8 Hardware Design - An Industry Perspective

Contents

8.2

Introduction

The following tips and perspectives are from a UTS student, Tom Elliot, who graduated in 2012 and moved to Silicon Valley to work for a small startup called Fitbit…

In Tom's words:

> *Here's a start for things I never learnt as a student. Some of this is probably specific to high volume production, but definitely principles to think about with any hardware design. Again, I'm not sure how or if you should include this type of content in the curriculum, but as a uni that prides itself on being practical, I can see this kind of understanding being a good differentiator for UTS grads.*

On a return trip to Sydney, Tom showed me the product he helped design:

Figure 8.1 – The Fitbit Surge

8.1 Printed Circuit Assembly

You don't just do one PCA build, you do several:

8.1.1 Non-form-factor (NFF)

Generally build as many as are needed for development - e.g. one for everyone on the firmware and hardware teams.

- A giant version of the hardware design. Generally the rule of thumb is to make it as giant as you think you can get away with. For a Fitbit device, the NFF would usually be 30cm x 20cm.
- Leaves room to cut traces and rewire as necessary.
- Makes it way easier to develop firmware.
- Use full size programming and debug headers. (Rather than having a 20pin JTAG header precariously dangling off the side of your form factor design, or a feature-limited 2 pin programming header).
- Add plenty of headers to allow for probing or modifications. The NFF for Surge had every single MCU ball run through a 2 pin header. By default the pins were shorted by traces on the board, but cutting the traces then lets you use the header to connect/disconnect/probe/rewire any of the MCU pins.
- Headers on the VCC of components also allow for easier current/power profiling.
- You can make it modular, so you have the host MCU on a giant base-board, and you can plug in a daughterboard with the display on it, the BT daughterboard, etc. This helps for a few things: For example if you have to change the display vendor you can keep working and swap out the module as it becomes available. Alternatively if you can't source enough parts (maybe you're getting parts that aren't actually in production yet, so you only get them as they roll out of the factory), then you can still build the base-boards and get started on development.
- Obviously any RF will need to be closer to the intended design, but even that doesn't have to be perfect. You don't need performance at this stage, it's just for enabling H/W mods and as a firmware development platform.

8.1.2 EVT/DVT/PVT/MP

- This is the hardware equivalent of agile development. As hardware timelines are so much longer than software, there's not really much opportunity for iteration. Instead multiple parallel options are investigated and narrowed down until a single design is chosen.
- AFAIK it's what's used at Apple, amongst others.
- There are 3 build stages to progress through before getting into Mass Production (MP): Engineering Verification Test (EVT), Design Verification Test (DVT), Production Verification Test (PVT).
  See here for the distinction between Verification and Validation:
  https://en.wikipedia.org/wiki/Verification_and_validation
  There's some good info here:
  https://www.instrumental.ai/blog/2016/11/14/hardware-engineers-speak-in-code-evt-dvt-pvt-decoded
  and here:
  https://www.instrumental.ai/blog/2016/11/8/hardware-engineers-get-more-dates-why-hardware-schedules-look-nothing-like-software-schedules.
- Each of these 3 stages has a bunch of phases inside it. There's a design phase (work out what's getting built), a manufacture phase (go and make it), and then the actual testing phase (does it do what we need it to do?). Once you've verified that the design criteria are met, you "exit" the stage and move onto the next one. Exit usually involves meetings with the boss people, and a bunch of spreadsheets with cells shaded like a traffic light to describe how much of a catastrophe things are.
- The idea is to exit EVT with confidence that the chosen design works, to exit DVT with confidence that the design can be manufactured at scale, and to exit PVT with confidence that the production line can meet production requirements. Generally there's a lot of push and shove between product managers, company leaders and engineering teams around the schedule, and those discussions often end up blurring the lines around exiting the different states and moving onto the next one.

8.2 Manufacturing Testing

- When you build a one-off, or something in the lab, you solve any problems that come up as you build it. Manufacturing doesn't work like that.
- A design might be fully functional 90% of the time, but fail 10% of the time. There's a good chance you won't catch this failure until you've started making more than just one-offs. If the business case allows for 10% failure rate that's fine, but you still need to know which ones are failures. If you can't afford 10% failure you need to know which ones are failures and find the root-cause.
- Manufacturing testing is a whole other engineering and design problem that gets solved for everything that gets mass produced, but is far less transparent.
- There's various types of testing
    - Panel testing: you put an assembled panel of DUTs into a test fixture and it runs a series of tests on each DUT.
    - One-up testing: usually used for offline diagnosis - a fixture for a single PCA with test points from the PCA broken out to make them easy to access.
    - Process testing: Test that a specific process passes (e.g. waterproof seal on a waterproof product).
    - Final assembly test: Once the product is assembled, a quick run through to make sure everything still works, program any final firmware, record any last serial numbers, etc.
- For on-the-line testing, during EVT you might want to test every single device. During DVT you might test 50%. During PVT you might test 1%.
- There's a whole separate set of testing that goes into the verification side of EVT/DVT/PVT. That's not done on the line, and includes things like functional testing, radio testing, ESD testing, wear and tear testing, FCC testing, etc.

8.3 Research vs Product Development vs Engineering vs Test

- As an electrical engineer there's not just one "Electrical Engineer" job that you can get. You can be an Electrical Engineer in a variety of roles. None of the roles mean you just get to design electronics all day every day.
- Research: Speculative investigation. A much broader role than just electrical engineering.
- Product development: How do you get the research into an actual product? It can be challenging because it's never been done before (and now it's subject to some arbitrary schedule). Interfacing between the Research people and the product team, so you're often stuck between conflicting incentives (research wants to get their thing right, engineering want to build something on schedule and on budget).
- Engineering: You actually design and build the product. Design is usually 35% of the work on a product and happens at the start of the project. Coordinating with suppliers and manufacturers, root-causing issues, updating the design, project-managing, etc. make up the majority of the work. For a project that takes 12 months to go to market, expect to spend 3 months of the year doing design. For the initial design, maybe a month doing the parts research and schematic design, maybe a week doing the PCB layout. Over the remaining builds there'll be schematic updates and another handful of PCB layouts to complete.
- Test: Making sure the verification part of EVT/DVT/PVT happens. Designing/implementing the Manufacturing Test equipment.

8.4 Git

No one likes SVN anymore.

8.5 Python

- Very versatile (I've used it to talk to MCUs over serial, do statistical analysis of manufacturing test results or for a web server).
- Very quick to learn.
- Very quick to make something to do a job.
- Very well designed.
- Compared to C for embedded systems, this is the exact opposite end of the spectrum in terms of abstraction (you never get anywhere near the hardware) and design principles (you don't have to try and jimmy OOP onto Python, everything is an object). It's nice to see what the other side is like and it's been super useful. Also maybe good to have an idea of how different it is to program in different languages and for different use cases.