Questions and Answers Vol. 2

or

"By the bean alone do you set your mind in motion."

Paul Pounds

11 April 2017

University of Queensland

But first…

Some house keeping

Calendar at a glance

Week	Dates	Lecture	Reviews	Demos	Assessment submissions
1	28/2 – 3/3	Introduction			
2	6/3 – 10/3	Principles of Mechatronic Systems design			Problem analysis
3	13/3 – 17/3	Professional Engineering Topics			
4	20/3 – 24/3	Introduction to Practical PCB Design	Progress review 1		
5	27/3 – 31/4	Your soldering is (probably) terrible			
6	3/4 – 7/4				
7	10/4 – 14/4		Progress seminar	25% demo	
Break	17/4 – 21/4				
8	24/4 – 28/4				
9	1/5 – 5/5			50% demo	
10	8/5 – 12/5		Progress review		
11	15/5 – 19/5			75% demo	Preliminary report
12	22/5 – 26/5				
13	29/5 – 2/6	Closing lecture		Final testing	Final report and reflection

You are here

Progress Seminars

- Third done… only two thirds to go!
- Most groups have done pretty well so far
  - But not everybody…
  - Some teams got a *stern talking to*
- But overall teams are reasonably prepared
  - Yay!

Progress Seminars

- Also, if you hadn't noticed, Friday sessions were moved to Wednesday due to holidays
  - Totally my fault, mea culpa, it'sa meesa Jar Jar.

- Note, this creates an awkward clash with the expected incremental demo times…
  - Hmmmmmmmmmmm
  - What to dooooo….

Incremental demo

- So, this sucks, but we have two options:
  1. I have one of the tutors do the demos during the prac sessions while I'm marking seminars
  2. I run the demos after the prac sessions outside of class time.

- If you have a strong opinion, let me know
  - Otherwise, it'll be option 1. during the Thursday prac in the following order:
  - Team 14?, Team 10, Team 6 (am I missing any?)

Breaking news

- Keith Lane reports the testing tank is no-go for this week
  - He fought a valiant effort, but succumbed
  - He promises ready for Week 9 for sure, for sure
- A revised backup plan is to use an ENGG1100 shallow tank for testing
  - Similar width, but only 600 mm deep.

FAQ Roundup

- **None as of yet**

<eyeofsauron>

Some comments on professionalism

A couple of incidents recently have greatly concerned the tutors and me:

1. A student broke one of the torpedoes, and when the tutor asked who was responsible, no one came forward.
2. A student disconnected the fan on one of the 3D printers, resulting in damage. Again, nobody took responsibility

Some comments on professionalism

- In industry, your success depends on your professionalism and that of your peers
  - Once you lose your integrity, no one will ever trust you or your company again – see VeeDub
- A problem (even caused by you) brought to light can be fixed
- A problem that festers in the dark can destroy careers and lives.

Some comments on professionalism

A professional:

- Works in good faith to the best of his or her skills and knowledge
- Deals honestly with clients and coworkers
- Takes responsibility for problems he or she may have caused
- Reports unsafe conditions and unethical conduct to the relevant authorities
  - That can include outside the company

Some comments on professionalism

- I know I come across as scary and stern
  - This is intentional; and some students need that
- However, I am foremost committed to ensuring that all students achieve their best
- I will not get mad if you bring problems and honest mistakes to my attention (no, really!)

Onwards!

To firmware and beyond!

Mini Lecture on Firmware

- Seems like firmware is a really good topic to do a lecture on, but something that really requires the due time and focus to write a really cogent and insightful corpus of slides.

- So… here's a mini lecture to get you going sooner rather than later!

Firmware: OMG what is it?

