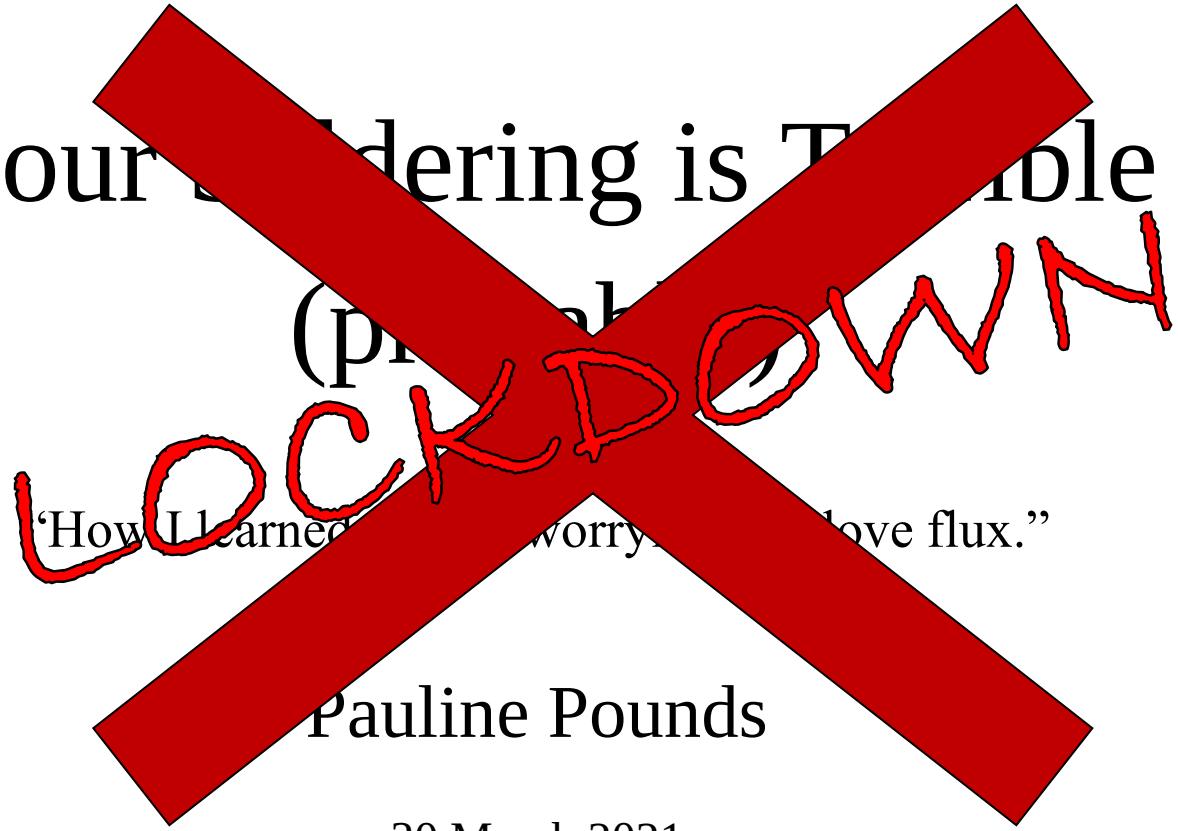

ASTRO

Your [illegible]dering is T[illegible]ble

(pr[illegible]h[illegible])



"How I learned [illegible] worry[illegible] love flux."

Pauline Pounds

30 March 2021

University of Queensland

Introduction to Firmware Design

Lockdown Edition

or

"Firmware: harder than software."

