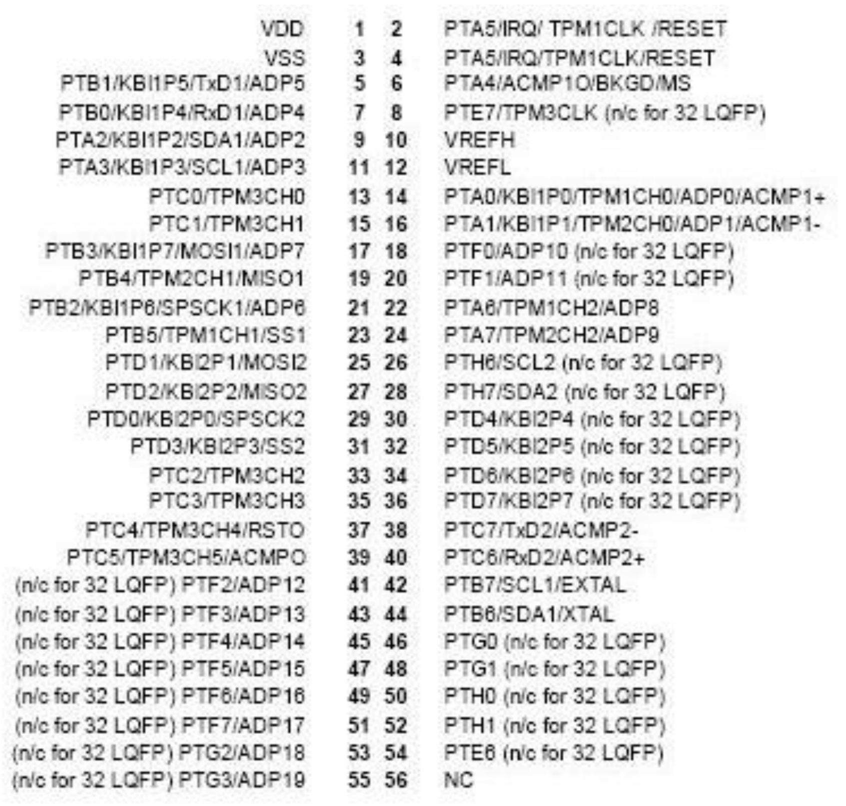



WIFI


Zigbee

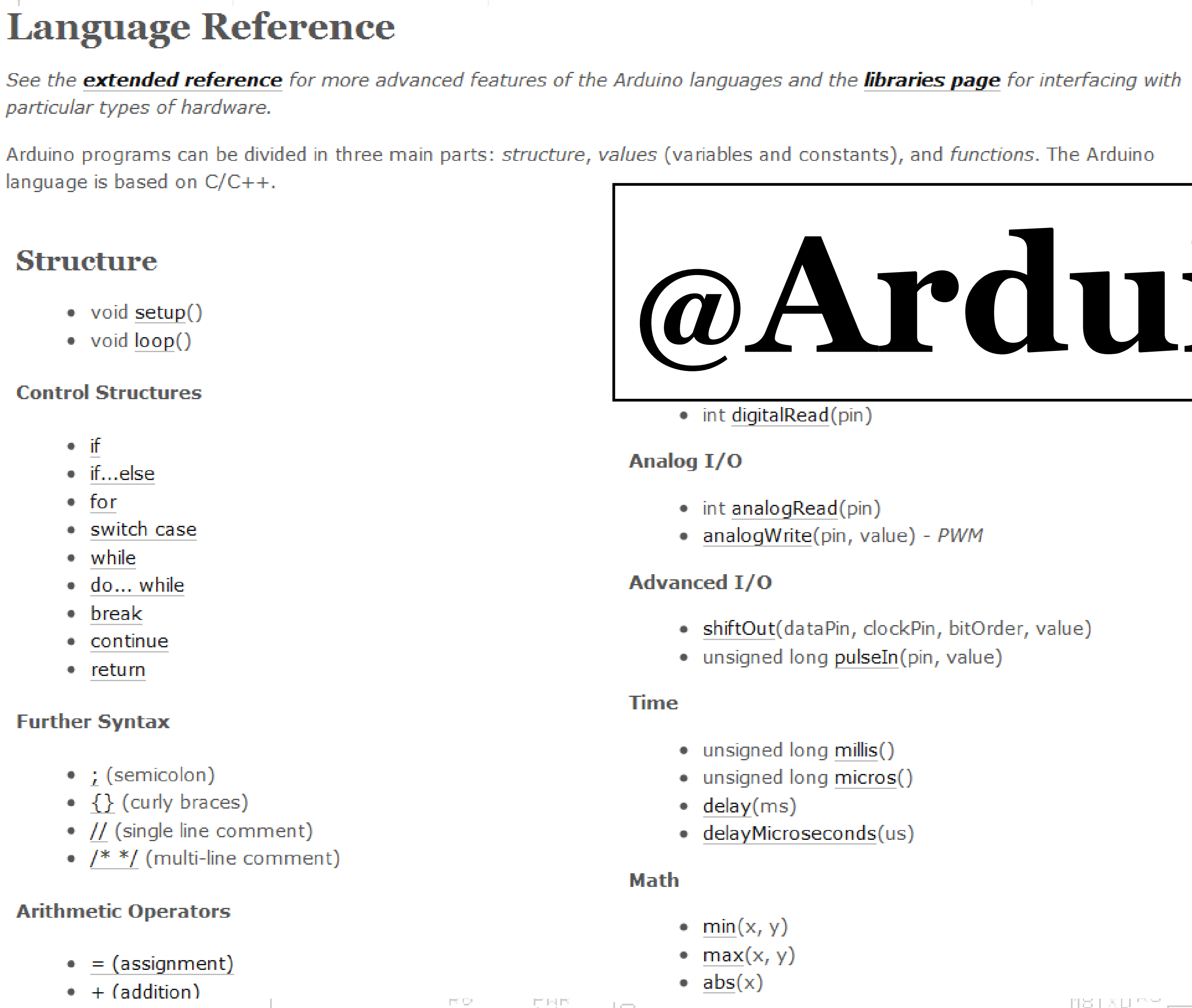
So What?

- Previously, MCU’s were very difficult to learn to use
- Required learning libraries, specialized protocols, timings, code minimization, 1,000+ page documentation



VDD	1	2	PTA5/IRQ/ TPM1CLK /RESET
VSS	3	4	PTA5/IRQ/TPM1CLK/RESET
PTB1/KBI1P5/TxD1/ADP5	5	6	PTA4/ACMP1O/BKGD/MS
PTB0/KBI1P4/RxD1/ADP4	7	8	PTE7/TPM3CLK (n/c for 32 LQFP)
PTA2/KBI1P2/SDA1/ADP2	9	10	VREFH
PTA3/KBI1P3/SCL1/ADP3	11	12	VREFL
PTC0/TPM3CH0	13	14	PTA0/KBI1P0/TPM1CH0/ADP0/ACMP1+
PTC1/TPM3CH1	15	16	PTA1/KBI1P1/TPM2CH0/ADP1/ACMP1-
PTB3/KBI1P7/MOSI1/ADP7	17	18	PTF0/ADP10 (n/c for 32 LQFP)
PTB4/TPM2CH1/MISO1	19	20	PTF1/ADP11 (n/c for 32 LQFP)
PTB2/KBI1P6/SPSCK1/ADP6	21	22	PTA6/TPM1CH2/ADP8
PTB5/TPM1CH1/SS1	23	24	PTA7/TPM2CH2/ADP9
