ECE-304

L5 – Onto Build2

3/25/22

JDP Schedule Spring 2022

	Date	Class Meeting Topic	Due COB
Week 1	1/28/22	Lecture 1 - Course Intro	
Week 2	2/4/22	Lecture 2 - Project Specifications	collect kit1 2/11/22
Week 3	2/11/22	Q&A with the pretend customer	
Week 4	2/18/22	Lecture 3 - Arduino & Sonar Demo	Build1 PDR report due
Week 5	2/25/22	Lecture 4 - Test plans, issues	
Week 6	3/4/22	No Class	Built1 Test plan & EVM1
Week 7	3/11/22	Build1 Report Due, no Class Meeting	Build1 report
	3/18/22	Spring Break	
Week 8	3/25/22	Lecture5 - avr-gcc tools demo	
Week 9	4/1/22	Lecture 6 - Risk Mgt, Q&A with pretend customer	
Week 10	4/8/22	Build2 PDR Due, no Class Meeting	Build2 PDR report due
Week 11	4/15/22	Senior Design Project (SDP) Topics	EVM2
Week 12	4/22/22	Senior Design Project (SDP) Topics	
Week 13	4/29/22	Build2 Report Due, no class meeting	Build2 report

ECE397A

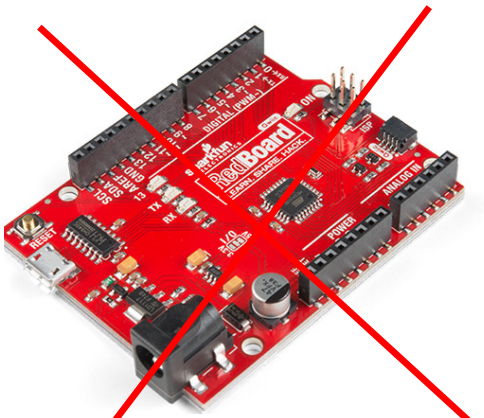
ECE397B

ECE304

build2

- Key requirement:
  - use bare ATmega328P IC on breadboard with battery & ANSI C
  - No Arduino Hardware or Scripts:

```
void setup(){
    Serial.begin(9600);
    pinMode(TRIG, OUTPUT);
    pinMode(ECHO, INPUT);
    pinMode(LED, OUTPUT);
}
void loop(){
    digitalWrite(LED, LOW);
    digitalWrite(TRIG, LOW);
    duration=pulseIn(ECHO, HIGH);
}
```



and now, for something completely different...

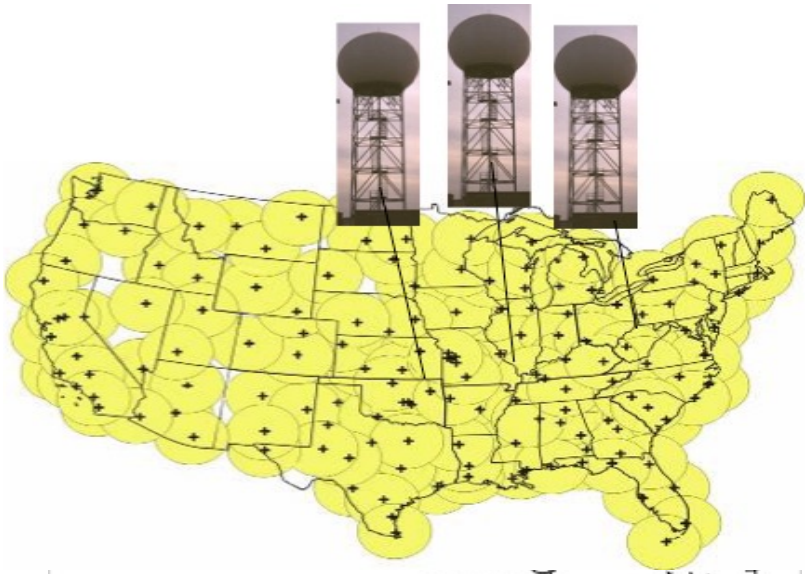
NextGen – what should the next generation US national aircraft & weather surveillance radar network look like?

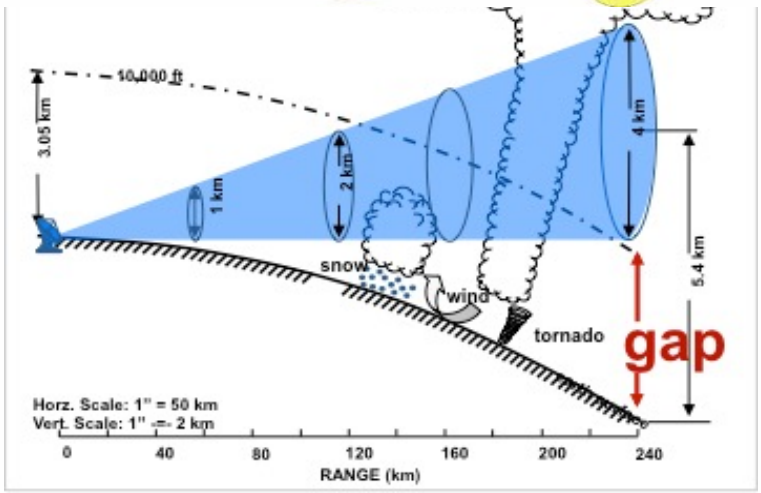
Which missions/functions should it have?
Detect aircraft in terminals?
Detect aircraft en-route?
Detect drones (un-piloted airborne systems)
Detect weather?
Detect tornadoes?
Detect incoming missiles far from the coastline?

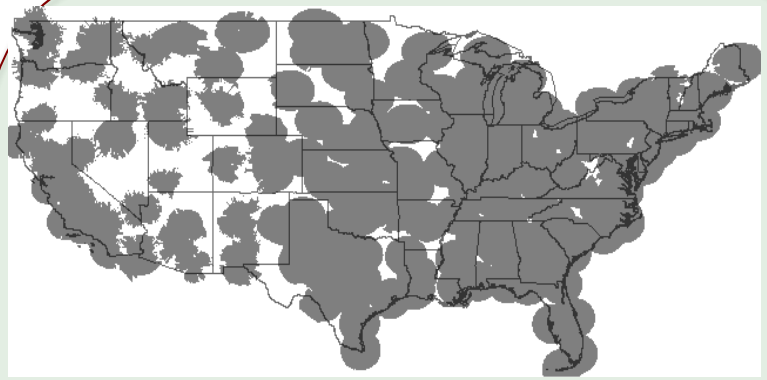
How well? What performance?
Sensitivity to detect small plastic drones?
Aircraft flying close to ground? (low-fliers)
Small tornadoes on the ground?
Detect high-altitude aircraft?

This is a several 10's of billion $ undertaking.
Need to establish the key performance requirements.

