1

# Introduction to Firmware Design

*Or* "Firmware: harder than software."

#### **Pauline Pounds**

16 April 2019 University of Queensland

#### But first...

#### A valuable message about safety...

**Campus Calamities presents** 

## Biohazard

#007

#### But secondly...

#### Some house keeping

#### Calendar at a glance

Week	Dates	Lecture	Reviews	Demos	Assessment submissions
1	25/2 - 1/3	Introduction			
2	4/3 - 8/3	Principles of Mechatronic Systems design			Problem analysis
3	11/3 – 15/3	Previous years deconstruction case studies			
4	18/3 – 22/3	Professional Engineering Topics	Progress review 1		
5	25/3 – 29/3	PCB design tips			
6	1/4 - 5/4	Your soldering is (probably)			
7	8/4 - 12/4	Intoittary-industrial design complex	Progress seminar	25% demo	
8	15/4 - 19/4	Introduction to firmware design			
Break	22/4 - 26/4				
9	29/4 - 3/5	Q and A sessions		50% demo	
10	6/5 - 10/5	No lecture	Progress review		
11	13/5 - 17/5	Q and A sessions		75% demo	Preliminary report
12	20/5 - 24/5	Monday lecture!!			
13	27/5 - 31/5	Closing lecture		Final testing	Final report and reflection

You are here 🤄

#### Progress seminars

- All done! Nobody failed! Yay!
- Please make sure you fill out your PAFs and return them before you leave
- If you haven't yet returned your PAF sheet from Progress Review 1, then do so immediately – *I AM COMING FOR YOU*

#### PAF 1 results

- I also now have complete PAF 1 results
  - If you are getting less than "C" from your peers, that's a fair sign that they are unhappy with you…
  - If your PAF 2 is also <C, I'll probably want to talk to you to see if there is a problem
- No PAF 1 result on Blackboard?
   Let me know!

#### Seminar and PAF 2 results

- Normally I have a histogram here, but I'm still waiting on Iain Rudge to send marks
   C'mon, Iain! つ <sup>o</sup>
  - I'll put them in the online version when finished
- Instead, some general comments:
  - OMG, keep to time! 10 mins means 10 mins
  - Analysis, analysis, analysis.
  - I'm not kidding about this seriously!

## Marks should all be updated

• People have been asking for PAF results and other marks

– Happy to oblige!

- With only a few exceptions, all your marks to-date should be online for your viewing pleasure
  - If not, you and/or someone probably owe me an assessment or PAF

#### Seminar Results

Results available via Blackboard right now!



2019

2015

## Managing your stress levels

• Stress levels have been low all throughout this year – concerning!



## FAQ Roundup

- Can you look at my part design and give me feedback?
  - Hell yeah! That's what I'm here for! Bring it!
- How much will this part cost?
  - Uhh... ask Jason Herriot?
- Does "wall plug power pack" include power packs with cables to the wall, like ATX supplies, laptop power bricks and such?
  - No. "Wall plug power pack" means power supplies where the transformer is integrated into the module that plugs into the wall. This is a safety compliance thing. Previously students have brought downright dangerous power supplies into the lab and it was a big problem.

#### STC sessions

• More STC sessions have been added for this week and during the break!

– Hooray!

- You need to book them through URite as usual
  - Hooray?

#### Foam extraction

- It has been brought to my attention that foam dust is Probably Not Great<sup>TM</sup>
  - You should avoid inhaling it or rubbing it into your eyes and mucus membranes.
- Not yet sure how much of a problem it will be, but I'm going to do the following:
  - Provide you with goggles
  - Provide you with a mini dust sucker to suck dust
  - Ask you to please keep your mucus membranes to yourself

#### Onwards!

#### To firmware and beyond!

## Firmware: OMG what is it?

- The embedded machine code inside a microcontroller
  - Like software, only closer to hardware... so
     "firm"-ware
- Unique from software, in that it can write directly to device physical outputs
  – Straight from 'The Matrix' to the "Real World"

Let's talk about microcontrollers...

## Key elements

- Microcontrollers behave like a whole computer on a chip:
  - 1. Co-locate RAM with working memory
  - 2. Integrated data bus
  - 3. Internal clock sources and timing
  - 4. Input/output hardware (PWM, SPI, CCP, etc)
  - 5. Specialised hardware (graphics/audio/RF/etc)

#### The legendary PIC 16F84A



19

#### Block diagram that thing...



#### So, what about it?

- Firmware is necessarily low level
  - Yes, IDEs help, but if it shields you from the details, it also shields you from TRUE POWER
  - In order of least to most abstracted:
  - Opcodes > ASM > C > Python > I dunno, Matlab, maybe?