PTD1/KBI2P1/MOSI2	25	26	PTH6/SCL2 (n/c for 32 LQFP)
PTD2/KBI2P2/MISO2	27	28	PTH7/SDA2 (n/c for 32 LQFP)
PTD0/KBI2P0/SPSCK2	29	30	PTD4/KBI2P4 (n/c for 32 LQFP)
PTD3/KBI2P3/SS2	31	32	PTD5/KBI2P5 (n/c for 32 LQFP)
PTC2/TPM3CH2	33	34	PTD6/KBI2P6 (n/c for 32 LQFP)
PTC3/TPM3CH3	35	36	PTD7/KBI2P7 (n/c for 32 LQFP)
PTC4/TPM3CH4/RSTO	37	38	PTC7/TxD2/ACMP2-
PTC5/TPM3CH5/ACMPO	39	40	PTC6/RxD2/ACMP2+
(n/c for 32 LQFP) PTF2/ADP12	41	42	PTB7/SCL1/EXTAL
(n/c for 32 LQFP) PTF3/ADP13	43	44	PTB6/SDA1/XTAL
(n/c for 32 LQFP) PTF4/ADP14	45	46	PTG0 (n/c for 32 LQFP)
(n/c for 32 LQFP) PTF5/ADP15	47	48	PTG1 (n/c for 32 LQFP)
(n/c for 32 LQFP) PTF6/ADP16	49	50	PTH0 (n/c for 32 LQFP)
(n/c for 32 LQFP) PTF7/ADP17	51	52	PTH1 (n/c for 32 LQFP)
(n/c for 32 LQFP) PTG2/ADP18	53	54	PTE6 (n/c for 32 LQFP)
(n/c for 32 LQFP) PTG3/ADP19	55	56	NC