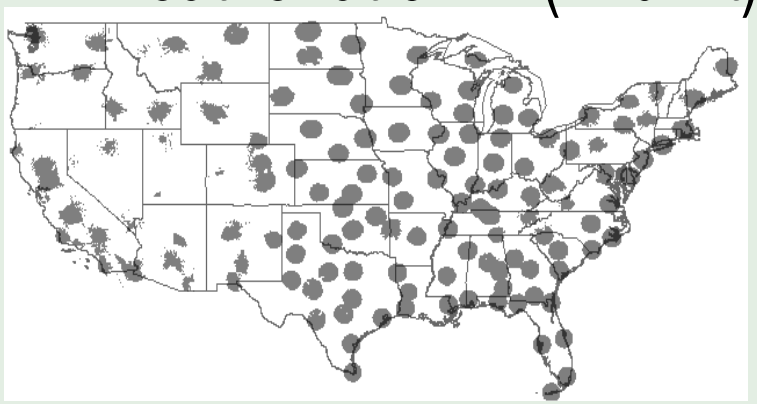
Today's Weather Surveillance – USA S Band







Weather at 3 km (~10k ft)



Weather at 1 km (~3200 ft)

widely-spaced radars, low-altitude coverage gaps

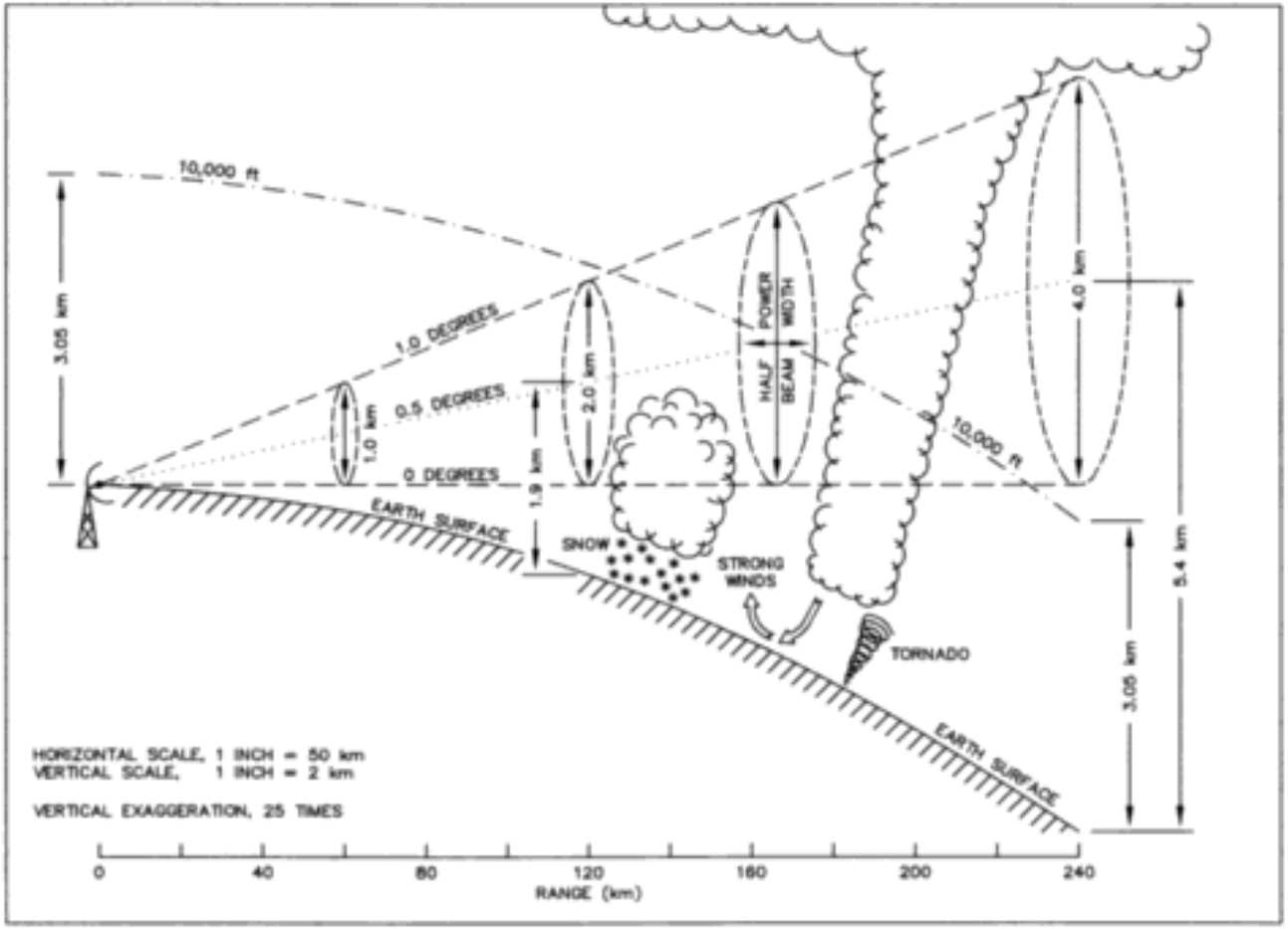


Figure A-3 Diagram illustrating the effect of range and earth curvature (with standard atmospheric refraction) on NEXRAD cross-beam resolution and coverage of low-level weather phenomena. Courtesy of SRI International.

what is the effective max range of a radar?