- "Teach me, Sensei teach me this power!"
  - "Grasshopper, the longest code starts with a single #define"

## Some principles

- Structure everything
  - Structure is a lifestyle
- Modularise
  - Yes, even into separate files
- Use C
  - It'll make it easier
- No, really: use C (or maybe Python).

– You'll thank me when you're older

• Don't do ANYTHING without structure

"Code without structure is chaos" – a really smart person, probably Dijkstra

• Comments help you understand code, but good code makes comments less important

Major moving parts

- Front matter
  - All the nuts-and-bolts stuff, plus globals
- Main(){ }
  - Your high-level structure should be visible here
  - Your program shouldn't live in a function (although this is debatable)
- Client functions
  - All the stuff commonly reused, outside of loops

- One some level, you write code like a report
  - Start with a "skeleton" outline of what the program must do
  - Flesh out the details incrementally
- Of course, you want to be testing each functional block of code as you go...

- Start broad and work your way down
  - 1. Main/functions first (high level)
  - 2. loops/switches (intermediate level)
  - 3. Conditionals/function calls (low level)
  - 4. Operations (details)

#### This is all the "block diagram stuff"



- Begin with pseudocode
  - Write it out in words, then add details

Eg.: Get the next value; If the value passes the test; Store it for use later;

#### Then:

```
read(var); //Get next value
if(var > foo) { //Test value
buf = var; //Store for reuse
}
```

- If you do it right, you get complete code with integrated comments for free!
- Also check out "Literate programming"
  - Natural language explanation of process flow intertwined with code
  - Easy to read, easy to write

#### Modular Me

• Code reuse is made of God and Win

– A lazy programmer is an efficient programmer

- Modular code is also easier to manage
  - Easy to "drop in" new functionality
  - Especially important for firmware, where porting from one processor to another is an alltoo-common chore

#### Modular Me

- Consider:
  - A specific header file for each particular microprocessor you use (almost mandatory!)
  - A separate source file and header file for each peripheral IC
  - A separate source file for each common major hardware function (eg. can.c, spi.c)

A source map isn't a bad idea after a while...

#### Modular Me

#### Example:

#ifndef PIC18F14K22\_H
#define PIC18F14K22\_H

- #define TMR0IF (INTCON.F2)
- #define TMR0IE (INTCON.F5)
- #define PEIE (INTCON.F6)
- #define GIEL (INTCON.F6)
- #define GIE (INTCON.F7)
- #define GIEH (INTCON.F7)

#define INTEDG0 (INTCON2.F6)
#define INTEDG1 (INTCON2.F5)
#define INTEDG2 (INTCON2.F4)

#### Use C

• C is the screwdriver of firmware languages

The number of things it's good for vastly outnumber the things it isn't good for

- Almost everything compiles C
  - Everything embedded DOES compile C
- If your firmware needs something C can't do, you're probably doing it wrong

#### Use C

- If you are not using C, it's because:
  - A. You're doing something very specialised that uses arcane language extensions that C simply cannot support
  - B. Your boss/manager is a moron who insists on using the latest Agile-Cloud-Based-Buzzword-On-Rails mumbo jumbo
  - C. You're just scared of it and need to step up to the challenge?

#### Kernels

• Kernels mediate between hardware processes and application software layers

- Most embedded tasks are on-going, continuous, repeating processes
  - A simple kernel makes program flow really straight-forward to manage
  - Ideally suitable for dynamic control

#### Kernels

• A really small kernel is a snap to write:

```
interrupt() {/**interrupt handling, asynchronous stuff**/}
main() {
  hardware setup();
  while(1) {
       if (timing flag) { //Regulate process loop rate
              service routines(); //Reset timers, flags
              io process(); //Read/write ports, flush buffers
              application process1();
              application process2(); //etc; can be dynamic
       }
       whenever routines(); //Quick stuff, eg. poll loops
  }
```

## Rules for beginners (and everybody)

- Keep it simple really simple!
  - Short routines, broken out into files by function
- Be one with your micro
  - Understand it and the circuit around it, totally
- Use a reliable toolchain IDE is life
- Start with the biggest micro
  - Once it works, *then* you can optimise/downscale
- Document *everything* 
  - Code comments are mandatory!

#### What about the Powah?

#### "But... I was promised TRUE POWER! This sucks..."

## Ok, with well-crafted firmware and hardware...

#### What about the Powah?

- You can get precision instruction timing that can be accurate down to *picoseconds* I designed a nano-second timer for GPS in C
- You can build a drone from \$10 of parts, in a circuit the size of a 10c coin.
- You can build a remote sensor that works for five years on a single coin cell.

#### Join ussss

• So yeah... firmware is important

- Embrace structure
- Embrace C
- Embrace TRUE POWER





## Tune-in next time for...

#### Questions and Answers Vol. 1

or

"The quick and the decaf"

Fun fact: Boeing writes avionics code in C.