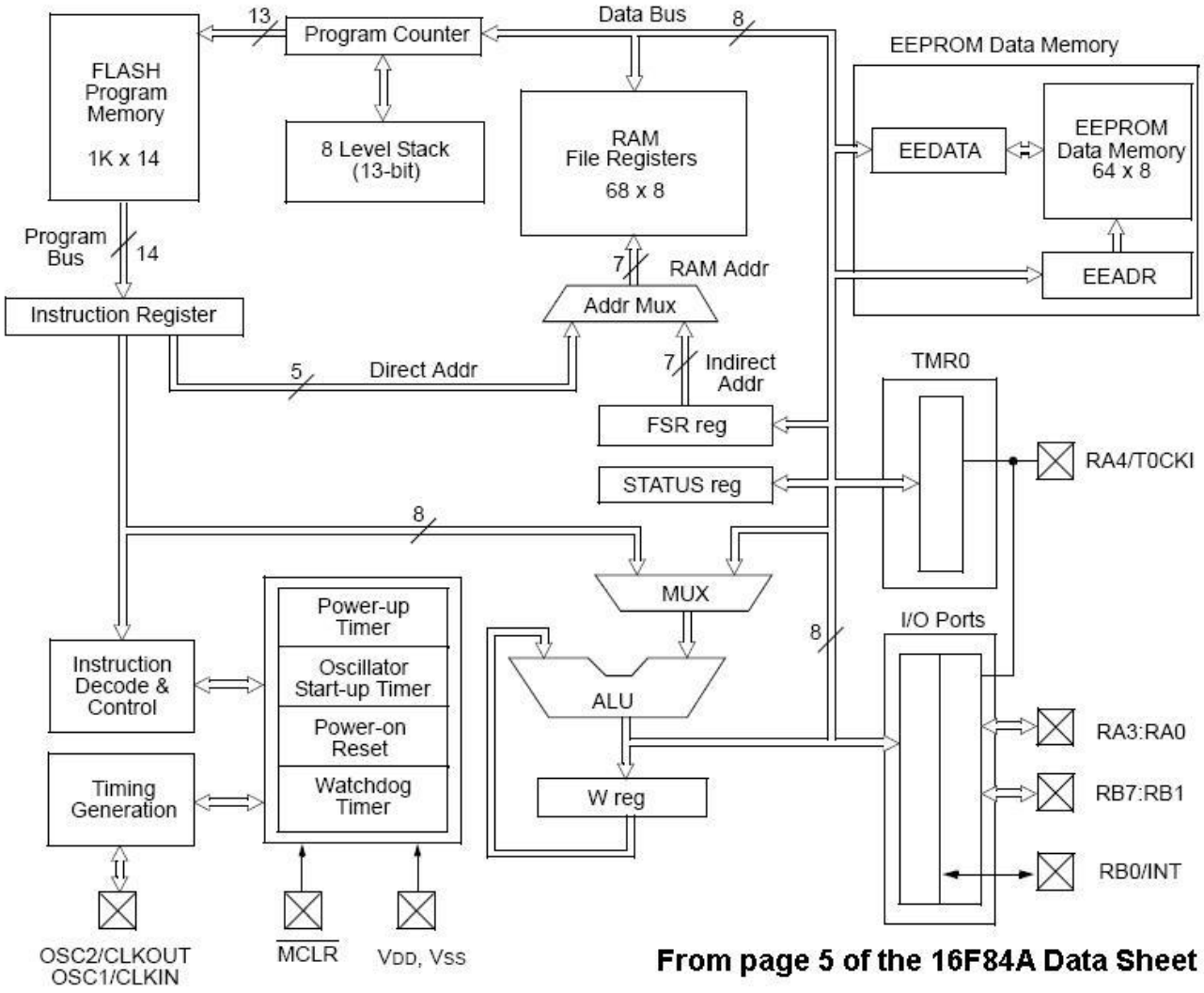
- The embedded machine code inside a microcontroller
  - Like software, only closer to hardware… so "firm"-ware
- Unique from software, in that it can write directly to device physical outputs
  - Straight from "the Matrix" to the "Real World"