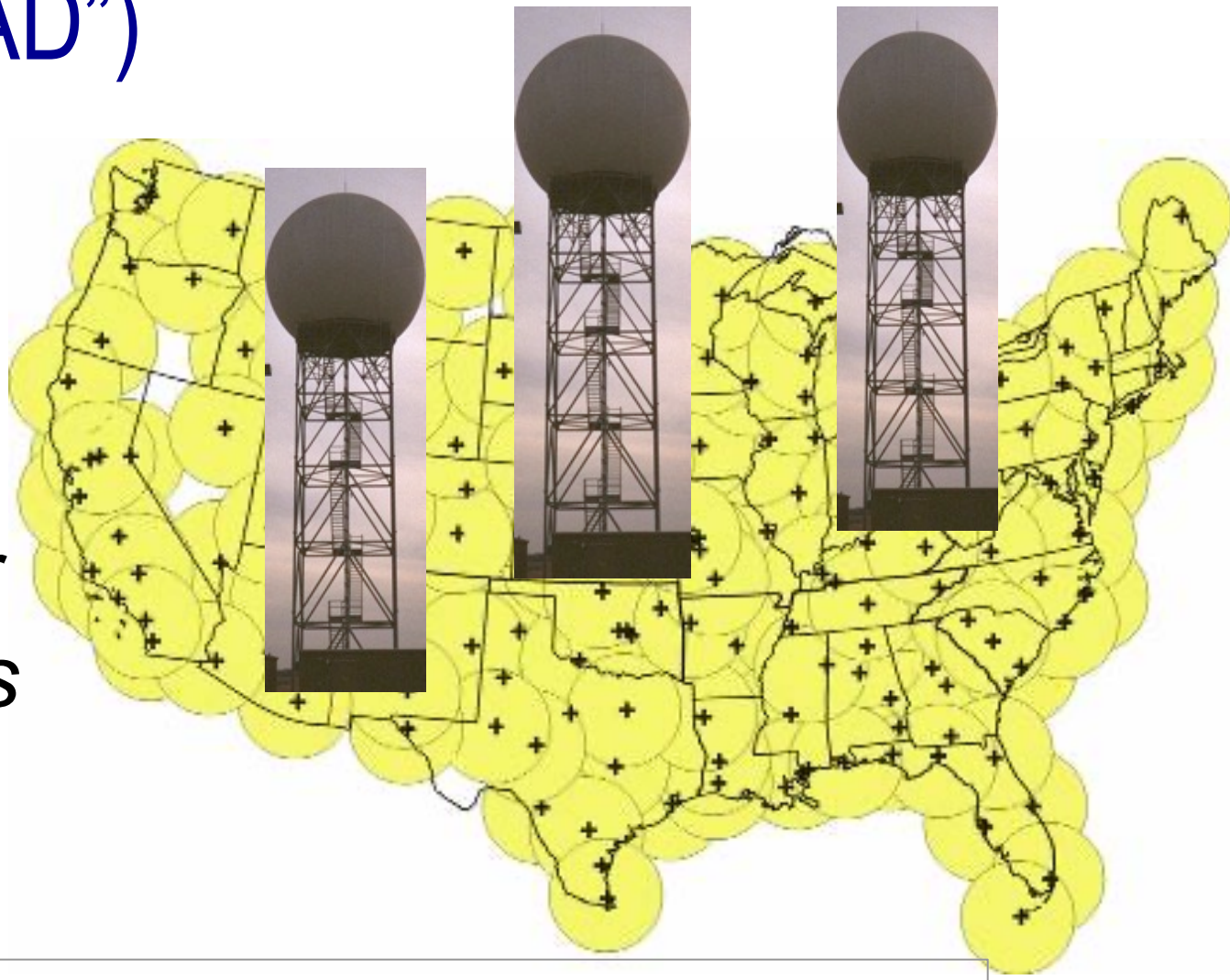
how small can raindrops be at max range?

how low a coverage floor can we tolerate?

similar questions for aircraft surveillance

Today's Infrastructure: WSR-88D (aka "NEXRAD")

150 radars
~ 230km apart
$10M/radar
x 150 radars
= $1.5Bn



Today's requirement: Comprehensive coverage of weather > 10,000' above ground level

What about aircraft?

Should we build large multifunction radars or single function radars?

What about the low-level coverage?

Should we build thousands of small radars?
What about coverage over the oceans?

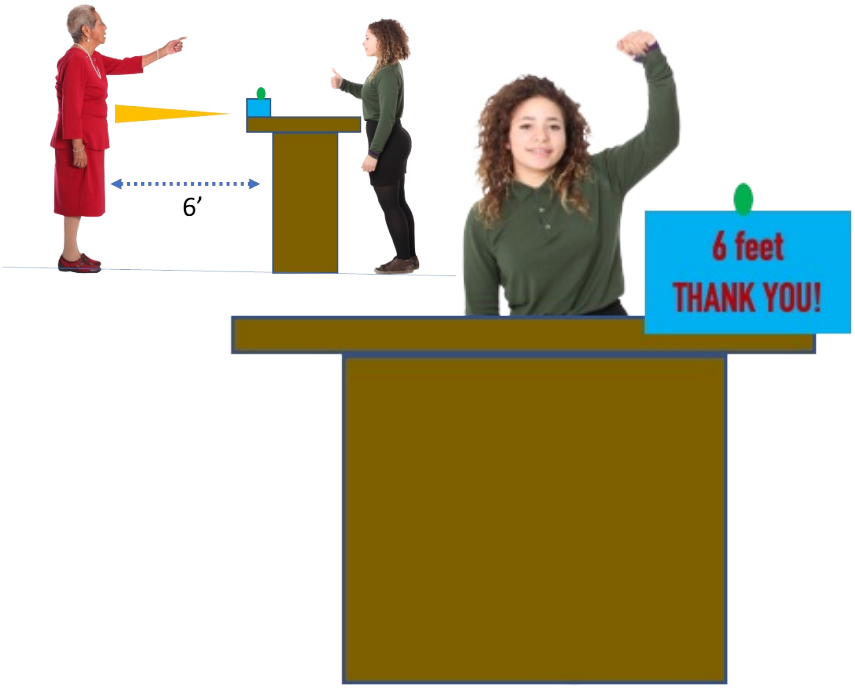
What is the correct set of functional, performance, and other requirements? (security, public data access, health & safety, reliability, ...)

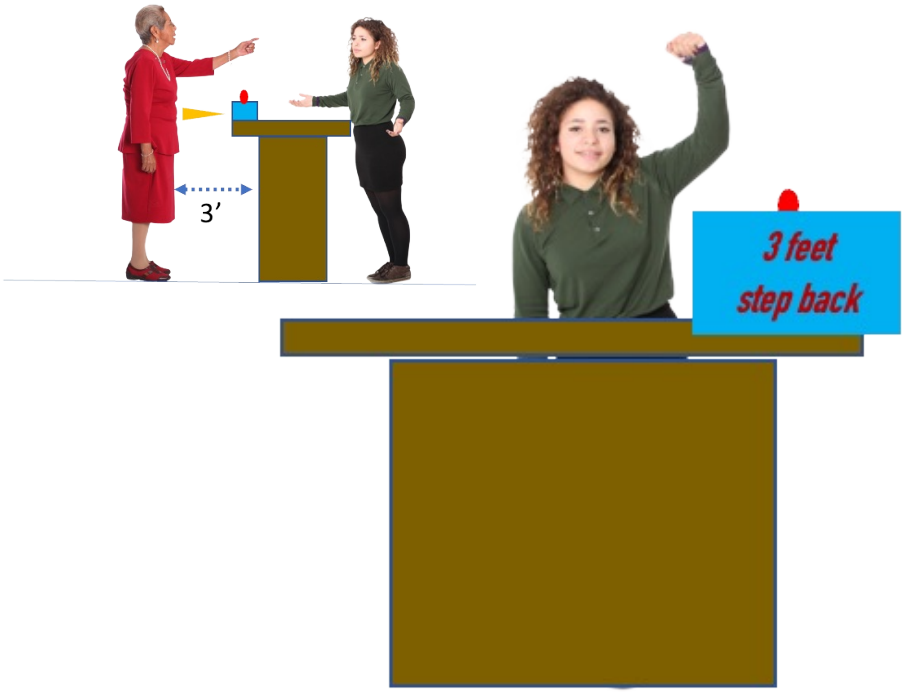
WE DON'T KNOW YET.

LOTS OF RESEARCH & EXPERIMENTS with **multiple prototype builds** BEING UNDERTAKEN TO TRY AND FIGURE THIS OUT.

