Introduction to Firmware Design

or

"Firmware: harder than software."

P Pounds

26 March 2018

University of Queensland

But first…

Some house keeping

Calendar at a glance

Week	Dates	Lecture	Reviews	Demos	Assessment submissions
1	19/2 – 24/2	Introduction			
2	26/2 – 2/3	Principles of Mechatronic Systems design			Problem analysis
3	5/3 – 9/3	Professional Engineering Topics			
4	20/3 – 24/3	Introduction to Practical PCB Design	Progress review 1		
5	19/3 – 23/3	Your soldering is (probably) terrible			
6	26/3 – 29/3	Introduction to firmware design			
Break	30/4 – 13/4				
7	16/4 – 20/4		Progress seminar	25% demo	
8	23/4 – 27/4				
9	30/4 – 4/5			50% demo	
10	8/5 – 11/5	No lecture	Progress review		
11	14/5 – 18/5			75% demo	Preliminary report
12	21/5 – 25/5				
13	28/5 – 1/6	Closing lecture		Final testing	Final report and reflection

You are here

Hooray!

Progress seminars

- Progress seminars are next teaching week!
  - First group-based assessment
  - Gives you presenting experience and brings us up to date with your team's progress
- Sign up for session slots via Doodle poll
  - Link to poll will be sent out via Blackboard announcement after the lecture (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 marked 50% on individual and 50% on group presentation, plus PAFs
  - Yep, more PAFs.
- 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.
- If you absolutely can't get a slot that works for all of your group, email me ASAP
  - *But this should never happen*

Lecture nominations

- This is the last of the regular lecture corpus
  - Nominated topics from here on out
  - If nothing is nominated, then it will be a Q&A

- Topic polls will be conducted in class
  - Hey, let's do that now!

OMG the gimbles are done

- Blaw... They should have been in the lab last week, but FWG forgot an M4 screw (?)
  - Also, we had to repack the bearings to make them run nice and smooooth
- All done now, though – they’re in the lab!

FAQ Roundup

- **None as of yet**

Onwards!

To firmware and beyond!

Firmware: OMG what is it?