Arduino makes it Easy!



Language Reference

*See the **extended reference** for more advanced features of the Arduino languages and the **libraries page** for interfacing with particular types of hardware.*

Arduino programs can be divided in three main parts: *structure*, *values* (variables and constants), and *functions*. The Arduino language is based on C/C++.

Structure

- void setup()
- void loop()

Control Structures

- if
- if...else
- for
- switch case
- while
- do... while
- break
- continue
- return

Further Syntax

- ; (semicolon)
- {} (curly braces)
- // (single line comment)
- /* */ (multi-line comment)

Arithmetic Operators

- = (assignment)
- + (addition)

- int digitalRead(pin)

Analog I/O

- int analogRead(pin)
- analogWrite(pin, value) - *PWM*

Advanced I/O

- shiftOut(dataPin, clockPin, bitOrder, value)
- unsigned long pulseIn(pin, value)

Time

- unsigned long millis()
- unsigned long micros()
- delay(ms)
- delayMicroseconds(us)

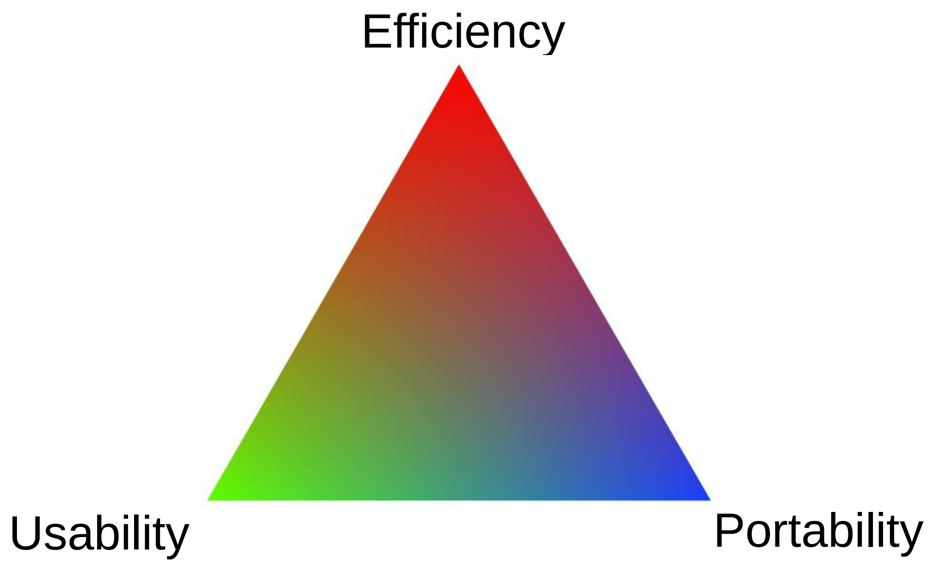
Math

- min(x, y)
- max(x, y)
- abs(x)

@Arduino.c

Question

If hardware interface design involves tradeoffs between three things, where in the triangle would you suspect Arduino was targeted?


Efficiency
Usability
Portability