THIS IS HOW MUCH NEW TECH GETS DEVELOPED.

Safe Reception Distance System (SRDS)





I (in my roles as customer & your teacher) have not built this system. And I do not intend to build it. That’s your job.

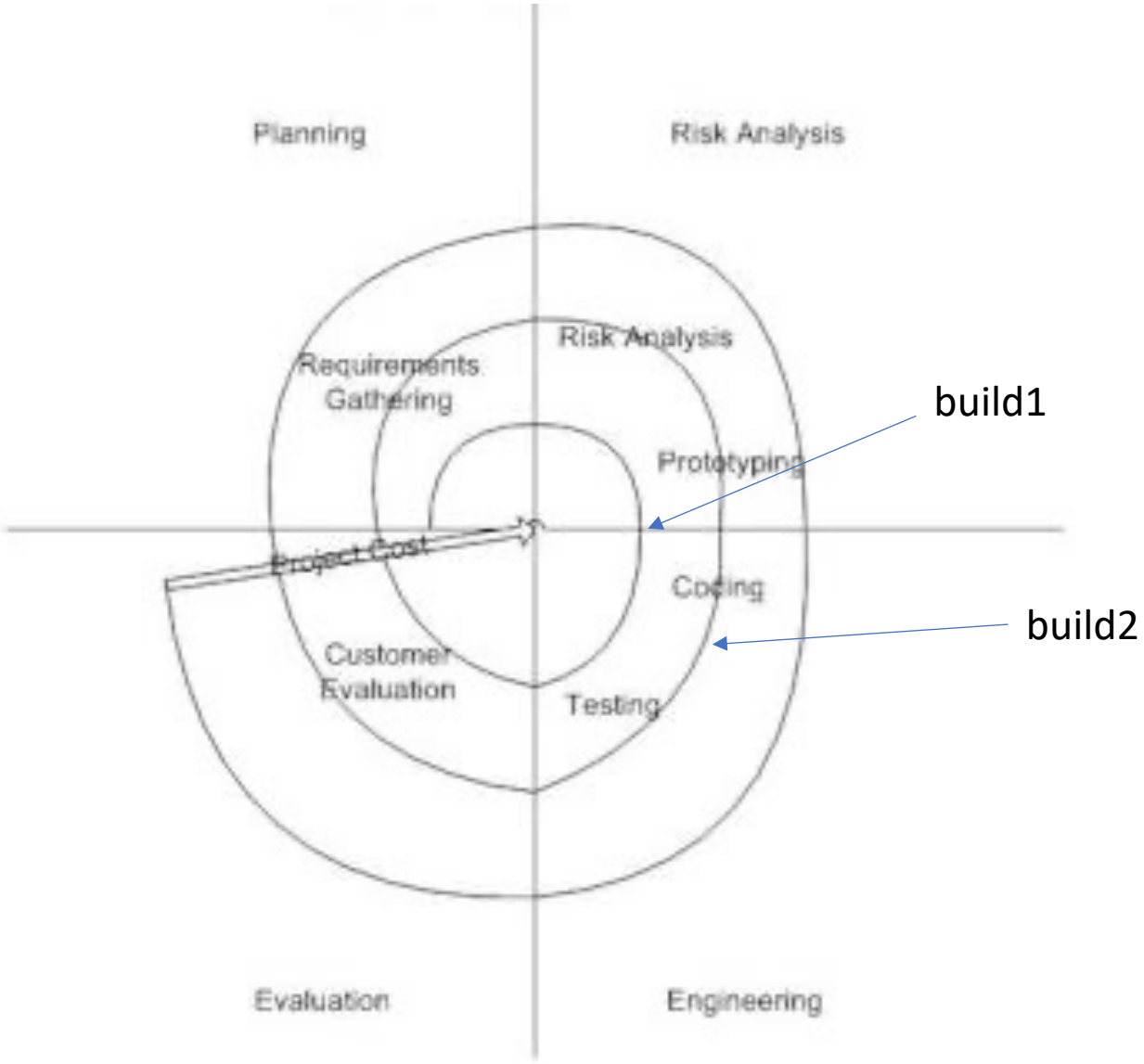
I don’t know if the initial set of requirements are the right ones or not. We will discover that as we proceed through a spiral engineering process.

Let's review our process:

- ❑ Initial statement of requirements/specifications (my best guess at functional & performance needs/requirements)
- ❑ Q&A with the customer (this is a KEY EVENT!) – miss it at your peril
- ❑ You did 1st level of design
- ❑ You submitted your PDR
  - ❑ your commitment to requirements
  - ❑ your initial guess at a block diagram
- ❑ Feedback on PDR & resubmit, as needed
- ❑ You build it. Figured out how to test & verify requirements. Figured out how long it takes to achieve functionality.
- ❑ You submitted your verification report
- ❑ Now we will repeat the whole process, having the benefit of experience...

Go back and re-read the Dog-House case study.

Spiral Model of Engineering Development



Characteristics of Good Requirements

Well-written requirements are essential for project success, including the following key characteristics.