Pauline Pounds

30 March 2021

University of Queensland

But first…

A valuable message about safety…


Campus Calamities presents
All-Nighter
#004

But secondly…

Some house keeping

Calendar at a glance

You are here

Week	Dates	Lecture	Reviews	Demos	Assessment submissions
1	22/2 – 26/2	Introduction			
2	1/3 – 5/3	Principles of Mechatronic Systems design			Problem analysis
3	8/3 – 12/3	Previous years deconstruction case studies			
4	15/3 – 19/3	Professional Engineering Topics	Progress review 1		
5	22/3 – 26/3	PCB design tips			
6	29/3 – 2/4*	Introduction to firmware design			
Break	5/4* – 9/4				
7	12/4 – 16/4	Your soldering is (probably) terrible	Progress seminar	25% demo	
8	19/4 – 23/4	No lecture!			
9	26/4 – 30/4	Q and A sessions		50% demo	
10	3/5* – 7/5	Q and A sessions	Progress review		
11	10/5 – 14/5	Q and A sessions		75% demo	Preliminary report
12	17/5 – 21/5	Q and A sessions			
13	24/5 – 28/5	Closing lecture		Final testing	Final report and reflection

Progress Review PAFs

- Please make sure you fill out your PAFs and return them via email – not every one has done this yet.
  - Still waiting for outstanding PAFs from one or more students in teams 1, 3, 5, 7-9, 11 and 16
  - No stress, just shoot them through.

- I will start following up with students who haven't submitted them yet, in the next couple of days

