Test Interface für HAL Tests auf dem Target

Konrad Wöhrle

September 4, 2025

Contents

1 Einleitung

1.1 Motivation, Zielsetzung und Fokus

Das Projekt wurde im Rahmen einer studentischen Tätigkeit durchgeführt. Es dient der fachlichen Vorbereitung auf das Modul Embedded Systems sowie der praktischen Nachbereitung der bereits behandelten C/C++-Grundlagen.
Anhand eines kompakten Beispielprojekts mit Unit-Tests auf Host und Target soll ein gezielter Einblick in zentrale Aspekte im Embedded-Bereich vermittelt werden. Dafür ist eine Strukturierung der Software, insbesondere im Hinblick auf die Trennung von Applikation, Treiber und Hardwareabstraktion notwendig. Ergänzend wird das Verständnis zur Buildumgebung, zur eingesetzten Toolchain und zum Umgang mit Frameworks geschult. Zusätzlich werden grundlegende Arbeitsweisen mit Visual Studio Code und dem Versionsverwaltungstool Git vermittelt und angewendet. Die Inbetriebnahme eines Mikrocontrollers und die Kommunikation über eine serielle Schnittstelle runden die technische Umsetzung ab.

1.2 Anforderungen und Rahmenbedingungen

Technisch basiert die Umsetzung auf einem Mikrocontroller-Board (z.B. STM32-Nucleo), das über GPIOs, Timer und UART-Schnittstellen verfügt. Die Softwareentwicklung erfolgt in C/C++ unter Verwendung von PlatformIO und CMake als Build-System. Für die Kommunikation mit dem Target kommt HTerm zum Einsatz. Die Testumgebung umfasst sowohl Host-basierte Unit-Tests mit Google Test als auch Target-nahe Tests mit einem Framework namens wogtest, das Google-Test-Kommandos interpretiert, aber speziell für ressourcenarme Systeme konzipiert ist.
Organisatorisch ist das Projekt auf eine Dauer von wenigen Wochen ausgelegt und wird als Einzelprojekt durchgeführt. Die Dokumentation erfolgt parallel zur Entwicklung und umfasst sowohl technische Details als auch eine Bedienungsanleitung und eine Testübersicht.

1.2.1 Aufgabenstellung

Google-Test Interface für HAL-Tests auf dem Target
Google-Test ist ein etabliertes Tool für Unit-Tests und hat sich in vielen Bereichen der C++-Software-Entwicklung verbreitet. Im Embedded Bereich können damit die oberen Abstraktionsschichten einer Software getestet werden. Google-Test ist ein großes Framework und deshalb nicht (oder nur bedingt) geeignet auf den begrenzten Ressourcen eines Micro-Controllers zu laufen. Unit-Tests der HAL (Hardware Abstraction Layer) sind deshalb selten möglich. Die Fa. MicroConsult, bekannt durch Trainings für die Software-Entwicklung, bietet eine Lösung an. Diese nennt sich 'wogtest' (Without Googletest). Es bietet eine leichtgewichtige Lösung, die Google-Test-Befehle versteht. Dabei wird nur ein include benötigt. Die zugehörige Implementierung muss für die jeweilige Zielplattform erstellt werden. Wogtest ist frei und kostenlos samt Doku im Download von Microconsult erhältlich.

Die Herausforderung
Verschiedene Aspekte der Software-Entwicklung werden berührt und vertieft. Dazu gehören neben dem eigentlichen Test-Projekt auch Bereiche, die den Projektaufbau und die Build-Umgebung, sowie die Toolchain betreffen.

Aufgaben

1. Vorhandenes Google-Test-Template in Betrieb nehmen unter Linux (wsl)
2. Anlegen eines (lokalen) Git-Repositories
3. Inbetriebnahme eines Micro-Controller-Boards (z.B. STM-Nucleo) mit einer kleinen Beispielapplikation (Blinky). Debugging, Flashen, ...
4. Implementierung der Funktionen aus dem 'wogtest'-include-File
5. Aufbau eines Test-Templates für eine HAL-Komponente mit vorgegebenem Micro-Controller
6. Stubbing und Mocking in Tests
7. Erstellen einer Ausgabe-Schnittstelle für die Testergebnisse, z.B. über Segger RTT
8. Erzeugen eines Test-Reports
9. Erstellung der Dokumentation inklusive Anleitung im Markup-Format
10. Optional: Umbau der Build-Umgebung zu einem Docker-Container

Voraussetzung
Grundkenntnisse in und Spaß an der hardwarenahen Software-Entwicklung.

1.2.2 aus der Aufgabenstellung abgeleiteter Projektablauf

- Vorhandene Testumgebung in Betrieb nehmen
- Microcontroller in Betrieb nehmen (PlatformIO)
- Einarbeitung in Git / Projekt in Git-Repositories anlegen
- Einarbeitung in Buildumgebung CMake
- Beispielprogramm (led-blink) / Taster einbinden / Architektur mit Application, Treiber und Hardwareschicht
- Einarbeitung in Buildumgebung PlatformIO -> mehrere Build-Projekte
- Google Test auf Host (inclusive Stubbing) / CMakeList erstellen
- Hterm in Betrieb nehmen (Von Hterm an Controller senden und wieder zurück)
- per printf an Hterm senden (printf umleiten)
- Test für Target schreiben (wogtest) und auf Hterm ausgeben
- Doku inclusive UML-Klassendiagramm erstellen

2 Planung

Bei der Planung spielten vor allem Überlegungen zur Strukturierung des Beispielprojekts, für welches beispielhaft Unit-Tests geschrieben werden sollen, eine Rolle.