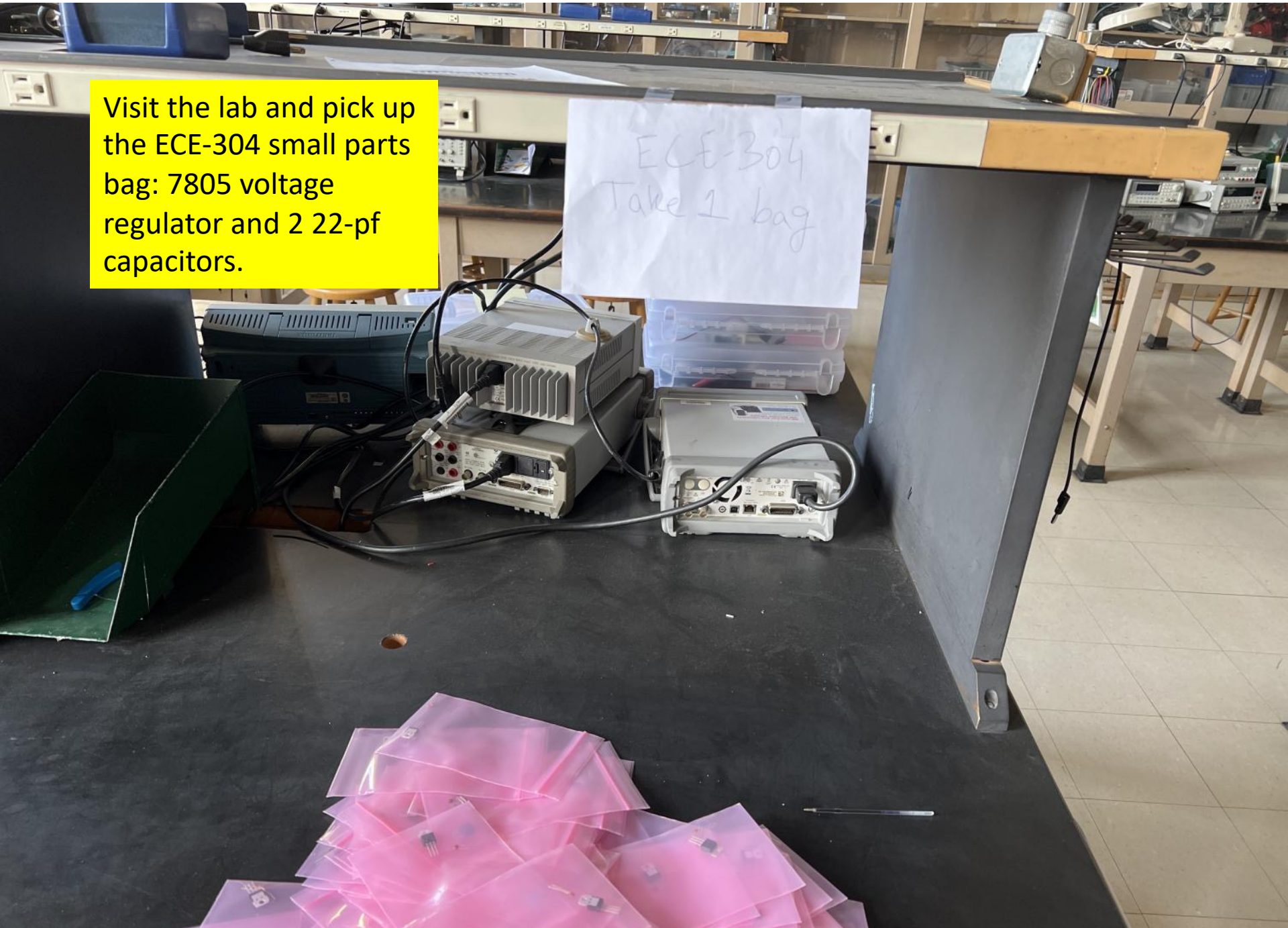
- **Necessary.** State only what is determined to be necessary for achieving the client's mission.
- **Clear.** Convey what must be achieved in a manner that can be understood by those who are expected to implement the requirement, without having to ask the author what was meant.
- **Achievable.** Confirm with the implementer that the requirement can be affordably achieved either by previously developed means, or within a reasonable period of development.
- **Traceable.** Ensure derived requirements can be traced to a user need or a higher-level specification.
- **Verifiable.** Requirements must be stated in a manner that compliance can be objectively confirmed. Typical methods of confirmation include *analysis, inspection, demonstration and test*.
- **Complete.** A set of requirements needs to be complete, such that if all are met then the resulting system will successfully achieve the client's need for the system. In addition, the needs of other stakeholders will be addressed to the agreed extent, and the regulations will be met.
- **Implementation Free.** State *what* is required and *how well* it needs to be done without bias for *how* it will be done. The design team should be allowed to choose the best means of accomplishing the requirements. This helps to provide stable requirements and to control cost.

Semester Schedule

JDP Schedule Spring 2022			
	Date	**Class Meeting Topic**	**Due COB**
Week 1	1/28/22	Lecture 1 - Course Intro	
Week 2	2/4/22	Lecture 2 - Project Specifications	collect kit
Week 3	2/11/22	Q&A with the pretend customer	
Week 4	2/18/22	Lecture 3 - Arduino & Sonar Demo	Build1 PDR report due
Week 5	2/25/22	Lecture 4 - Test plans, issues	
Week 6	3/4/22	No Class	Built1 Test plan & EVM1
Week 7	3/11/22	Build1 Report Due, no Class Meeting	Build1 report
	3/18/22	Spring Break	
Week 8	3/25/22	Lecture5 - avr-gcc tools demo	
Week 9	4/1/22	Lecture 6 - Risk Mgt, Q&A with pretend customer	
Week 10	4/8/22	Build2 PDR Due, no Class Meeting	Build2 PDR report due
Week 11	4/15/22	Senior Design Project (SDP) Topics	EVM2
Week 12	4/22/22	Senior Design Project (SDP) Topics	
Week 13	4/29/22	Build2 Report Due, no class meeting	Build2 report

What's next?

- Next week: Risk Analysis & Q&A With the pretend customer. Your chance to ask clarifying questions about build2 requirements
- 2 Weeks: PDR2: Problem Statement, System Specification, & Block Diagram for your system
  - problem statement ~ a paragraph
  - system specification ~ 7-10 requirements (design agnostic) [You then build to these specs. They don't change]
  - block diagram – This will change & evolve. Draw with a computer, not hand-drawn.


Visit the lab and pick up the ECE-304 small parts bag: 7805 voltage regulator and 2 22-pf capacitors.
ECE-304
Take 1 bag

going forward: SOLO VS TEAM OPTION

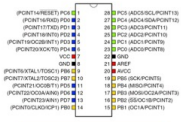
- Solo:
  - **a table-top demo is OK**
  - **emphasis on functional requirements (ok to miss some performance requirements)**

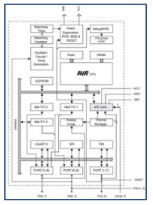
- Team option:
  - **full-scale system deployed on a counter**
  - **emphasis on both functional and performance requirements**

Each team of 2-3 develops, builds, tests 1 system
everyone does same amount of work on team

Programming the ATmega328P Chip

1. Develop source code (typically in C) using **header files** for the **target** MCU (ATmega328P)
2. Compile source code onto the **target** MCU, link with pre-compiled library objects (this is called **building** the application)
3. Flash resulting binary code onto the target MCU (this is called **programming** the target)
4. Debug, repeat, often using a dev board
5. Finally, deploy the code & MCU IC in the embedded system





- Machine Language Instruction set
- Registers
- External clock
- Functional block diagram
- Extensive Data sheet (100's pages)
- Everything inside happens in binary (1's & 0's) at a speed driven by a clock, using program counter, & other key elements

The Embedded Programming Super-Loop Architecture

```
int main(void)
{

        initialize();                   // call function to set things up

        while(1)                        // loop forever
        {
                doSomething();          // application task
                doSomethingElse();      // another application task
        }

        return 0;                       // execution never gets here
}
```

adapted from https://www.includehelp.com/c/the-super-loop-architecture-for-embedded-c-programming.aspx

ATmega328P MCU (28 or 32 pin package)

