ECE 471 – Embedded Systems
Lecture 7

Vince Weaver

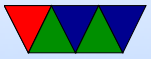
https://web.eece.maine.edu/~vweaver

vincent.weaver@maine.edu

18 September 2024

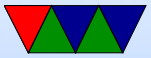
Announcements

- HW#1 grades out soon
- Don't forget HW#2 is due Friday
- Embedded in news: what do you do if your boss asks you to install a self-destruct device in your embedded system? Is it ethical?

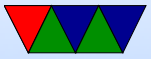


Last Time

- Whirlwind tour of Command Prompt
- Here are a few extra things
- Can use the Command Prompt in a GUI, use Terminal emulator
- File/device permissions:
  - Operating system enforces permissions on files, can see with `ls -l`
  - Can belong to users, groups
  - Can set read/write/execute for user/group/everyone

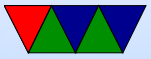


- use `chmod`, `chgrp`, `chown`
- If you have root/super-user/sysadmin privileges you can ignore permissions
- Use “sudo” but be careful, with great power comes great responsibility



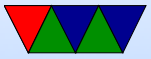
HW#1 – Characteristics of Embedded System

- embedded inside – sometimes hard to know. Is a raw pi one? Pi used as desktop? Pi used as retro-pi? Pi controlling a 3D printer?
  Lack of being able to update not necessarily the same
- resource constrained
- dedicated purpose
- lots of I/O
- real-time constraints

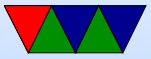


HW#1 – Other characteristics

- Low-cost?
  This is complicated. Something like a desktop might be optimized for cost extremely, while a one-off embedded system might not, and in fact might be over-engineered (like a space probe) because has to operate in tough conditions.
- Low-power?
  again, this can be part of resource constrained but be sure to explain

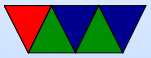


- Operating system?
  Can have an OS and still be considered embedded.
- Real-Time Confusion: we will discuss this more in future, for example Just turning off the motor, and it takes an extra 1/2s is not really considered a real time thing. No one dies, no hardware destroyed, just mild annoyance if noticed at all. Now if somehow it had to keep the waveform to H-bridge exact within 1ms or the motor would overheat and catch on fire, that could be a real-time issue.



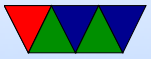
HW#1 – Identifying an Embedded System

- Be decisive with your answer, and be specific with your reasoning
- iPhone
  real time doesn't necessarily mean quick-response, or FLOPS
  updatable not a characteristic
- Toothbrush is actual specs I came across Note low-price is not a characteristic, often opposite might be true
- Microwave: having a clock doesn't make it real time.



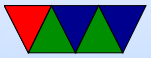
HW#1 – Bits

- ARM1176 is generally considered 32-bits
- ARMv8 is generally considered 64-bits
- 6502 generally considered 8 bits
- There are people who will have long drawn-out internet arguments about the bitness of old systems



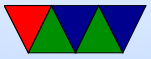
HW#1 – ASIC vs ucontroller

- cost/power. Depends a lot on numbers made, process, and how well designed it is.
- Could be lower-cost/faster speed, but not necessarily. Why bother then? Cost?
- Extra hardware overhead? ASIC mostly just flip flops and gates. SoC internally a lot more, but these days not much else is needed.
- More secure? Can you reverse engineer an ASIC?



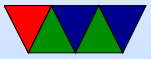
C Review

In past years sometimes the reason a HW assignment didn't work was due to using C poorly rather than misunderstandings of the desired algorithm.