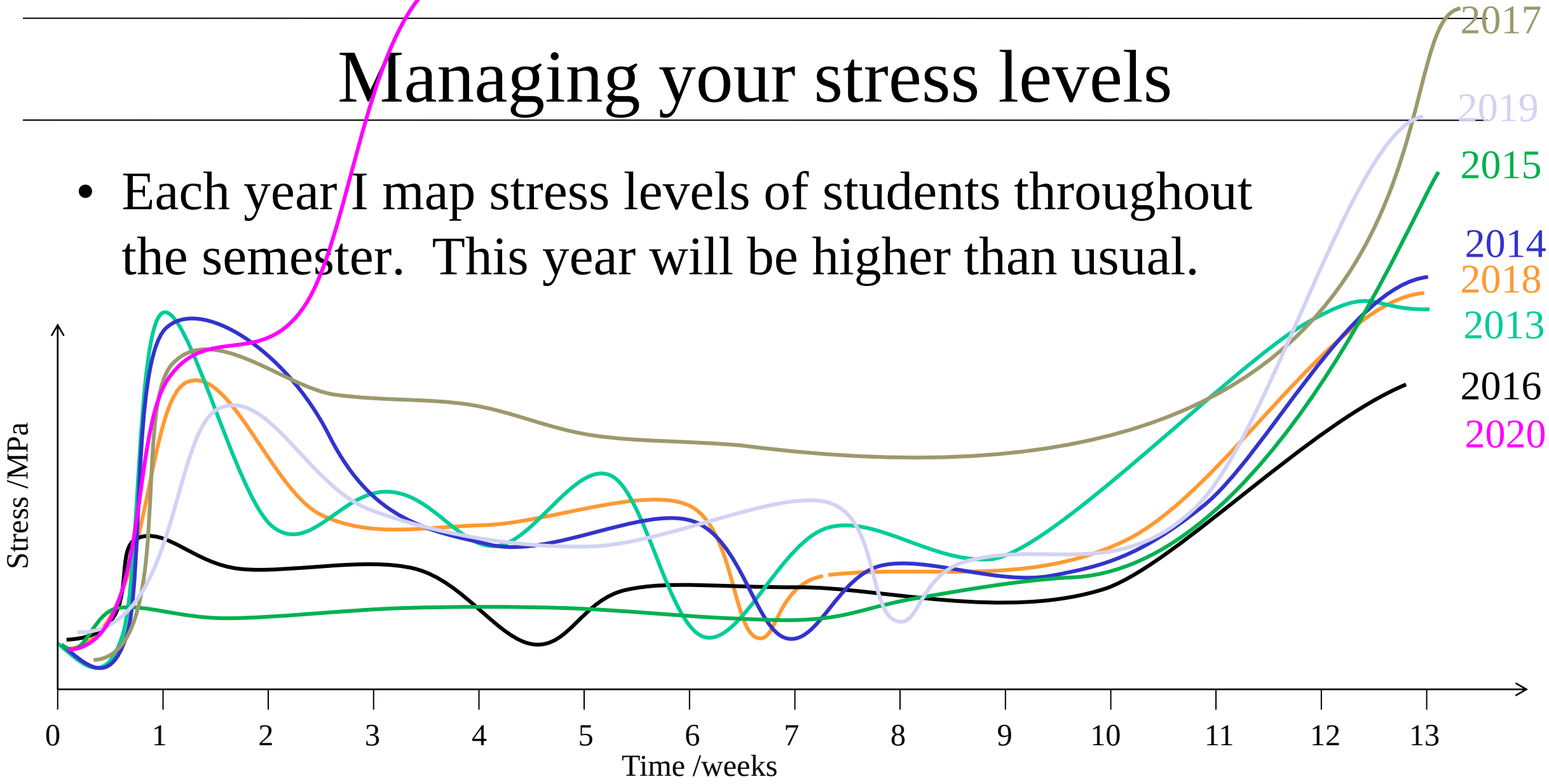
PAF 1 results

- I *almost* now have complete PAF 1 results, but I cannot post them until I have every result from a team
  - A PAF above 0.9 is good – all systems go, keep it up.
  - A PAF below 0.9 means room for improvement.
  - A PAF below 0.6 is a warning sign.

- No PAF 1 result on Blackboard?
  - Check that your team mates have submitted their PAF
  - They all have, but still no PAF result? Let me know!

Managing your stress levels

- Each year I map stress levels of students throughout the semester. This year will be higher than usual.



As always…

- Ask Pauline!
  - Pros: very happy to help, usually available, always attentive, genuinely loves the course and (the vast majority of) her students.
  - Cons: not a miracle worker, sleeps one third of the day, terrible dress-sense, has no rhythm.

- If you have queries or concerns, shoot me an email.
  - If you ask me now, I can change things *before* it's too late!
  - Much harder to fix any problems by the time we hit week 13…

FAQ Roundup

- **What fans can we use?**
  - Professionally produced, commercially-available units – as a discrete module or as independent fan/duct units designed to interoperate. If you need to quibble about the meaning of "commercially-available" you're probably not going to be allowed do it.
- **Can you tell me if our approach to the task is rules-compliant?**
  - Sure, send it through via email!
- **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?
- **Do we need a Powerpoint presentation for the seminars?**
  - It doesn't have to be Powerpoint, per se, but a slideshow is strongly recommended. If you have physical items to show off, think in advance how to best mix screen sharing with live content. I'll talk more about this shortly.


ANY QUESTIONS?

Progress seminars

- Progress seminars are Week 7 – right after the break!
  - First group-based assessment
  - Gives you presenting experience and brings us up to date with your team's progress
  - In person for flex students, online via Zoom for external students: https://uqz.zoom.us/j/81705489650

- Sign up for session slots via Doodle poll
  - Sign up here: https://doodle.com/poll/aqkwp86ga536tg7r
  - (Link will also go up on Blackboard shortly, closes Friday)

Progress seminars

- Group presentation – 10 minutes per team
  - Stand up and talk about your progress
  - Each person talks for roughly equal time

- Focus on progress, *not* the requirements!
  - We know what the project goal is (really!)
  - We know what your proposed solution is.
  - Don’t waste valuable time repeating them.
  - Just show us your ***progress***.

Progress seminars

- You will be ranked 66% on individual and 33% on group presentation, plus PAFs
  - Yep, more PAFs.
  - Submit in person or via email – best done right after the seminar!
- Recall the presentation tips and tricks from lecture 3 – expectations are high!

Progress seminars

- How to sign up:
  - Have **one and only one** member of your team nominate a time for your team on the poll
  - When they sign up, they must include their **full name and team number**. If they don't have both, the slot will be cleared.
  - That link again: https://doodle.com/poll/aqkwp86ga536tg7r
- If you absolutely can't get a slot that works for all of your group, email me ASAP
  - *But this should never happen*

Progress Seminar plans

- Assuming the lockdown ends as expected, we will do the normal in-person Progress Seminars in person.

- If the lockdown is unexpectedly extended, we will do abnormal virtual seminars virtually.
  - I will let you know via announcement as far in advance as possible.


ANY QUESTIONS?

