Keep posting project pitches!

# 10: Embedded Programming and Watchdog timers

*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
- Context switching from interrupts
- Limited by hardware
  - memory, power, cpu speed, I/O latency
- Care more about scheduling/deadlines
- Safety-critical applications

# 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

# 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);
    var4 = add(var2, var3);
}
```

```
void main() {
    …
    var3 = var1 + var2;
    var4 = var2 + var3;
}
```

*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

| | | |
|---|---|---|
| | w | x |
| + | y | z |
| | | |

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

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
- Human-readable format: "x.y"
- Machine format: fixed divisor not stored data; interpreted in code

# Fixed point example

Interpret the bits “01010110” in different formats:

| **format** | **regular/int** | **1.7** | **5.3** |
|---|---|---|---|
| **divisor** | n/a | 2^7 = 128 | |
| **Interpreted value** | 86 | | |

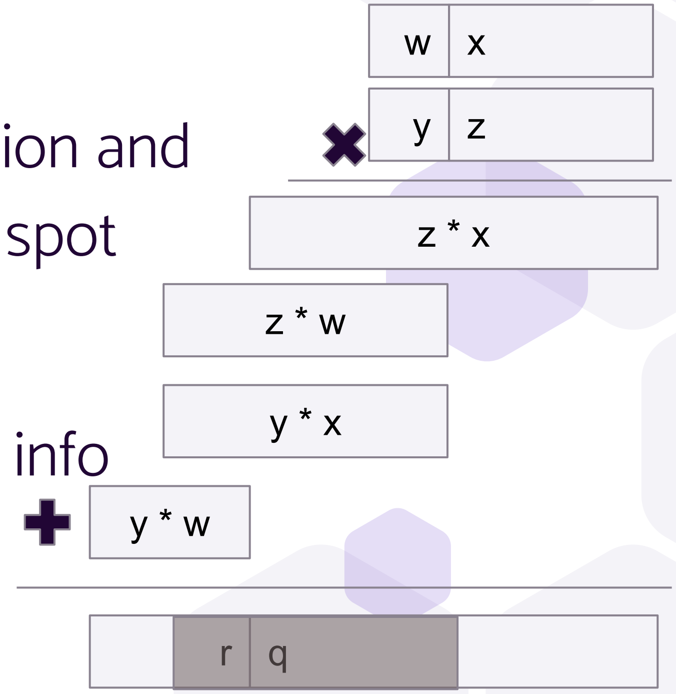
# Fixed point math

Addition/subtraction work as usual

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

Multiplication: need to truncate

See inset in 8.2 of Lee/Seshia for more info



*What are some reasons (software bugs or external causes) that embedded software might hang?*

Petting the watchdog

# Watchdog timers

Special timer peripheral that counts down to 0 on a clock that can't be powered off

Can be reset by writing a value to a special register ("petting" the watchdog)

If reaches 0, resets (or shuts down) entire system

- Idea is to detect system hang

