08: Embedded programming

Project

Brainstorm/propose projects on Ed thread

Next week, we'll open a form to rank your top choices and match you with a team

Project must:

- Use PWM, ADC, or DAC
- Have at least one interrupt service routine
- Have a watchdog timer (doesn't count as your ISR)
- Use at least one of: Serial communication, Wifi, Timer/counter

Most project ideas can be refined to meet these requirements; focus on the idea for now

Final writeup will be required to have process and modeling & verification documentation

Why we're thinking about the project so early

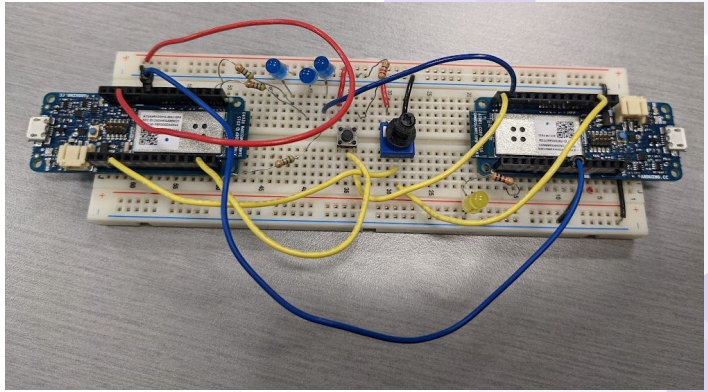
Time to find you resources

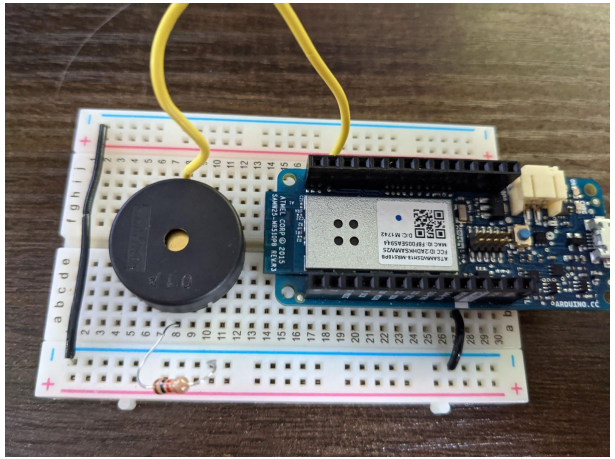
Time to refine the design

Time to order supplies

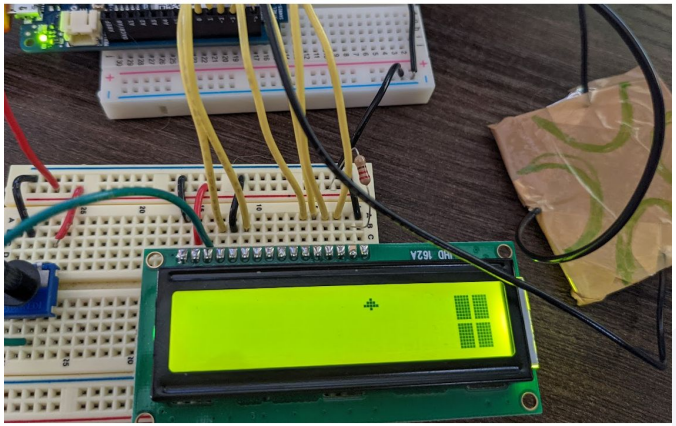
- Each team will have a small ($40) budget

Skills in upcoming labs





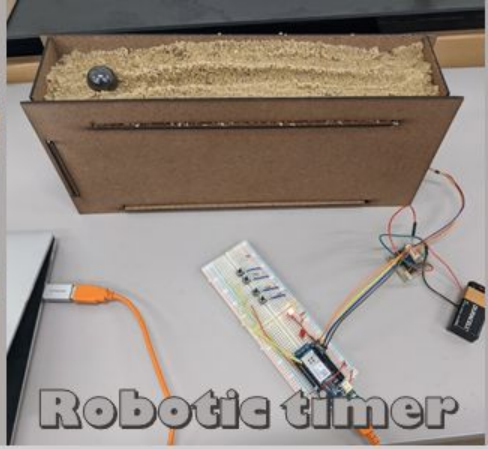
don't call yourself a "gamer"



if you haven't played this game!