Meditations on presenting online

- Just like regular meetings and presentations... except not!
- Many habits/reflexes of in-person meetings don't work:
  - Can't time discussion flow based on shared attention
  - Body language is often limited or entirely absent
  - Unholy AV problems
  - Problems are compounded when you have to co-present

<u>Mostly boils down to being prepared</u>

Be prepared

- Be on time – even 15 minutes early (!)
  - Check you have working invite links, passwords, settings, etc
- Check your AV equipment, turn off downloads/uploads
  - A broken camera is fine, a broken mic means disaster
- Check your personal grooming and background
  - Eliminate distractions: pets, kids, family members

Change your name, loser

- LeetH@x0rGibson81
- BigP1peLadyKiller69
- Weed4life420Blazeit
- ~~=ℜIℚ𝕌1ᘔ=~~
- xXx_MLG_NoScoper360_xXx
- ♜ Sɳιρєz▄▬┻═┬─
- YourName_1995

Be prepared

- Know the order in which people will speak, who will say what and how
  - One person driving a single shared screen, or switching across multiple shared screens through the presentation?
  - Unrehearsed switching can be awkward
- Have your notes/slides/files ready to go before you start
  - Open on your desktop, neatly arranged.
  - Perhaps across the task bar in the order you'll use them?
  - Hide anything potentially embarrassing… *ahem.*

Be prepared

- Check that your animations/effects play correctly over the meeting application – the dreaded framey ‘streaming’ video
  - Consider uploading to Youtube and link to participants in chat
  - If the meeting application and your presentation applications conflict, you want to find out *beforehand*.
- Implement necessary security precautions
  - Be mindful of where you post your meeting links
  - Do you need a regularly changed group password?


ANY QUESTIONS?

Practice!

- Practice using the meeting application
  - Muting and unmuting, turning video on/off
  - Sharing desktops, typing in chat, etc

- Practice presentation craft
  - Pacing and language
  - Transitions and hand-overs
  - Leave extra time in your presentation for technical “friction”
  - Consider question slides for questions.

Presentation manner

IF YOU ARE NOT SPEAKING,

MUTE YOUR %&@#$*! MICROPHONE

Presentation manner

- Look at the camera, not the screen or videos of other participants – the one time you *don't* make eye-contact!
- You won't have regular language cues: stay on topic
  - Don't ramble or trail off; nobody will jump in and save you
- Monitor the other participants/chat for questions
  - Ask to leave questions to the end, or leave pauses to answer those questions, if appropriate

Presentation manner

- If the internet connection gets choppy, kill your video
  - Not at all rude to ask others to do the same (via chat?)
  - ~~Not at all rude to threaten microphone offenders with murder~~
- Do not make it obvious you are drinking alcohol
  - Especially not when giving a lecture.

Other resources

Stuff I found useful preparing these slides:

- https://www.groovehq.com/blog/zoom-tips-and-tricks
- https://www.inc.com/kevin-daum/10-tips-for-giving-great-online-presentations.html
- https://www.gsb.stanford.edu/insights/10-tips-giving-effective-virtual-presentations


ANY QUESTIONS?

Onwards!

To firmware and beyond!

Firmware: OMG what is it?