# Rules for watchdog timers

When to pet - before it reaches 0

- Have an estimate for how long your execution takes
- Make sure it can catch any task failure

How to pet - complex enough so that it's not an accident

# Anti-patterns for watchdogs

Using a watchdog for control/functionality

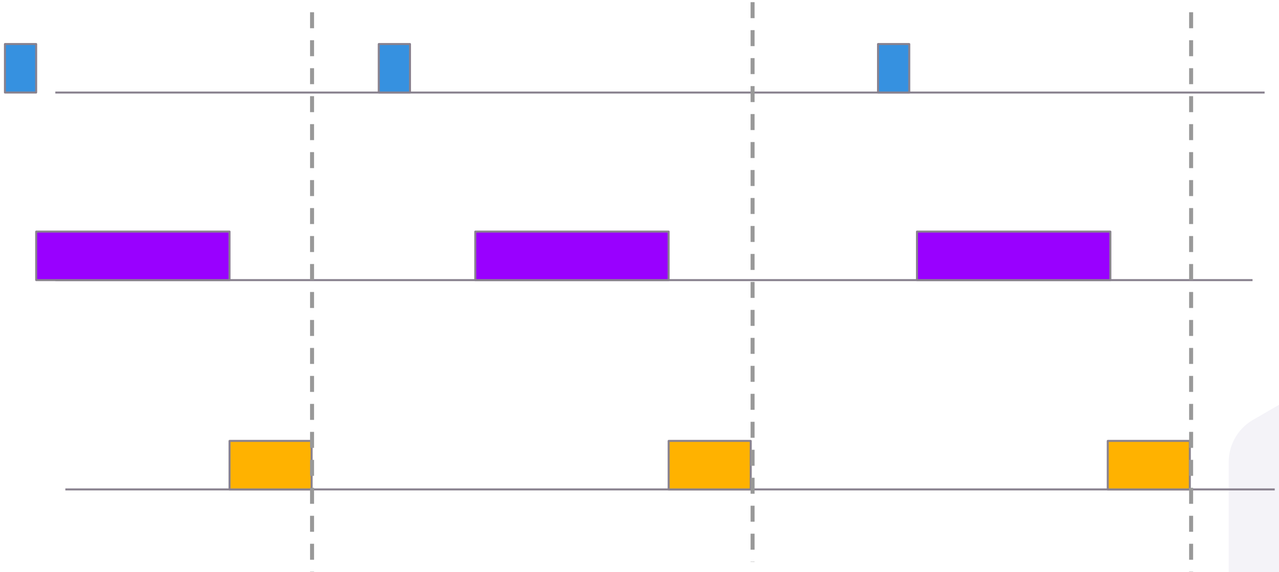
Petting in too many places

Using a timer to pet the watchdog

Turning the watchdog off in software

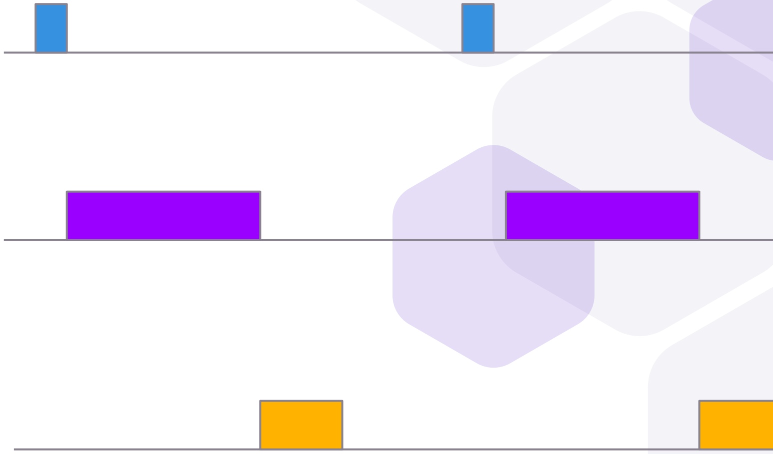
# A preview: periodic tasks

*n* tasks each with a given period and worst case execution time (for now assume same period)



# What's the problem with this?

```
blueTask {
    … do stuff; …
    pet_watchdog; }
purpleTask {
    … do stuff; …
    pet_watchdog; }
goldTask {
    … do stuff; …
    pet_watchdog; }
```



# Blocking vs. non-blocking functions

**Simplest task scheduler:**

```
void loop() {
  blueTask();
  purpleTask();
  goldTask();
}
```

Blocking function:

```
void goldTask() {
  res = 0;
  while (! res) {
    res = serverTask();
  }
  … // compute on res
}
```

Non-blocking function:

```
void goldTask() {
  res = serverTask();
  if (res) {
    // compute on res
  }
}
```

# Blocking vs. non-blocking functions

**Simplest task scheduler:**

```
void loop() {
  blueTask();
  purpleTask();
  goldTask();
}
```

Blocking function:

```
void goldTask() {
  int res = 0;
  while (! res) {
    res = serverSend();
  }
  … // compute on res
  petWatchdog();
}
```

Non-blocking function:

```
void goldTask() {
  int res = serverSend();
  if (res) {
    … // compute on res
    petWatchdog();
  }
}
```

*How would you pet the watchdog for a multitasked system?*

# Challenge mode