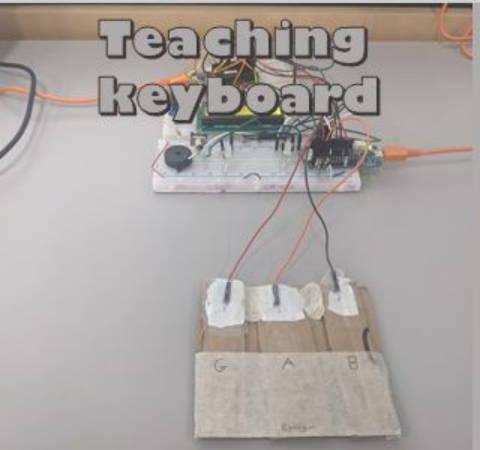
Project ideas

- Games
  - Electronic whack-a-mole
- Art
  - Music, visual art, etc
- Controllers
  - Keyboard, game controller, etc prototype
- Other
  - Plant moisture/light monitor



Incredible work on your projects, CS 1600! We want to brag about you!

- Arun, Jason, Stephen, and Prof. Zizyte



Why do we need to know about the type and layout of memory on an MCU?

Knowing about limitations (motivates optimizations)

- gives you a sense of feasibility
- if you need to add hardware
- limited number of writes to a location

Which memory is volatile

Understanding of memory layout helps you diagnose bugs

Keywords for sharing data

static

Value of local variable will persist between function calls (is in **memory** rather than the stack)

Useful in a function like loop() when you don't want to declare a global variable

Still local to the function

volatile

Means variable can change outside of main execution (e.g. by an ISR)

Always use volatile when working with variables that change in ISRs!

Tells compiler not to make certain optimizations (never keep value in a register)

Besides speed and memory use, what are some other metrics we may target when optimizing embedded code?

Embedded programming

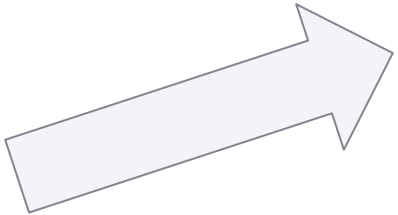
Reasons embedded programming differs from general-purpose computing:

- Cannot assume portability
- Parallelism from interrupts
- Limited by hardware
  - memory, power, cpu speed, I/O latency
- Care more about scheduling/deadlines
- Safety-critical applications

Example tradeoffs - inline functions

Compiler copies the contents of the function any time a call to the function appears in code

```
inline int add(int a, int b) {
    return a + b;
}
...
void main() {
    ...
    var3 = add(var1, var2)
}
```



```
void main() {
    ...
    var3 = var1 + var2
}
```

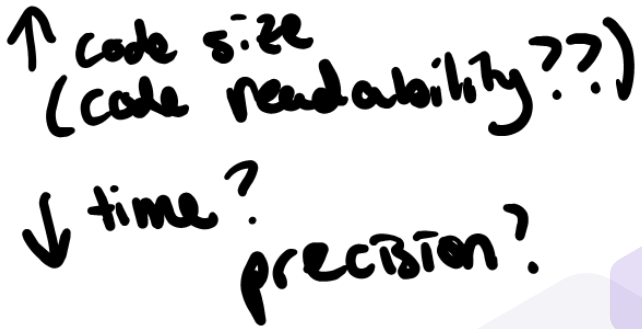
↑ code size

↓ subroutine overhead (time)

Example tradeoffs – lookup tables

A switch statement or an array in memory gives the answer for every possible input, instead of doing a computation

```
switch(x) {
    case 3:
        return 2;
        break;
    case 10:
        return 3;
        break;
    case 1:
        return 1;
        break;
```



Example tradeoffs – global variables

Declare a global variable that sits in memory instead of passing it around in function calls

- decreased runtime
- increased spaghetti code

Why is recursion dangerous on an MCU?

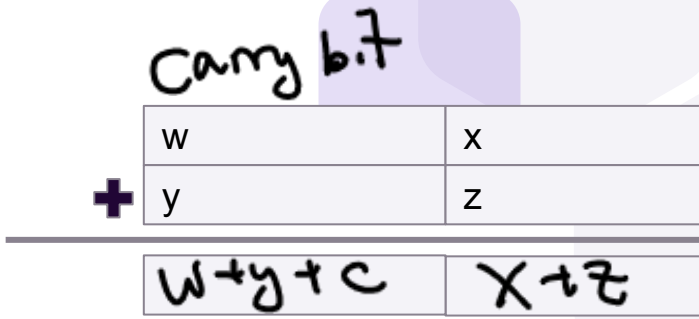
Coding practices: portability

Word size

> **`int`** will mean different things on an 8-bit CPU vs a 32-bit CPU

Tip: be specific about size

> **`int8`**, **`uint16`**, etc



What if you need to emulate a 16-bit int on a 8-bit CPU?