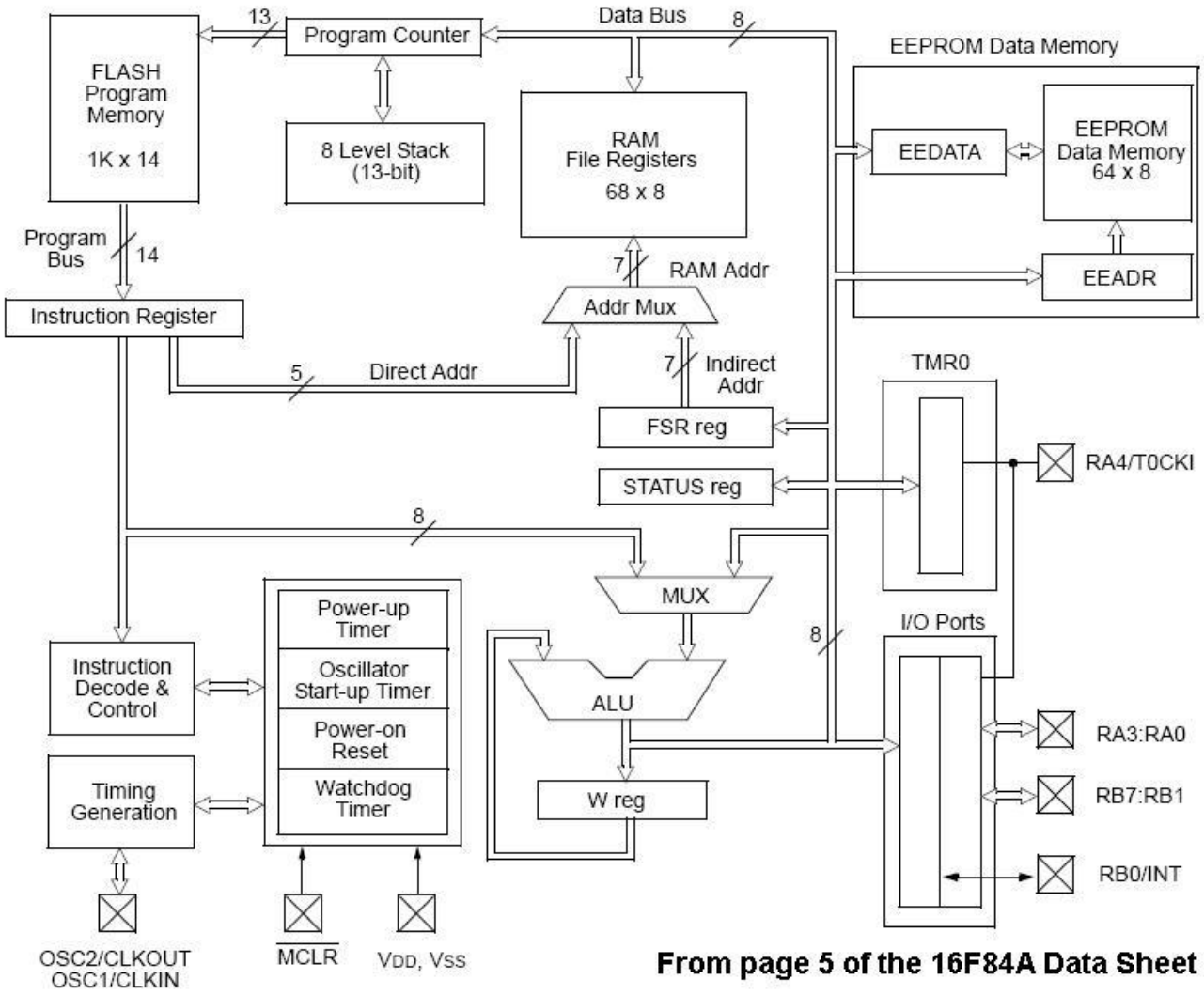
- The embedded machine code inside a microcontroller
  - Like software, only closer to hardware… so "firm"-ware
- Unique from software, in that it can write directly to device physical outputs
  - Straight from The Matrix to the "Real World"

