Embedded TDD

For Cambridge Software Crafters
13 March 2024

Brice Fernandes
brice@fractallambda.com

Logistics and Wifi

🚽 Behind the main screen to the right of the corridor.

🛜 Wifi is **The Bradfield Centre** password is **Ca3Br1d5e**

🚨 We do not expect alarms. Assume a fire alarm is real and make your way to the car park.

Slides and code available on Github

github.com/bricef/embedded-tdd-katas

Plan for this evening

1. Intro
2. What is "Embedded"
3. Embedded craftsmanship
4. Using Replit
5. The Katas
    1. LED Driver Kata
    2. Interrupt Kata
6. Recap

Why this talk?

TDD Refresh

TDD Loop

Write a failing test

Make the test pass

Refactor

Ping Pong TDD

What is "Embedded"

Embedded constraints

- Resource constraints (RAM, CPU)
- Lack of standard libraries
- No or limited filesystem
- Limited Interface (serial? UART, SWI)
- No Operating System
- No standard library
- Direct hardware access
- Lack of MMU/PMMU

Special pains

- Late hardware delivery
- Hardware scarcity
- Hardware bugs
- Long target compile times
- Long target setup and upload time
- Compiler licenses

Embedded Strategy

Dual targeting

- Dual targeting
  - Simulate hard-to-duplicate conditions
  - Get around target bottleneck
  - Running the test suite locally
  - Automated CI

Embedded TDD Cycles

Automated HW tests

- There's no reason why you can't create an automated harness that runs the unit test on test devices.
- There's no reason why your CI builds couldn't use HW tests. Including cloud runners!
- You might want to ship tests in production devices as part of a HW self-test suite.

We won't go into depth in this topic tonight.

Test Doubles

- Crititcal for embedded
- Mock the HAL
- Mock the clock

How to Mock?

In order of preference

1. Link time substitution
   (Requires appropriate code structure)
2. Function pointer substitution
3. Syntactic substitution (preprocessor)

Combine at will...

Simulators

Can permit testing compiled target code in CI without target hardware.

Should be able to run your test suite.

Go hand-in-hand with Test Doubles.

TDD Test Cycles

Craftsmanship fundamentals still matter

SOLID

1. Single Responsibility Principle
2. Open Closed Principle
3. Liskov Substitution Principle
4. Interface Segregation Principle
5. Dependency Inversion Priciple

Let's get the party started!

Using Replit

Create a replit.com account

(use a throwaway email if you'd like)

1. Create a new Repl

2. Import from Github

3. Choose "From URL"

```
https://github.com/bricef/embedded-tdd-katas.git
```

4. Choose 'C' as a language

```
https://github.com/bricef/embedded-tdd-katas.git
```

Local alternative

If you're confident in your local toolchain

Clone the repository locally:

```
$ git clone https://github.com/bricef/embedded-tdd-katas.git
```

KEEP CALM

IT'S DEMO TIME

The Katas

LED Driver Kata

Look at `KATA.md` in `Code/src/leddriver`

Interrupt Kata

Look at `KATA.md` in `Code/src/interrupt`

Recap

What we learnt

1. TDD is possible and *useful* for embedded software.
2. Embedded TDD strategies make the process easier.
3. Dual targeting is worth it.

(C is fun, maybe?)

Further Reading

- TDD for Embedded C
- ThrowTheSwitch.org
- Unity Test Framework

Thank you 🙏

Q&A ❓

I'm available for contracting

brice@fractallambda.com