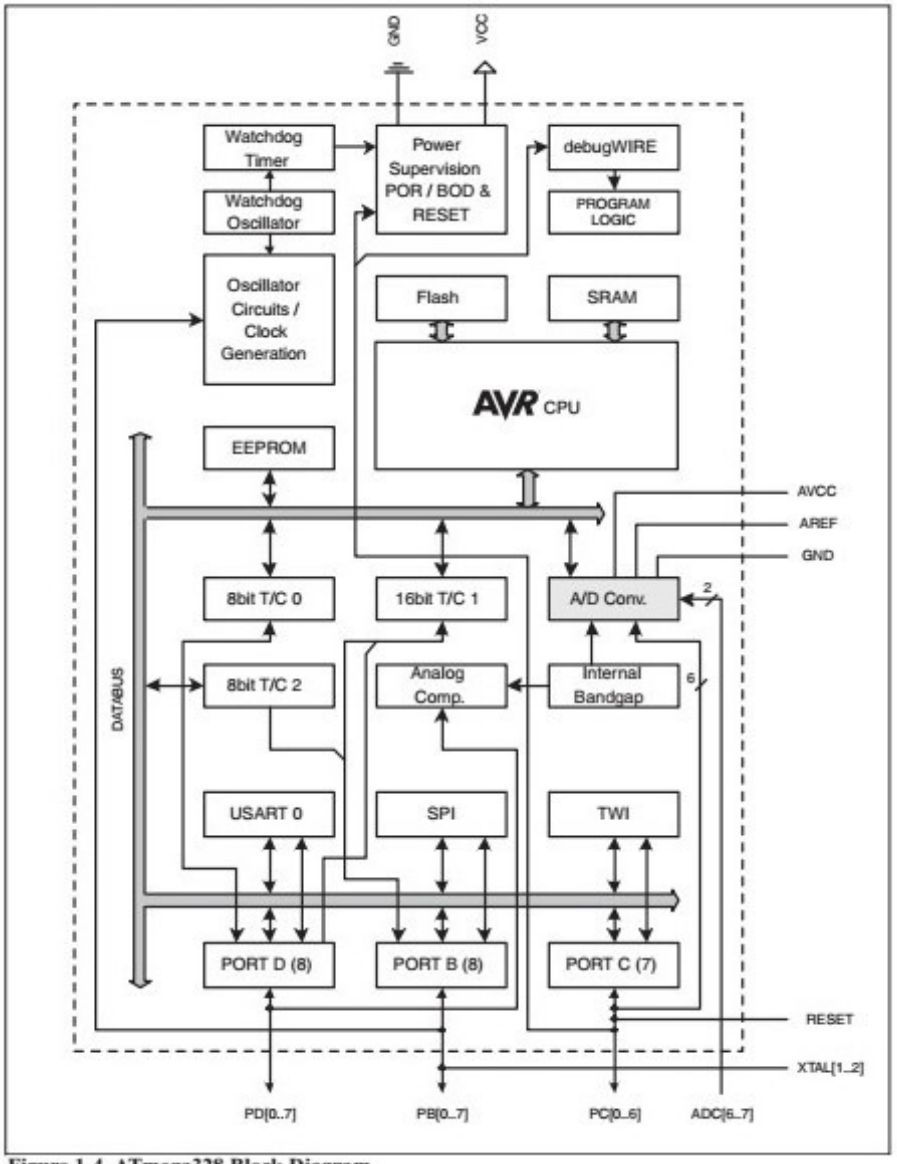
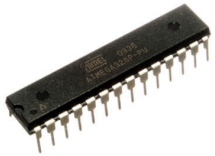


Figure 1-4. ATmega328 Block Diagram

$3.51 Digikey

28 pin
PDIP



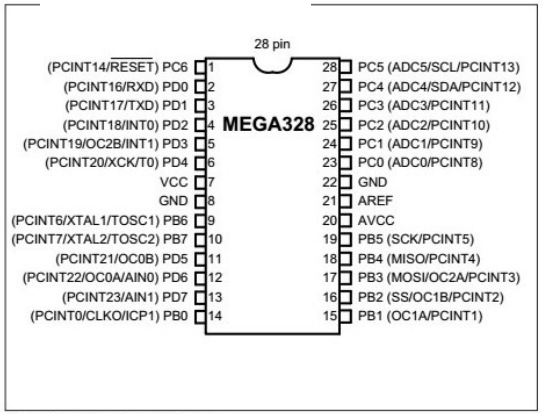


Figure 4-1. ATmega328 Pin Diagram

- 23 General Purpose I/O (GPIO) pins
- 8 Analog to Digital Converter (ADC) pins
- 3 Timers
- USART for serial communication

- 32K Byte Flash ROM for Code
- 2K Byte data RAM
- 1K Byte data EEPROM

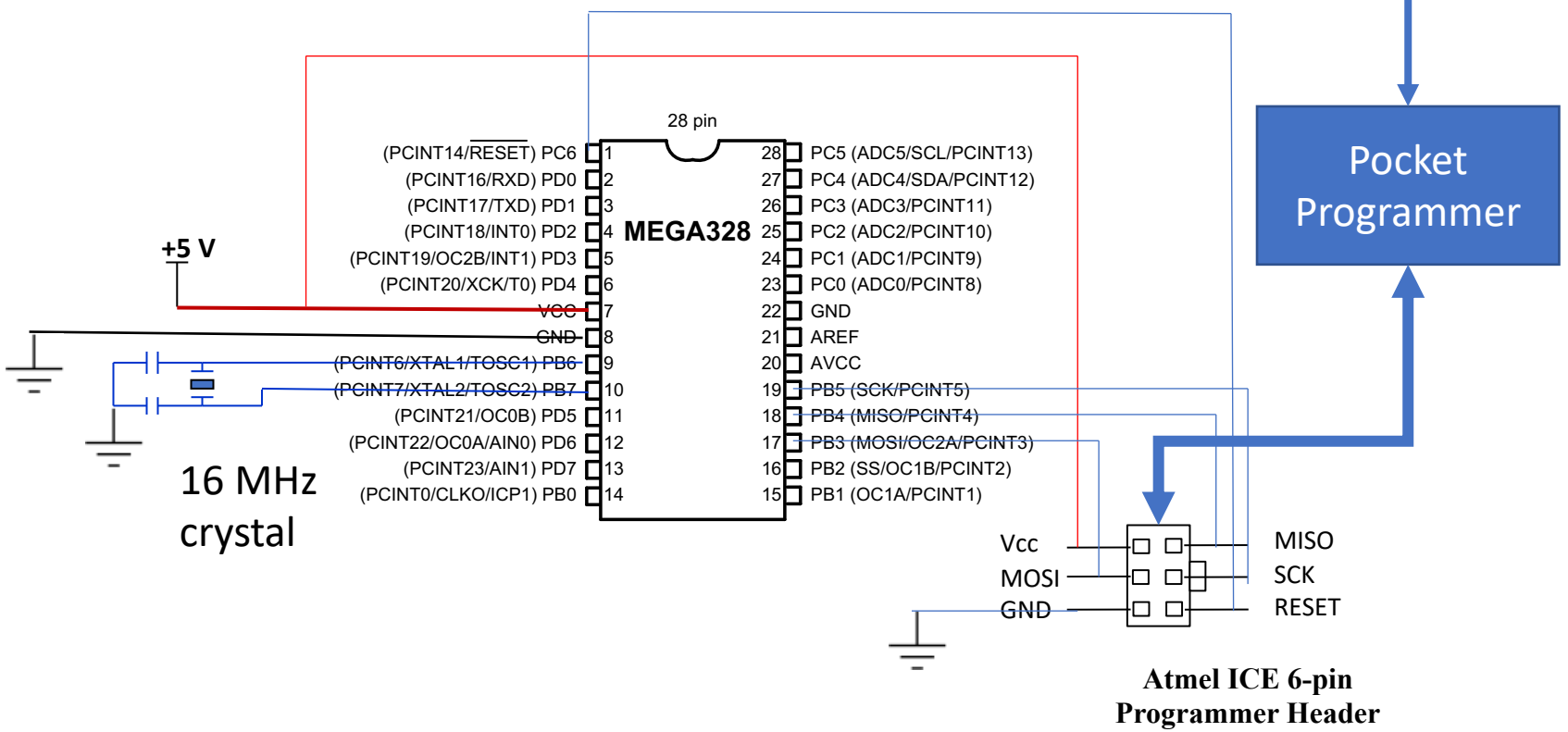
Program bare MCU using a external programmer