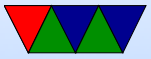
Loops in C

- `int i;` (why int? is int signed? inside of loop def?)
- `for(i=0;i<10;i++) {...}`
  0, 1, 2, ... 9
- `i=0; while(i<10) { ...; i++; }`
- `i=0; do { ...; i++; } while(i<10);`
  Always runs at least once



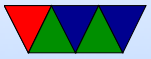
`printf()` in C

- Lots of options, see man page
- How print an integer? `printf("%d",i);`
- Character? String? floating point?
  `printf("%c %s %f %x",c,s,f,x);`
- More advanced formatting stuff
  `printf("%0.3f",f);`
- Escape characters like percent, newlines and quotes
  `printf("\t \n \" \%");`



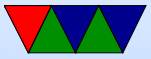
Common C Pitfalls – Static Memory

- Allocating things like arrays (`int a[5]`)
- C doesn't prevent you from accessing past the end
- What happens if you do go outside the boundary?
  - Crash? Memory corruption?
  - Nothing? (you are lucky and it hits something unimportant. Is that best or worst case?)



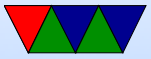
Common C Pitfalls – Dynamic Memory

- Often avoided on embedded systems
- Dynamically allocate memory with `malloc()` and `calloc()`
- `char *ptr=malloc(128);`
- Should check returned value against NULL.
  What happens if you de-reference a NULL pointer?
- Out of bounds memory access same issue as with static `ptr[200]=0;`



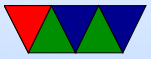
Common C Pitfalls – Freeing Memory

- Memory allocated with malloc/calloc needs to be freed with `free()`
- What happens if you forget to free memory?
  Memory Leak
- Might not be an issue if you allocate something once and use it all program. More of a problem if you're constantly allocating/freeing and miss freeing.
- What happens if you free the same memory twice?
  Crash and/or security issue



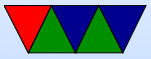
More on Memory Leaks

- Note not all memory leaks are critical
- If you allocate it once and use if for the whole program what happens?
- If you have an operating system then generally it will free all allocated memory when the program exits
- It can still be considered polite to free() things anyway



Debugging Memory Access issues

- The **Valgrind** utility can help debug these errors
  Mostly dynamic, not much can be done about static
- It translates your program on the fly, instruments all memory allocations, and monitors all loads/stores to see if they are in bounds
- Valgrind can also help find memory leaks
- Downside: really slow

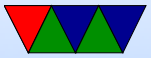


C Pitfalls – Strings

- C strings are just zero-terminated character arrays

```
char s[]="Hello";
H e l l o \0
```

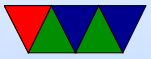
- You can end up with all the same problems with memory accesses, especially running off the end
- Sometimes this is called NUL terminated, note that NUL means 0 here and is unrelated to NULL pointers



C Pitfalls – C String Library

- There are versions of the string routines that take a length (`strncpy()` or `strlcpy()` instead of `strcpy()` but beware those have their own issues

```
void strcpy(char *dest, *src) {
    while(*src!=0) {
        *dest=*src;
        *dest++; *src++;
    }
}
```

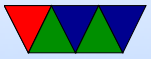


C Pitfalls – Braces

- Missing braces

```
if  (a==0)
        b=2;

if  (a==0)
        b=2;
        c=3;
```

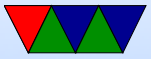


C Pitfalls – equality check

- = vs ==

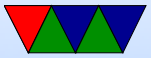
```
if (a=0) do_something_important();
```

- Never ignore warnings from the compiler!
- Some people will use `if (0=a)` to force an error



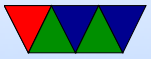
C Pitfalls – Type Issues / Casting

- `int x; float f; x=f;`
- C will happily auto-convert types for you
- Also be careful of signed/unsigned issues
- You might get warnings that go away if you cast things like `x=(int)f;`
  Be sure you know what you are doing here as it can go terribly wrong.



C Pitfalls – Setting Constants

- Floating point constants can be tricky, setting `double x=9/5;` will get you 1, you want 9.5/5.0
- Leading zeros specify Octal (base-8) numbers so something like `int x=010;` might give surprising results.



Debugging – when things go wrong

- Use a debugger like gdb
  - Compile your code with `-g` for debug symbols
  - Run `gdb ./hello`
  - `bt` backtrace, `info regis` gives register, `disassem` disassembles, etc.
- Sprinkle printf calls