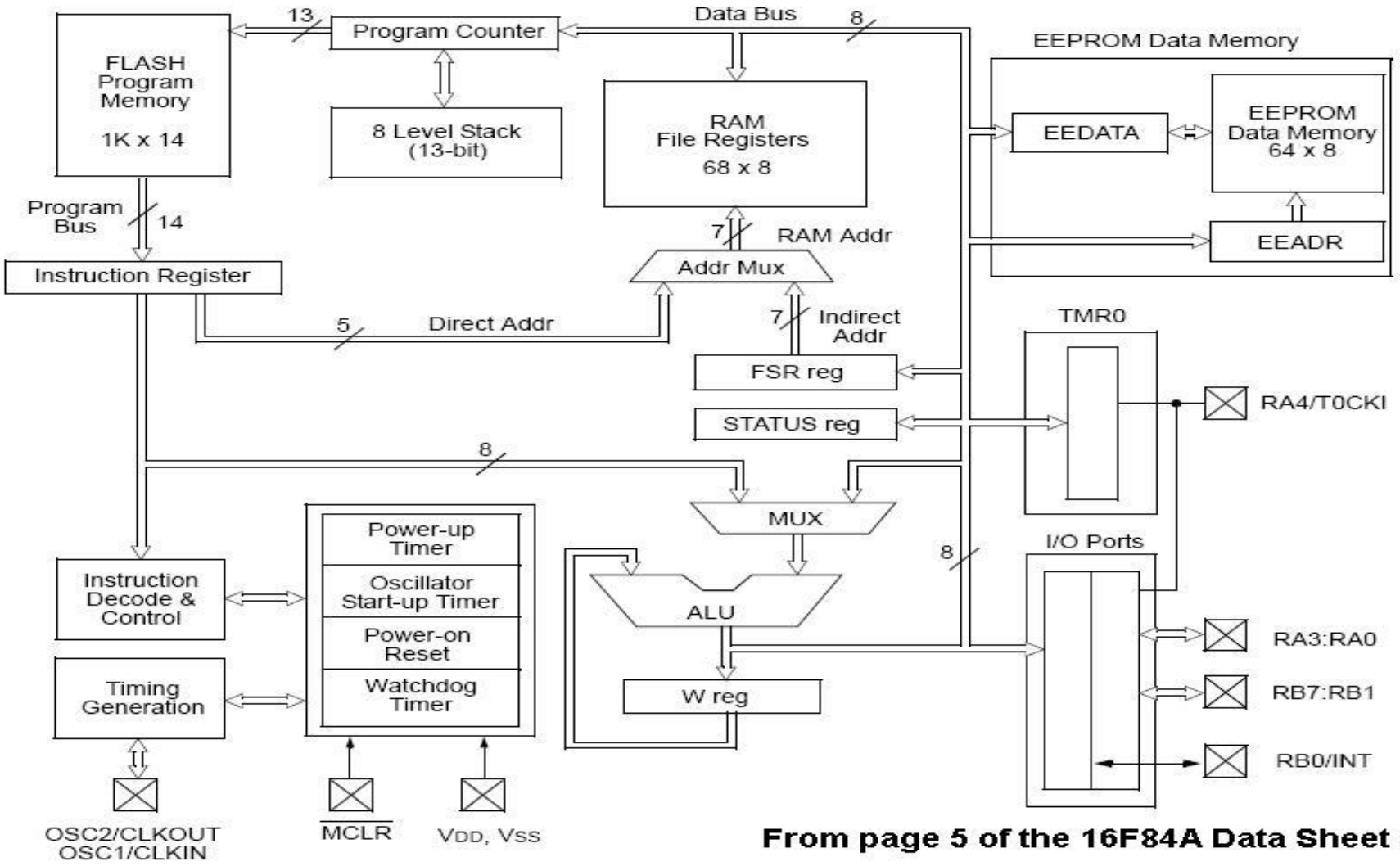
- The embedded machine code inside a microcontroller
  - Like software, only closer to hardware… so “firm”-ware
- Unique from software, in that it can write directly to device physical outputs (no OS mediation)
  - Straight from ‘The Matrix’ to the “Real World”