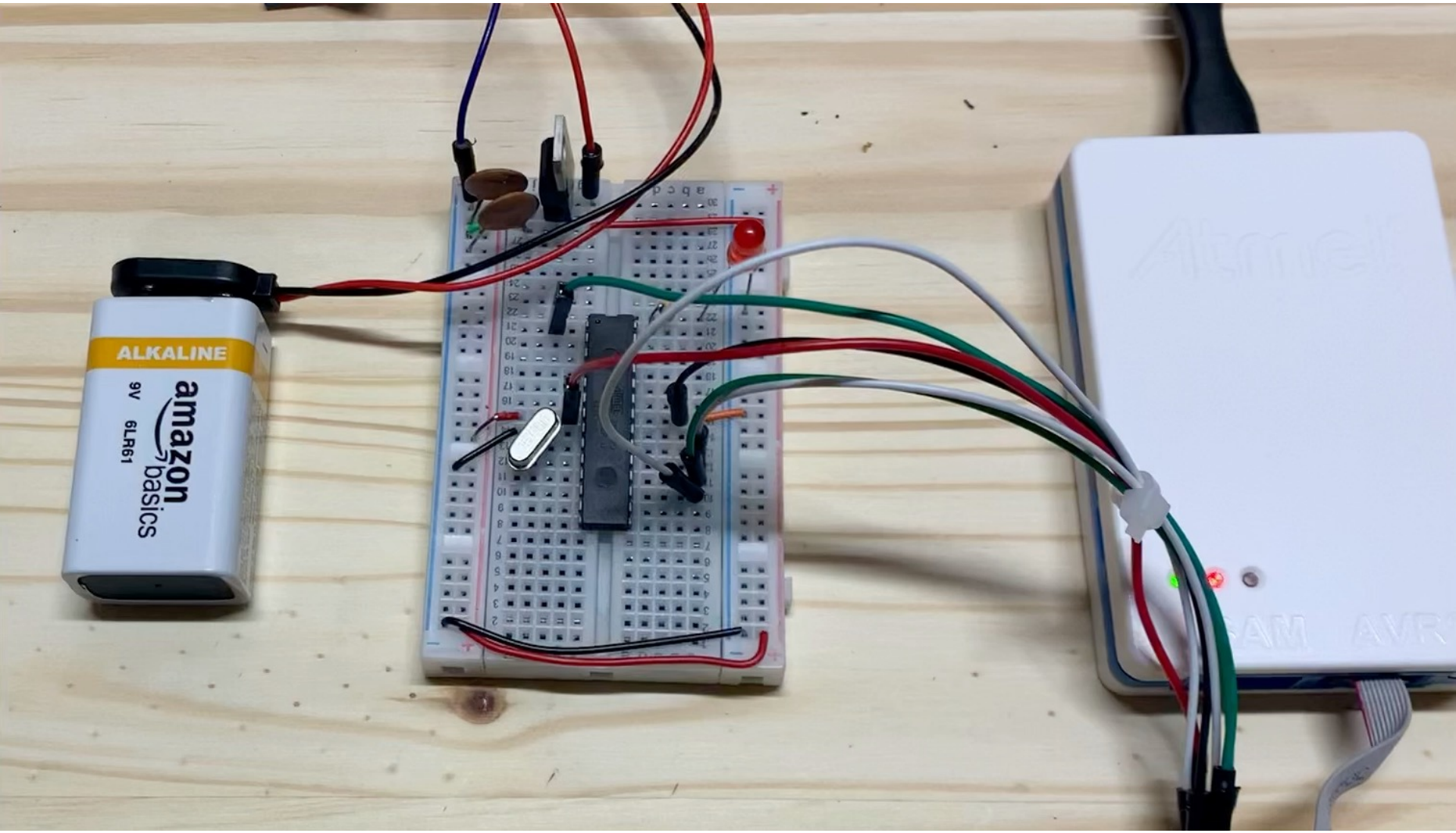
```
/*   blink_led.c
     blink led on PC5 with delay
*/
#include <avr/io.h>
#include <util/delay.h>

int main(void)
{
     DDRC = 0xFF;
     while(1)
     {
          PORTC = 0xFF;
          _delay_ms(100);
          PORTC = 0x00;
          _delay_ms(100);
     }
}
```