> Fake it with multi-precision math!

Floating point is often avoided in MCU applications. Why?

Why

it might not exist for 8-bit or 16-bit sizes?

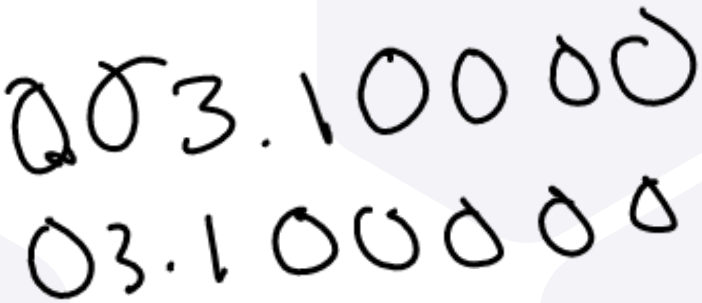
it might take more instructions to do addition/multiplication

- takes more CPU cycles

may require more complicated hardware (FPU, or floating point unit)

- some MCUs straight up don't have one

quantization optimizations for sensor inputs that don't require floating point



QUESTION 1

Logistics/warmup **0.30000000000000004** / 0.3 pts

1.1	Multiple choice	**0.1** / 0.1 pts
1.2	Fill-in-the-blank	**0.1** / 0.1 pts
1.3	Select-all	**0.1** / 0.1 pts

Fixed point

Represent fractional values with implicit *fixed* divisor

- Decimal example: if fixed divisor were 1000, we would represent 0.04 as “40” (e.g. counting by milliseconds instead of seconds)
- In binary, we use powers of two as divisors
- Write format as “x.y”, with x digits before ~~decimal point~~ mantissa and y after
  - All values use this format (position of mantissa doesn’t change between variables)

Fixed point example

Interpret the bits “01010110” in different formats:

64+16+4+2

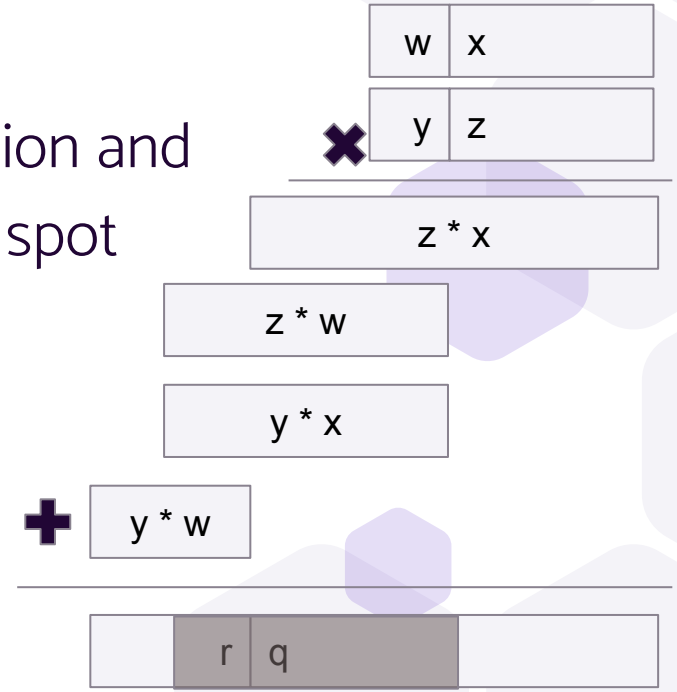
format	regular/int	1.7	4.4	5.3
divisor	n/a	2^7 = 128	$2^4 = 16$	$2^3 = 8$
Interpreted value	86	86/128	86/16	86/8

Fixed point math

Addition/subtraction work as usual

Let the CPU perform the computation and interpret the mantissa at the same spot

Multiplication: need to truncate



Summary

- Your code gets turned into assembly gets turned into machine code
- Machine code is executed on the CPU
- Data for programs is stored in different areas of memory
- Because of these architectures, embedded programming has some unique considerations