Let’s talk about microcontrollers…

Key elements

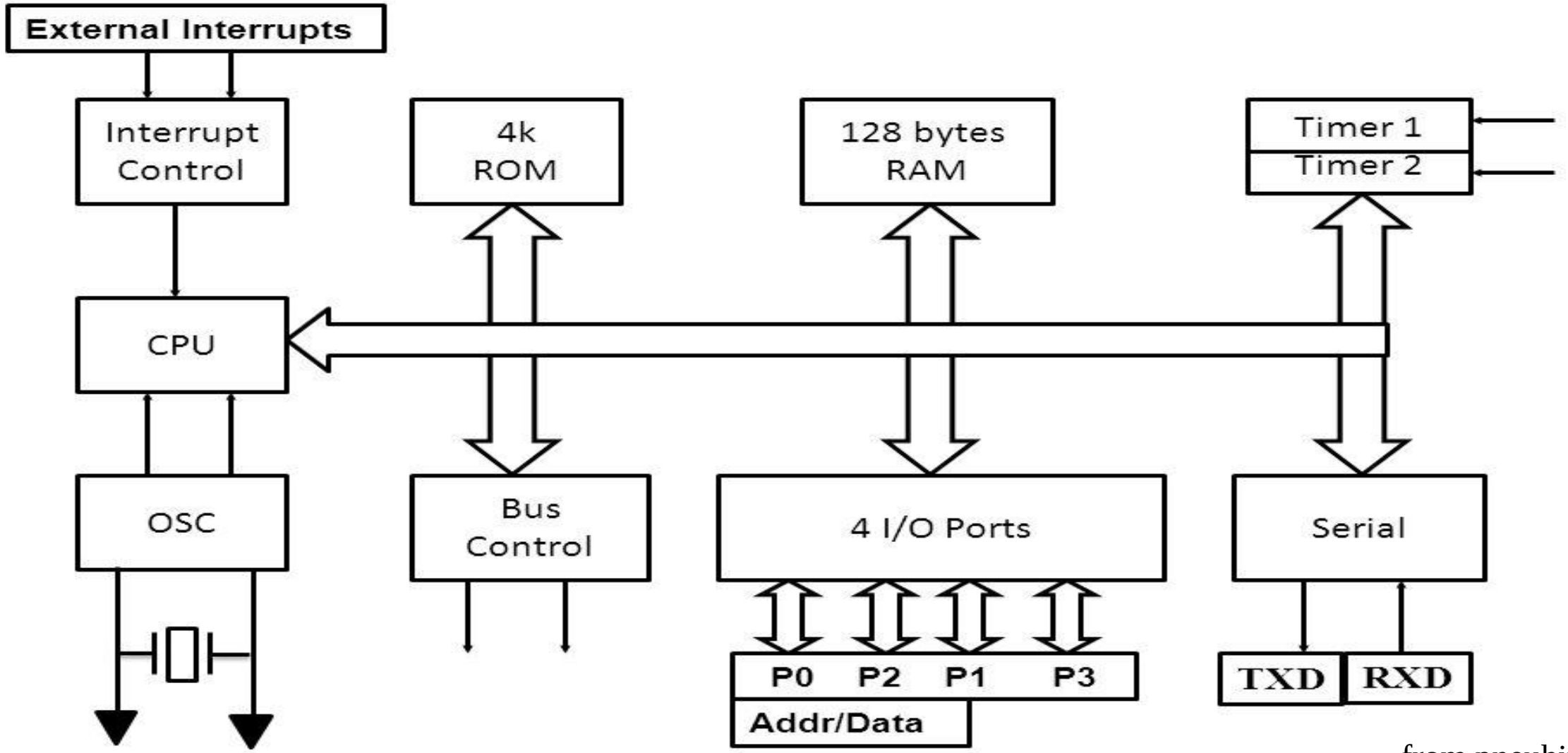
- Microcontrollers behave like a whole computer on a chip:
  1. Co-locate RAM with working memory
  2. Integrated data bus
  3. Internal clock sources and timing
  4. Input/output hardware (PWM, SPI, CCP, etc)
  5. Specialised hardware (graphics/audio/RF/etc)

The legendary PIC 16F84A



From page 5 of the 16F84A Data Sheet

Block diagram that thing…



from pneuhiver.info

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 > High-level Matlab > High-level OO languages

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


ANY QUESTIONS?

Some principles

- Structure everything
  - Structure is a lifestyle
- Modularise
  - Yes, even into separate files
- Test everything
- Use C (or maybe Python). Really.
  - It'll make it easier. You'll thank me when you're older

Structure is a lifestyle

- Don’t do ANYTHING without structure

> “Code without structure is chaos”
>
> – a really smart person, probably Dijkstra

- Axiom: comments help you understand code, but good code is readable without comments
  - Corollary: good, commented code is the most understandable code

Structure is a lifestyle

Major moving parts:

- Front matter
  - #define and all the nuts-and-bolts stuff: headers, macros, globals
- Main(){}
  - Your high-level logical structure should be visible here
  - Your program shouldn’t live in a function (although this is debatable)
- Client functions (aka sub-routines)
  - All the stuff commonly reused, outside of loops

Structure is a lifestyle

- On some level, you write code like a report
  - Start with a “skeleton” outline of what the program must do
  - Flesh out the details incrementally, block by block, function by function
- Of course, you’ll want to be testing each meaningful block of code as you go…
  - More on that later!