Let's talk about microcontrollers…

Key elements

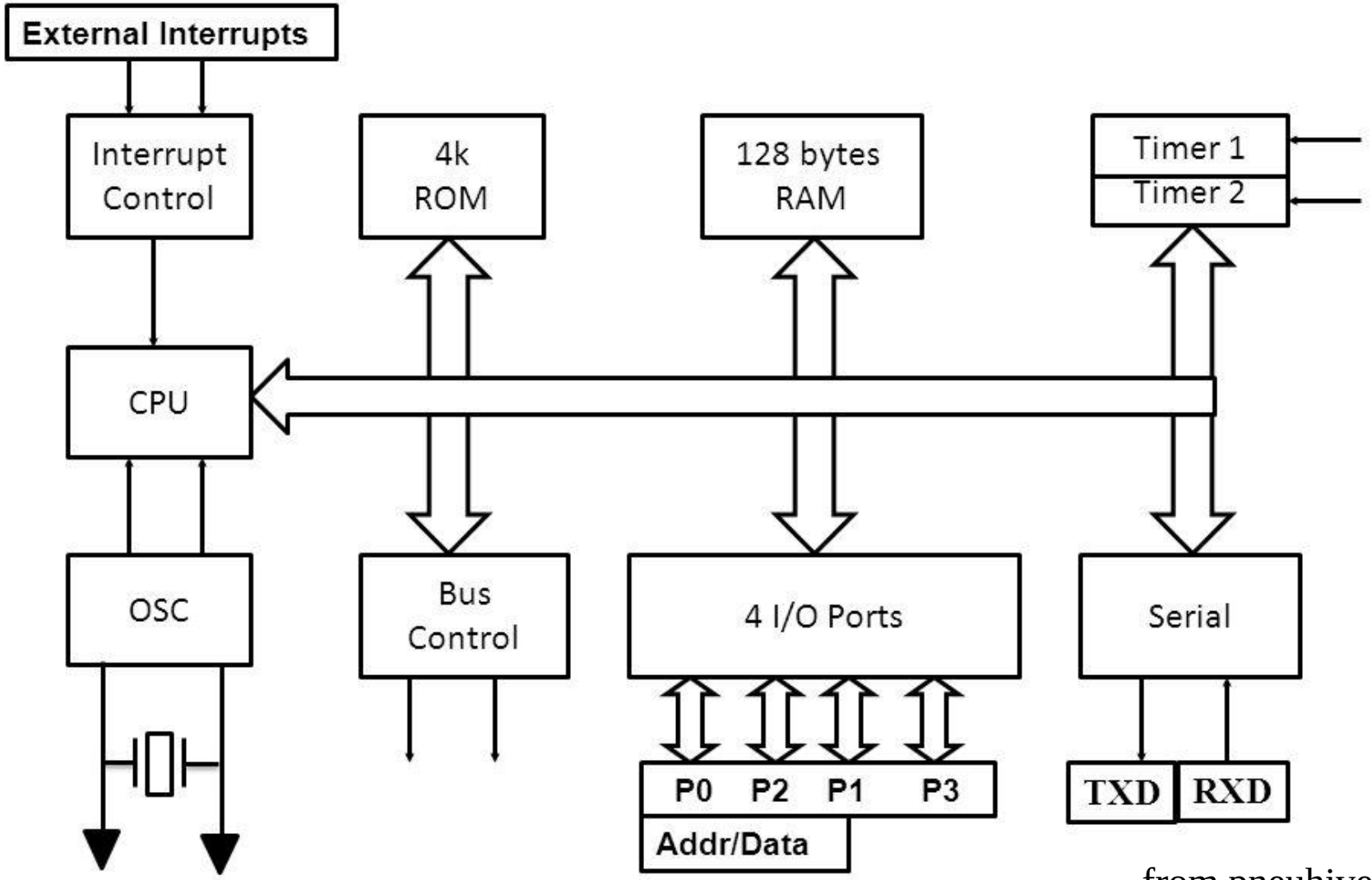
- Microcontrollers behave like a whole computer on a chip:
  1. Co-locate RAM with working memory
  2. Integrated data bus
  3. Internal clock sources and timing
  4. Input/output hardware (PWM, SPI, CCP, etc)
  5. Specialised hardware (graphics/audio/RF/etc)

The legendary PIC 16F84A



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
- "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.
  - 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

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

Structure is a lifestyle

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

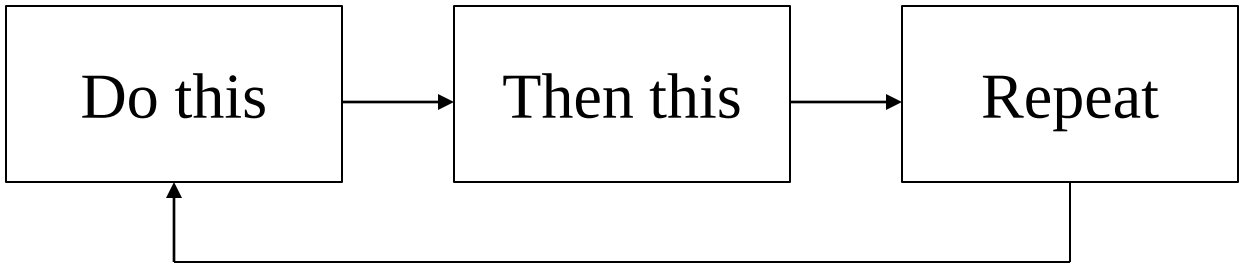
Structure is a lifestyle

- 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 funtional block of code as you go…

Structure is a lifestyle

- 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”



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;
```

Then:

```
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, 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 all-too-common chore

Modular Me

- Consider:
  - A specific header file for each particular mircoprocessor 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:

```
hardware_setup();
main(){
   while(1){
        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
        }
   }
```

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

Tune-in next time for…

Questions and Answers Vol. 1

or

"The quick and the decaf"

Fun fact: Boeing writes avionics code in C.