Let's talk about microcontrollers…

Key elements

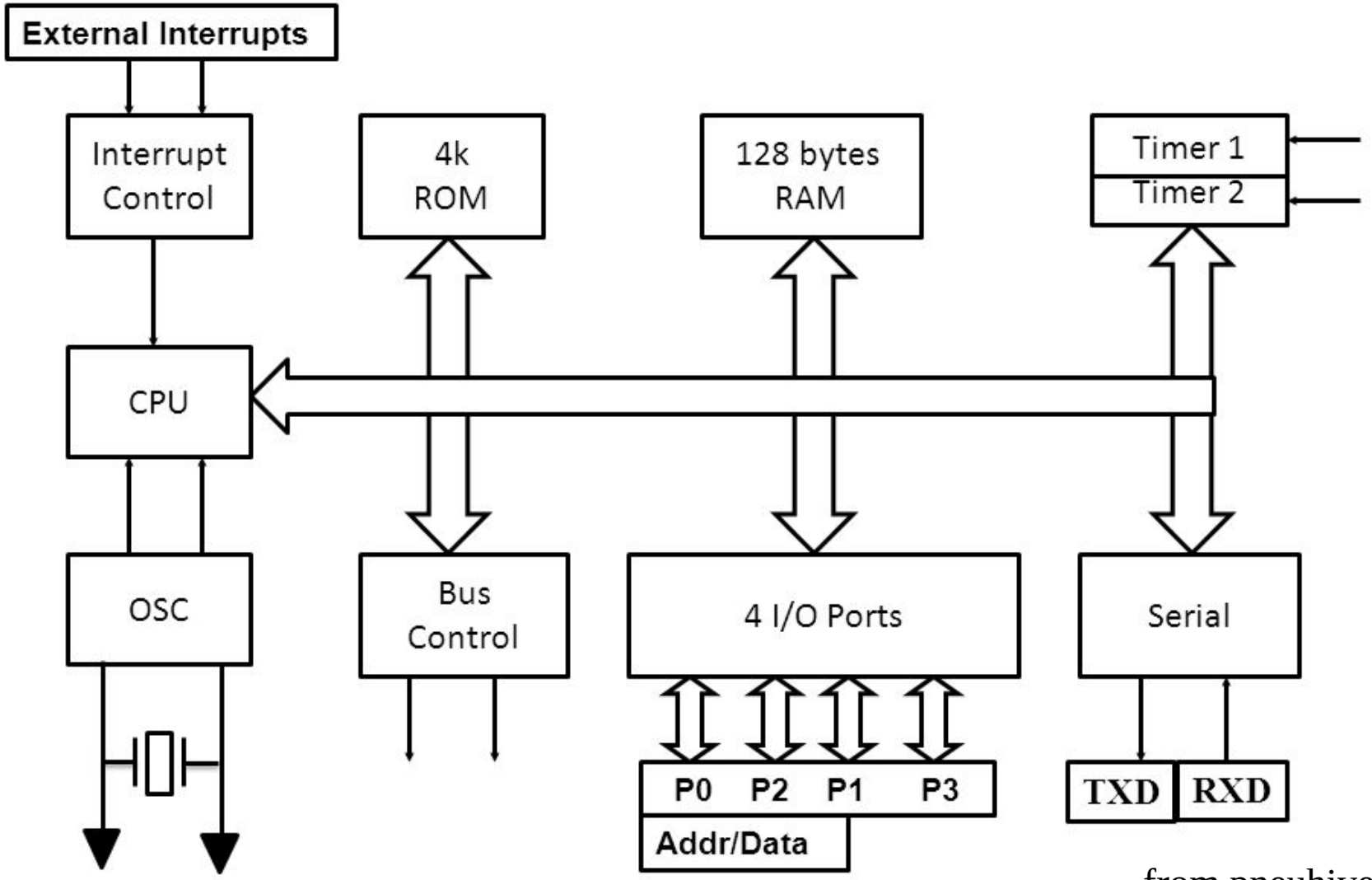
- Microcontrollers behave like a whole computer on a chip:
  1. Co-locate RAM with working memory
  2. Integrated data bus
  3. Internal clock sources and timing
  4. Input/output hardware (PWM, SPI, CCP, etc)
  5. Specialised hardware (graphics/audio/RF/etc)

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

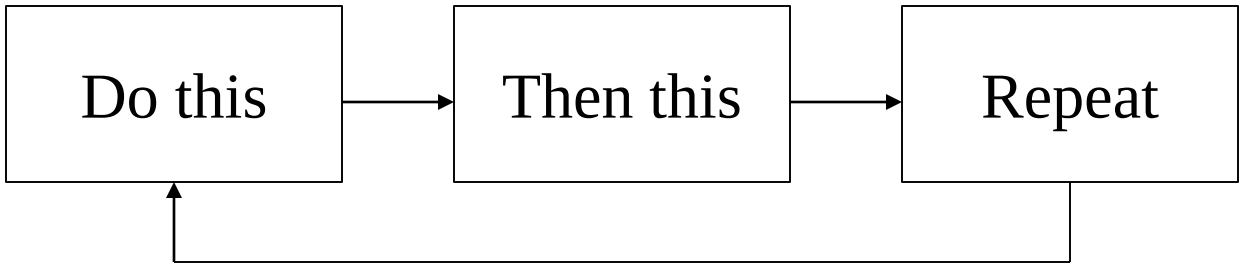
Major moving parts

- 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

- 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

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

And now…

Gratuitous project tips!

Simple simple simple

Robust robust robust

Test test test

(and test again)

Gratuitous project tips!

- Some things engineers *never* try to build if they can buy, copy or otherwise avoid it:
  - Power supplies
  - Motor drivers
  - Analog amplifiers
  - Inertial Measurement Units
  - Sensor fusion and estimation algorithms
  - Vision processing libraries

Gratuitous project tips

- You almost certainly don't need silky smooth video.
  - Why waste all your processor cycles just passing data around? Don't pipe data through a micro unless it's completely unavoidable.
- There is more to wireless communications than Wifi and Bluetooth… seriously
  - FM? Xbee? Zigbee? Smoke signals?

Gratuitous project tips

- Nobody is thinking about manipulation nearly enough
  - **Nobody** has a good solution yet
  - Grabbing the submarine is the hardest part of this project – ***disregard it at your peril!***

- How are you going to line up on the submarine, anyway?
  - Seems hard… hmmm

Gratuitous project tips

That's all for now!

But maybe more later…

Questions

Tune-in next time for…

Questions and Answers Vol. 3

or

"The quick and the decaf"

Fun fact: Boeing writes avionics code in C.