Structure is a lifestyle

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

This is all the “block diagram stuff”

Structure is a lifestyle

- 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;
```

Pseudocode magically becomes comments:

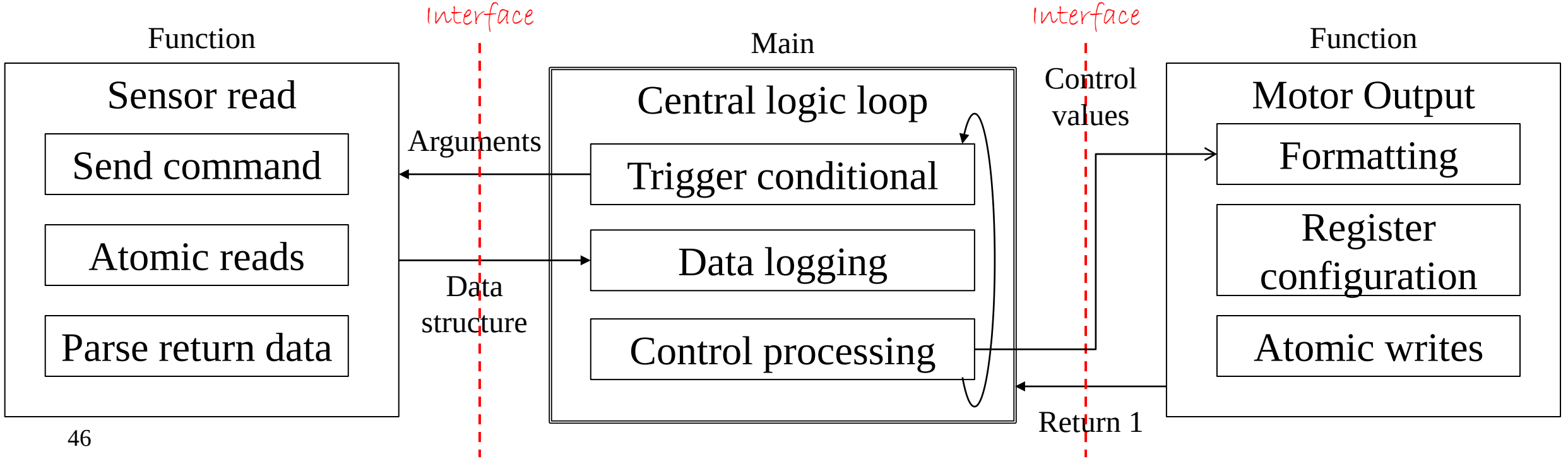
```
read(var); //Get next value
if(var > foo){ //Test value
 buf = var; //Store for reuse
}
```

Structure is a lifestyle

- 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, (mostly) easy to write!

Structure is a lifestyle

- Just like the systems engineering approach!
  - Ie. high level structure broken down into smaller subsystems
  - Boundaries between subsystems are defined by arguments/API




ANY QUESTIONS?

Modular Me

- Code reuse is the noblest way to hack
  - Ie. 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 all-too-common chore
- All of these strongly benefit from standardised interfaces

Modular Me

- Consider:
  - A specific header file for each particular microprocessor you use (almost mandatory!)
  - A separate source file and header for each major logical function
  - 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)

After a while, a source map isn't a bad idea…

Tales from project.c

```
#include “pic18f14k22.h“ //This device
#include "mpu9150.h"
#include "sst25vf032b.h“

#define clip(v,a,b) (((v) < (a))?(a):(((v) > (b))?(b):(v))) /*Enforce limits a<v<b */

#define CONFIG1H_ADDR                       300001 //Used for setting up 64 MHz clock

//TMR0 is to be setup to run at 1MHz for PWM width timing
#define TMR0H_RELOAD                        0xb1  //Set to rollover at 20000 for 50 Hz
#define TMR0L_RELOAD                        0xdf

