ECE 471 – Embedded Systems
Lecture 30

Vince Weaver

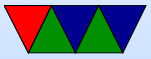
http://web.eece.maine.edu/~vweaver

vincent.weaver@maine.edu

16 November 2022

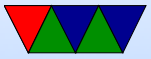
Announcements

- Midterm Friday. Can have one piece of 8.5"x11" paper for notes (single side)
- Project status report due 22nd (Tuesday before Thanksgiving)
  Important parts are re-stating the topic, how you're doing, are you OK on parts, and finally which day you'd like to go (early Friday, or M/W/F)
- NASA SLS rocket finally launched late last night.
  Computer delay, ethernet switch needed replaced?



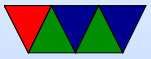
HW#7 Review – Code

- People managed merging 2 bytes into one OK
- A lot of trouble converting binary to hexadecimal.
  Trouble with embedded systems is off-by one values like this can be really hard to debug
  Note on gcc at least you can enter binary constants like `0b10100101`
- Divide by 1023 vs 1024
- What is the max frequency? Last year someone setting to 500kHz by accident, a few degrees different. Data



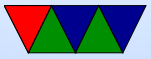
sheet unclear

- Be sure to check for error on open(), biggest source of errors. Linux won't crash, it will happily just report errors that your code is likely ignoring.
- Errors: exiting. Not print plausibly real invalid values. In our case, printing 0V when actually 3.3V not an issue, but imagine if it were 10,000V and you print 0V

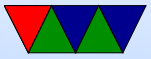


HW#7 Review – Questions

- Disadvantage of SPI?
  More wires, no standard, no errors
- Advantage of SPI?
  Lower Power, Full Duplex, No max speed
- TMP36 on end of cable.
  Voltage Drop, Noise?
  Datasheet has two options, convert to current, or an extra resistor.
- Minimum frequency of 10kHz or results invalid. Maybe

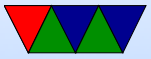


cannot go this fast if bitbanging via GPIO. Also context switch in middle, Linux not realtime?



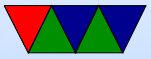
HW#7 Review – Linux "fun"

- /dev/null
- /dev/full
- /dev/zero, holes in files
- /dev/random – give explanation on sources of randomness (entropy), pseudo-randomness, etc.
- Mention related DOS/Windows compatibility issue with device filenames



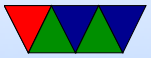
Homework 8 – Code

- Error checking. Exit if cannot open. If you don't, can segfault if try to fscanf a NULL FILE*
- Returning -1 on error might be bad idea
- What to report on error? What's an invalid temperature? Not just unlikely? (Below Absolute zero)
- If using streams (FILE *fff), on fopen() error it returns NULL, not -1.
- Be sure to close files, otherwise leak file descriptors Be careful if multiple exit points, must close at all (goto)



- Be careful with your 9/5 Fahrenheit conversion!
- Finding a file using C. `opendir()` `readdir()`, horrible interface

  Bit of a tangent on the downsides of the `readdir()` interface

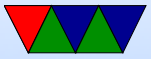


HW#8 – Questions

- Why need Vdd? To provide enough current for this particular chip needs extra current if you want parasite mode.

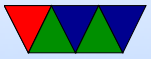
  You can try without Vdd but you will always read out 85C.

  Manual suggests MOSFET, but apparently it's possible on Pi if use 4.7k resistor as well as "strong-pullup=y" kernel command line option.
- Because of distance, 1-wire



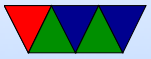
- shell script
  - `#!/bin/sh` should be first line (magic number)
  - Trouble if edit on windows, why (linefeed vs carriage return)

    shebang description
  - Making executable with chmod
  - Default shell, can put other things there, like python or perl, etc, even ARM emulator
  - sh vs bash

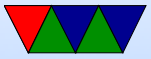


Ethics in Software Engineering

- There's ethics when programming
- What if company wants you to code Dark Patterns?
- Privacy? Data Logging? Tracking?
- Unintentional security leaks: fitness trackers giving away military locations
- Thermostats: forget to change password if move or divorce, others now control your heating
- Amazon/Google devices, always listening in your house
- Web-cameras everywhere

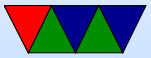


How Can You Avoid Bad/Buggy Code?



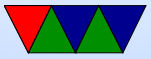
Code Safety Standards

- Avionics: DO-178C (1992 for B)
- Industrial: IEC 61508 (1998)
- Railway: CENELEC EN 50128 (2001)
- Nuclear: IEC 61513 (2001)
- Medical: IEC 62304 (2006)
- Automotive: ISO 26262 (2011)



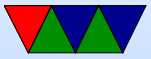
Code Safety Standards

- Is it easy to get a hold of copies of these?