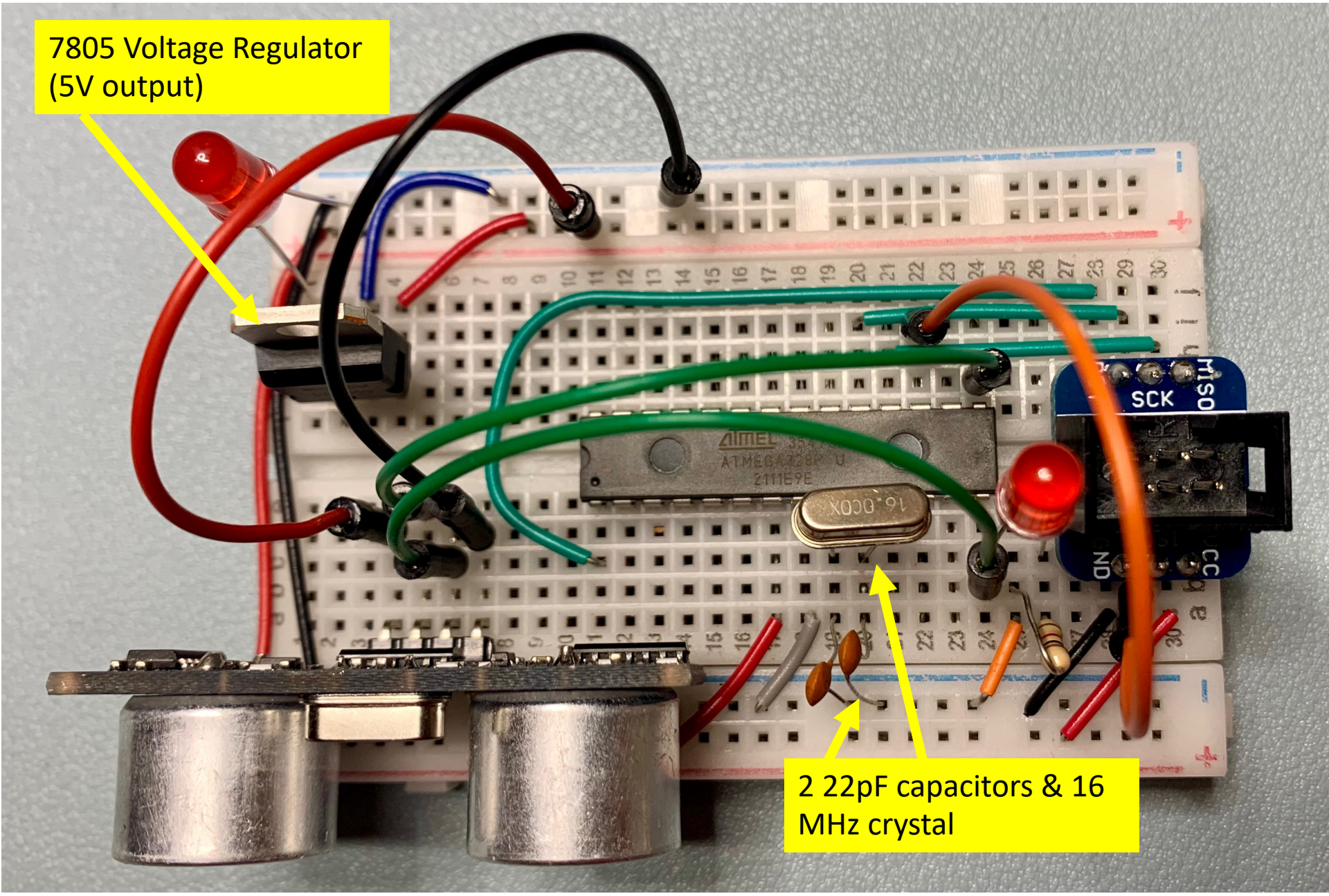
Develop & compile code

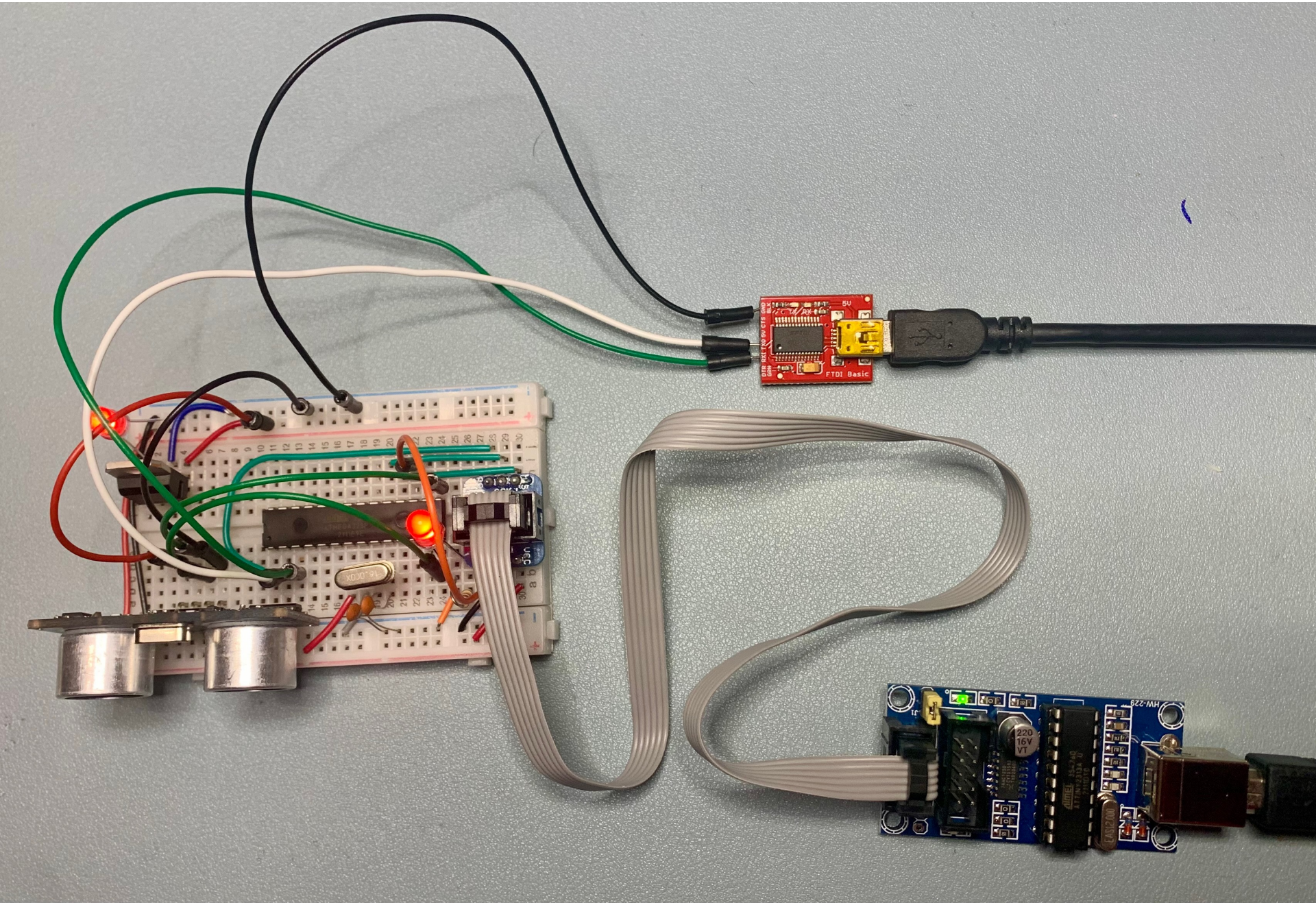
USB

Pocket Programmer




ALKALINE
amazon basics
9V
6LR61


7805 Voltage Regulator
(5V output)
2 22pF capacitors & 16
MHz crystal
SCK
MISO


FTDI Basic
5V
220
16V
VT

Code Development Options
Editor, Compiler, Downloader

Use a big Integrated Development Environment (IDE)

MPLABX – Windows, MacOS (with Intel processor)
Microchip Studio – Windows

Use a code editor & command-line toolchain

Visual Studio Code (editor)
avr-gcc toolchain
avrdude – downloader
Command Prompt (Windows) or Terminal (MacOS)

This is the code development environment we are using in ECE-231. This works on Windows, macOS (Intel) macOS (Apple silicon)

All lecture slides & links to all zoom videos for this semester's ECE-231 course are posted to the ECE-304 moodle site if you want a refresher in embedded C.