#define NULL                                0
#define LED0                                (PORTC.F3) //Green
#define LED1                                (PORTC.F6) //Red
```

Example contents of a device header

```
#ifndef PIC18F14K22_H
#define             PIC18F14K22_H    //This stops duplicate definitions
/* Bit/pin name.... Register definition */
#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)
```


ANY QUESTIONS?

Use C

- C is the screwdriver of firmware languages
  - The number of things it's good for vastly outnumbers the things it isn't good for
- Almost everything compiles C
  - Everything embedded DOES compile C
  - C++ compiles down to C at build time
- If your firmware needs something C can't do, you're (probably) doing it wrong

C comes in many flavours

- C++: object oriented and convenient “//” comments
- C#: object oriented for web, mobile; memory management
- MISRA C: extra checks and type protections for reliability
- C11, Annex K: security features via libraries

Special mention:

- CPython: the thing that actually runs Python scripts
  - Python is useful for OS-side interfaces to your embedded system

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
   }
}
```


ANY QUESTIONS?

Rules for beginners (and everybody)

- Keep it simple – *really* simple!
  - Short routines, broken out into files by function
- Be one with your micro
  - Fully understand it and the circuit around it
  - 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!

Based on the “15 rules” by 123Jack

Testing is a lifestyle

The opposite order of coding!

- Test small snippets first, then larger snippets, and finally the whole program all-up
- Think about what you need to test the functionality
  - Stubs and Boilerplate
  - Test data for normal operation and expected failure modes
  - Can you automate testing?

What about the Powah?

"But… I was promised TRUE POWER!

This sucks…"

Ok, with well-crafted

firmware and hardware…

What about the Powah?

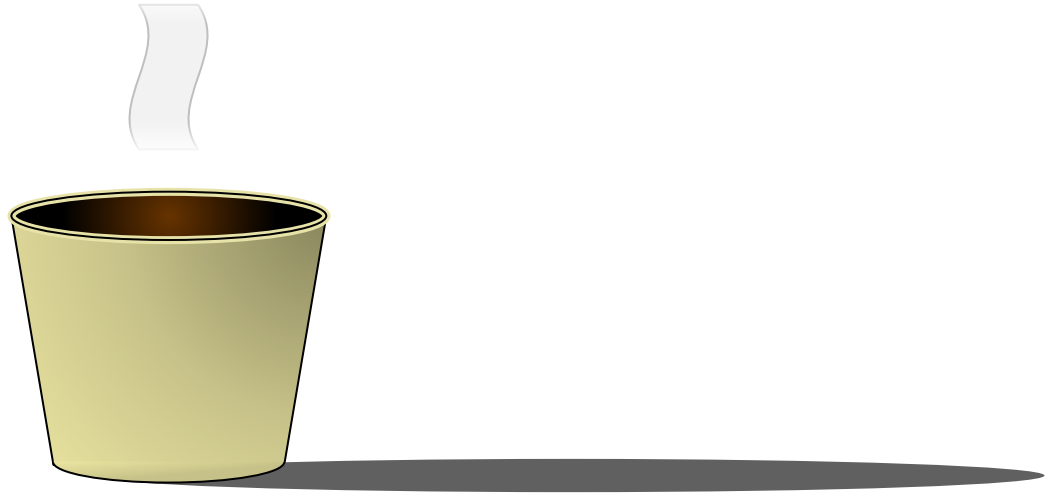
- You can get precision instruction timing that can be accurate down to *picoseconds*
  - I designed a nano-second timer for GPS in C
- You can build a drone from $10 of parts, in a circuit the size of a 10c coin.
- You can build a remote sensor that works for five years on a single coin cell.

Join ussss

- So yeah… firmware is important

- Embrace structure
- Embrace C
- Embrace TRUE POWER

Questions



“By caffeine alone do you set your mind in motion”

Tune-in next time for…

Your Soldering is Terrible (probably)

or

"How I learned to stop worrying and love the flux"

Fun fact: Lockheed Martin and Boeing write avionics code in C, C++ and Ada