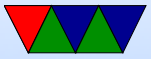
Aviation

- DO-178B / DO-178C
- Software Considerations in Airborne Systems and Equipment Certification
  - Catastrophic: fatalities, loss of plane
  - Hazardous: negative safety, serious/fatal injuries
  - Major: reduce safety, inconvenience or minor injuries
  - Minor: slightly reduce safety, mild inconvenience
  - No Effect: no safety or workload impact



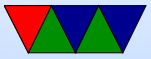
Automotive ISO 26262

- What is a document like this like?
- Vocab and definitions
- Management
- Safety Life Cycle
- Supporting processes
- Safety Analysis
- Risk Strategy
- Severity
  - S0 – No injuries

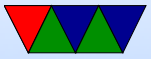


- S1 – No injuries
- S2 – Severe injuries
- S3 – Not survive-able

- Exposure
  - E0 – Unlikely to Happen
  - ...
  - E4 – High probability
- Controllability
  - C0 – Controllable
  - ...
  - C3 – Uncontrollable

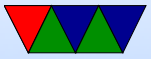


- Look up those in a matrix so you know how to assess risk, know how important to fix, know what resources to apply



Medical Response

- IEC 62304 – medical device software – software lifecycle
  - Quality management system – establish the requirements needed for such a device, then design methods to be sure it meets these
  - Avoid reusing software of unknown pedigree (don't just cut and paste from stackoverflow)
  - Risk management – determining what all the risks involved are, then determine ways to avoid or minimize them



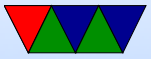
- Software safety classification

  Class A: no injury possible

  Class B: Nonserious injury possible

  Class C: serious injury or death possible

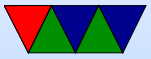
  Software sorted into these areas. Class A do not require the same precautions as the others.



Other notes

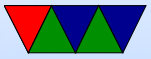
- Top down vs Bottom up Design
  Spec out whole thing and how they work first
  Start with core part and just keep adding to it until it works

- Requirements/Specifications?

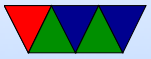


Writing Good (Embedded) C Code

- Various books
- Comment your code!
- Strict, common code formatting (indentation)
- More exact variable types (int32_t not int) Size can vary on machine, and on operating system
- Subset to avoid undefined behavior

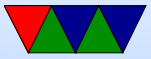


- Tool that enforces the coding standards
- Good to write safe code even if it isn't meant for a safe application. Why? Good practice. Also who knows who or when your code might be copied into another project.



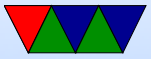
MISRA

- **MISRA: Guidelines for the Use of the C Language in Critical Systems**
- Motor Industry Software Reliability Association
- Guidelines: Mandatory, Required, Advisory
- Some sample guidelines
  - Avoid compiler differences int (16 or 32 bit?) `int32_t`
  - Avoid using functions that can fail (malloc()) allocate memory at beginning of program not throughout
  - Maintainable code, comments, coding style (see

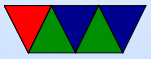


below)

- Compliance
  - All mandatory rules must be met
  - All required rules must have formal deviation
- Deviation
  - Must make a format explanation for why deviation is necessary
  - Prove you've thought about the issue
- MISRA 2012 has 143 rules, 16 directives
- NOTE: YOU CAN STILL WRITE BAD CODE EVEN WHEN FOLLOWING THIS

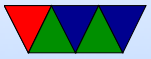


It just makes it easier to write good maintainable code.



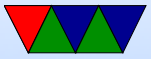
C Style

- What can C look like?
  IOCCC (International Obfuscated C Code Competition)
- Variable style, `CamelCase`, `under_score`, Hungarian Notation (arru8NumberList)
- Indentation (tabs vs spaces)
- Curly braces on same or next line
- Comment style
- Auto-generated documentation from comments



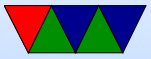
Good Test Practices

- Unit testing
- Test Driven Development – tests written before the code happens, needs to pass the tests before done
- Fuzzing
- Device Hardening?



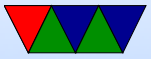
Good Documentation Practices

- Comment your code
- Write documentation! Make sure it matches code! There are some tools that can auto-generate documentation from special code comments
- Use source control (git, subversion, mercurial)
- Use good commit messages in your source control



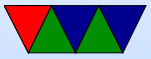
Space Shuttle Design

- https://www.nasa.gov/mission_pages/shuttle/flyo flyfeature_shuttlecomputers.html
- Issues normal embedded systems don't have: Vibration at liftoff, Radiation in Space
- If computer stopped for more than 120ms, shuttle could crash
- "Modern" update in 1991: 1MB Ram, 1.4MIPS. Earlier was 416k and 1/3 as fast and twice as big
- Change to code, 9 months testing in simulator, 6 months



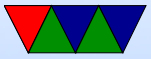
more extensive testing

- 24 years w/o in-orbit SW problem needing patches
- 12 year stretch only 3 SW bugs found
- 400k lines of code
- HAL/S high-order assembly language (high-level language similar to PL/I)
- PASS software – runs tasks. Too big to fit in memory at once
- BFS – backup flight software. Bare minimum to takeoff, stay in orbit, safely land, fits in memory, monitors pASS during takeoff/landing Written by completely different



team.

- 28 months to develop new version
- IBM
- Extensive verification. One internal pass, one external
- 4 computers running PASS, one running BFS
- Single failure mission can continue; still land with two failures
- 4 computers in lock-step, vote, defective one kicked out



SpaceX Falcon 9

- Linux – on dual core x86 systems
- Three each, vote
- Flight software in C/C++
- Dragon displays in Chromium+JS