ECE 471 – Embedded Systems
Lecture 20

Vince Weaver

https://web.eece.maine.edu/~vweaver

vincent.weaver@maine.edu

26 October 2026

Announcements

- Don't forget HW#7. Pick up parts if you haven't already.
- Still working on grading HW#5 and midterm exam

Floating Point in Embedded Systems

- You might want to use floating point in HW#7
- Let's go over some of the issues with floating point in embedded systems

C Integer Size Notes

- How big is an `int` in C?
  16-bit? 32-bit? 64-bit?
  Varies by architecture
- Rule is char ≤ short ≤ int ≤ long
- For a surprisingly long time int was 16-bits on low-end and embedded systems

C Variable Sizes

- 32-bit Linux is ILP32
  `int`, `long`, and pointers are all 32-bit
  `long long` if you want 64-bit
- 64-bit Linux is LP64
  `int` is 32-bit, `long` and pointers are 64-bit
- Windows is LLP64
  `int` and `long` 32-bit
- What's the best way to force size?
  Use stdint include, things like `uint64_t`

C Floating Point Notes

- You might not have used floating point in C much. Briefly covered in ECE271
- 32-bit signed integer only lets you have values from $-2,147,483,648$ to $2,147,483,647$
- What if you want bigger? Smaller? Fractional / decimals?

Floating Point

- IEEE-754 standard
- For a 32 bit value, have one bit for sign, 8 bits for exponent, 23 bits for mantissa. How this actually works a bit beyond this class
- $(-1)^s \times 2^{exp-127} \times 1.mantissa$
- Can have a much wider range of values, also things like infinity, NaN, etc.
- Can do this all in hardware. Many low-end embedded systems these days will have support (all the Pis do)

What do you do on systems w/o Floating Point

- Many embedded boards do not have it
- The boards in ECE271 do because Hummels uses them for DSP class
- All Raspberry Pis do (it's why they are compiled ARMHF – HF = hardware floating point)
- You can have either the operating system or the compiler "emulate" floating point in software. It's slow
- Alternatively you can use fixed point math instead

Fixed Point

- With integer math you can have "fixed point" where you arbitrarily put the decimal point at a fixed place in the value.
- 16.16 means 16 bits integer part, 16 bits fractional
- add/subtract work as normal, have to adjust after multiply/divide
- You might use this on an embedded system w/o hardware floating point

Arbitrary Precision Math

- You can have libraries that dynamically allocate memory for arbitrarily large numbers and fractions
- This is overkill for normal math you might want to do
- GMP library is one such, GNU Multi-Precision Math Library, size of numbers only limited by RAM

Floating Point Sizes in C

- `float` is 32-bit
- `double` is 64-bit
- Can you have 16-bit, 8-bit, 4-bit, 2-bit floating point? A bit beyond this class, but yes, and they are popular on GPUs, especially when doing AI/machine learning

Math Library in C

- Can do things like add/sub/mul/div
- Anything more complex need the math library (sqrt, sin, cos, etc.)
- Back in the day for size reasons you wouldn't link against it unless specifically doing fancy math
- You will need to include math.h
- You will also when linking need to link in the math library, `-lm`

Floating Point to Integer Conversion

```
int value=45;
double temp;

temp=value;              // works, but truncates
temp=(float)value;       // casts make the conversion explicit
                         // but can potentially hide bugs
```

- C will auto-cast things so you can do things like this, but be careful as sometimes the rules are a bit obscure. Generally when converting float to integer it will truncate rather than round
- Constants 9/5 vs 9.0/5.0
  The first is an integer so just "1". Second is expected 1.8.

Explicit float/double Conversion

- Rather than cast
- floor()
- ceil()
- round()
- rint()
- nearbyint()

Floating Point Printing

- Printing. First prints a double. Second prints a double with only 2 digits after decimal.

```
printf("%f\n",temp);       // print 32-bit float
printf("%lf\n",temp);      // print 64-bit double (long float)
printf("%.2lf\n",temp);    // print double with 2 digits after .
```

Floating Point Pitfalls – Truncation

- Sort of like in decimal where only fractions of $1/2$ and $1/5$ have exact values (because base 10) on binary computers only multiples of $1/2$ are exact
- This means unexpected things happen, like if you add $1/3$ and $2/3$ you might not get 3.0 but something close like 2.999999999999999943
- That will round normally in `printf()` but if you do an implicit cast to an integer you might confusingly get 2 instead of 3

Firmware

- Software – code
- Hardware – the physical device (can throw it out the window)
- Firmware – low level code closely tied to hardware, often difficult (or impossible) to change

Firmware Location

- In old days in ROM (read-only memory)
- ROMs got more programmable over the years, EPROM, EEPROM, flash
- These days probably in flash of some sort
- Usually is harder to update than software

System Booting

One prominent place you'll find firmware is during the boot process

Boot Firmware

Provides booting, configuration/setup, sometimes provides rudimentary hardware access routines.
Kernel developers like to complain about firmware authors. Often mysterious bugs, only tested under Windows, etc.

- BIOS – legacy 16-bit interface on x86 machines
- UEFI – Unified Extensible Firmware Interface
  ia64, x86, ARM. From Intel. Replaces BIOS
- OpenFirmware – old macs, SPARC
- LinuxBIOS

Boot Firmware Programming

- Programming it can be complicated
- Often assembly language, possibly before RAM is enabled
- Tell story of my job at Frontpath where needed to enable spread-spectrum via i2c on a web-pad early in the boot process with no RAM configured yet