2.1 Systemarchitektur

Bei der Planung der Systemarchitektur geht es darum, die grundlegende Struktur in Software und das Zusammenspiel aller Komponenten des Systems frühzeitig und durchdacht festzulegen. Das Ziel ist, eine klare, wartbare und erweiterbare Architektur zu schaffen, die sowohl funktionale Anforderungen als auch technische Rahmenbedingungen berücksichtigt. Dafür sollte jede Komponente eine klar definierte Aufgabe haben. Beispielsweise übernimmt die HAL ausschließlich hardwarenahe Funktionen, während die Applikation die Logik steuert. Die Software wird hierfür in Applikation, Treiber und Hardwareabstraktionlayer (HAL) unterteilt. Diese Trennung fördert Wiederverwendbarkeit und Testbarkeit. Die Architektur sollte so gestaltet sein, dass Unit-Tests auf Host und Target möglich sind.

2.1.1 UML-Klassendiagramm

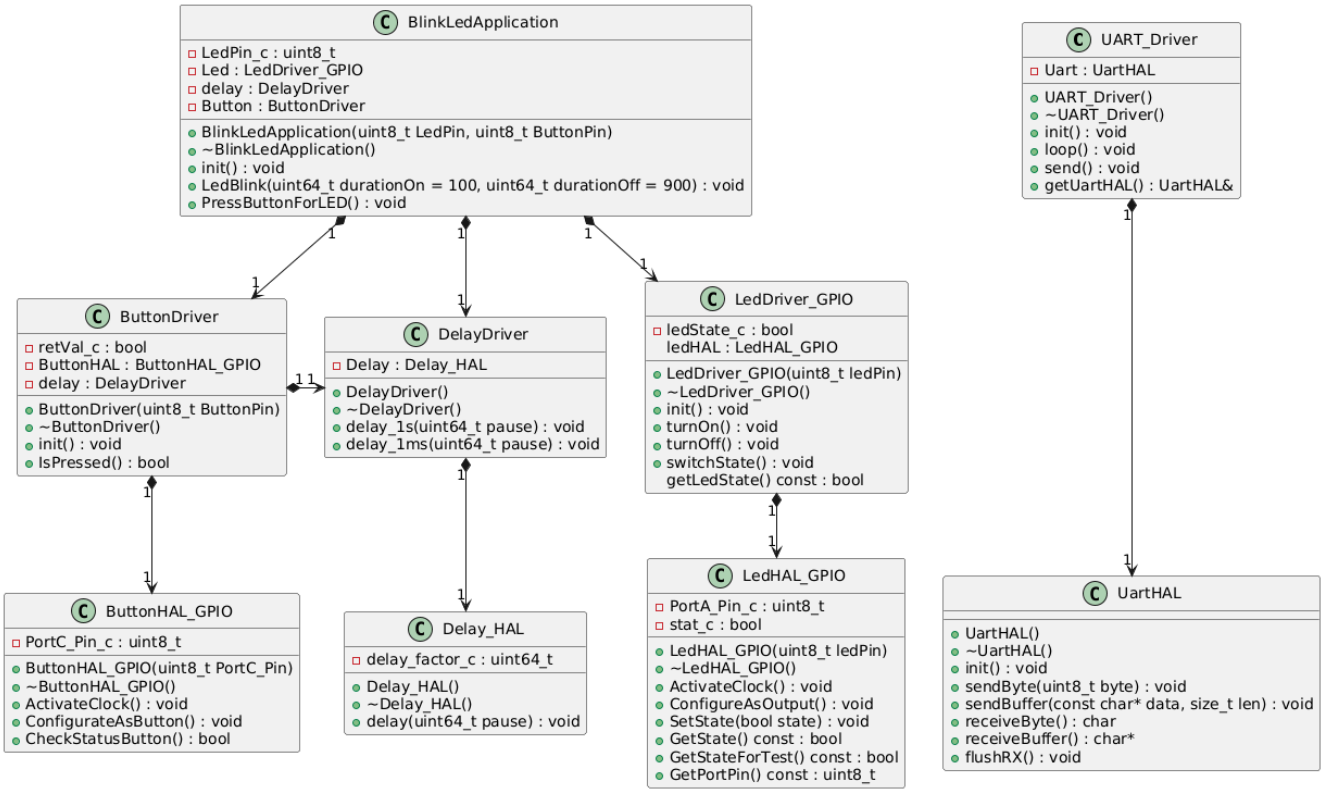

Figure 1: UMLKlassendiagramm

2.1.2 Layer-Modell

3 Projekt

3.1 Code

3.1.1 Builds

die Builds entwickelten sich im Laufe des

4 Testkonzept

4.1 Teststrategie (Host/Target)

4.2 Mocking/Stubbing

4.3 Wogtest-Integration

4.4 Testausgabe

5 Bedienungsanleitung

5.1 Beispielapplikation

5.2 hier LaTeX testen

```
if(ButtonPressed)
{
    ledState = ! ledState;
}
```

Your Abstractö qwfq wbo9qbvf9 quzwvauvbza ibzuvbEFH UZBoac BZBuzbviu eb vububI-UBVDubbvvd Bb bdcub I UBCu bUB UBVI bvuz ubdvsdzbvVUZBVDB BU B b biSDBVZUIB VuzsbdviszudaizsudhI D IUZBi

Das ist fett. *Das ist kursiv.* ***Fett und kursiv!***

5.3

6 Anhang

6.1 Code

6.1.1 Klassen