Implementierung einer Ampelsteuerung

Klausurersatzleistung Embedded Systems in der Automation

im Studiengang Elektrotechnik

an der Dualen Hochschule Baden-Württemberg Mosbach

von

Niklas Stein

20.06.2022

Matrikelnummer	3840855
Ausbildungsfirma	Bosch Rexroth AG, Schweinfurt
Gruppenmitglieder	Cedric Franke, Sven Mößner, Timo Kempf
Dozent:in	Claudia Heß

Ehrenwörtliche Erklärung

Ich versichere hiermit, dass ich meine Klausurersatzleistung mit dem Thema: *Implementierung einer Ampelsteuerung* selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe. Ich versichere zudem, dass die eingereichte elektronische Fassung mit der gedruckten Fassung übereinstimmt.

Ort, Datum

Unterschrift

Inhaltsverzeichnis

1 Einleitung

1.1 Aufgabenstellung

Die Aufgabe dieser Klausurersatzleistung ist es, mit Hilfe eines STM32F401 Nucleo Boards eine Ampelsteuerung in der Programmiersprache C++ zu implementieren. Die Ampel, wie auch ihre Steuerung soll sowohl softwareseitig, als auch hardwareseitig umgesetzt werden. Dem Nutzer wird dadurch ermöglicht die Ein- und Ausgaben der Ampelsteuerung entweder an der Hardware über Taster und LEDs oder rein softwareseitig über die Kommandozeile zu realisieren. Die konkrete Aufgabenstellung mit einzelnen Unterschritten wurde dem Laborskript entnommen.

Im Folgenden werden einige Meilensteine dieser Aufgabenstellung genannt:

- Eine `GPIO`-Klasse, welche auf die `GPIO`-Register des STM32F401 zugreift, wird zur Ansteuerung der LED´s und zum Auslesen der Taster implementiert.
- Um dem Anwender den Umgang mit den `GPIOs` zu erleichtern, werden zwei weitere Klassen, welche auf die `GPIO`-Klasse zugreifen, realisiert.
- Um einen Überblick über das Gesamtsystem zu schaffen, wird ein Zustandsautomat in UML modelliert. Für den gefundenen Automaten wird wiederum ein Klassendiagramm nach dem Entwurfsmuster „State“ modelliert.
- Im folgenden Schritt wird das erstellte Klassendiagramm implementiert.
- In einem Redesign der Ampelsteuerung wird eine Interface-Klasse implementiert, über welche es möglich ist, zwischen dem Betrieb der Ampel auf der Hardware bzw. rein softwareseitigem Betrieb zu wechseln.
- Schließlich werden alle implementierten Komponenten zusammengefügt und ein neues „großes“ Klassendiagramm erstellt. Es werden Funktionstest sowie Codereviews zur Qualitätssicherung durchgeführt.

Die Dokumentation zum gesamten Projekt erfolgt zum einen durch eine Quellcodedokumentation mit Doxygen, zum anderen mit der hier vorliegenden Ausarbeitung.

1.2 Vorgehensweise

Folgende Abbildung 1.1 zeigt das Vorgehen zur Realisierung dieses Projekts.

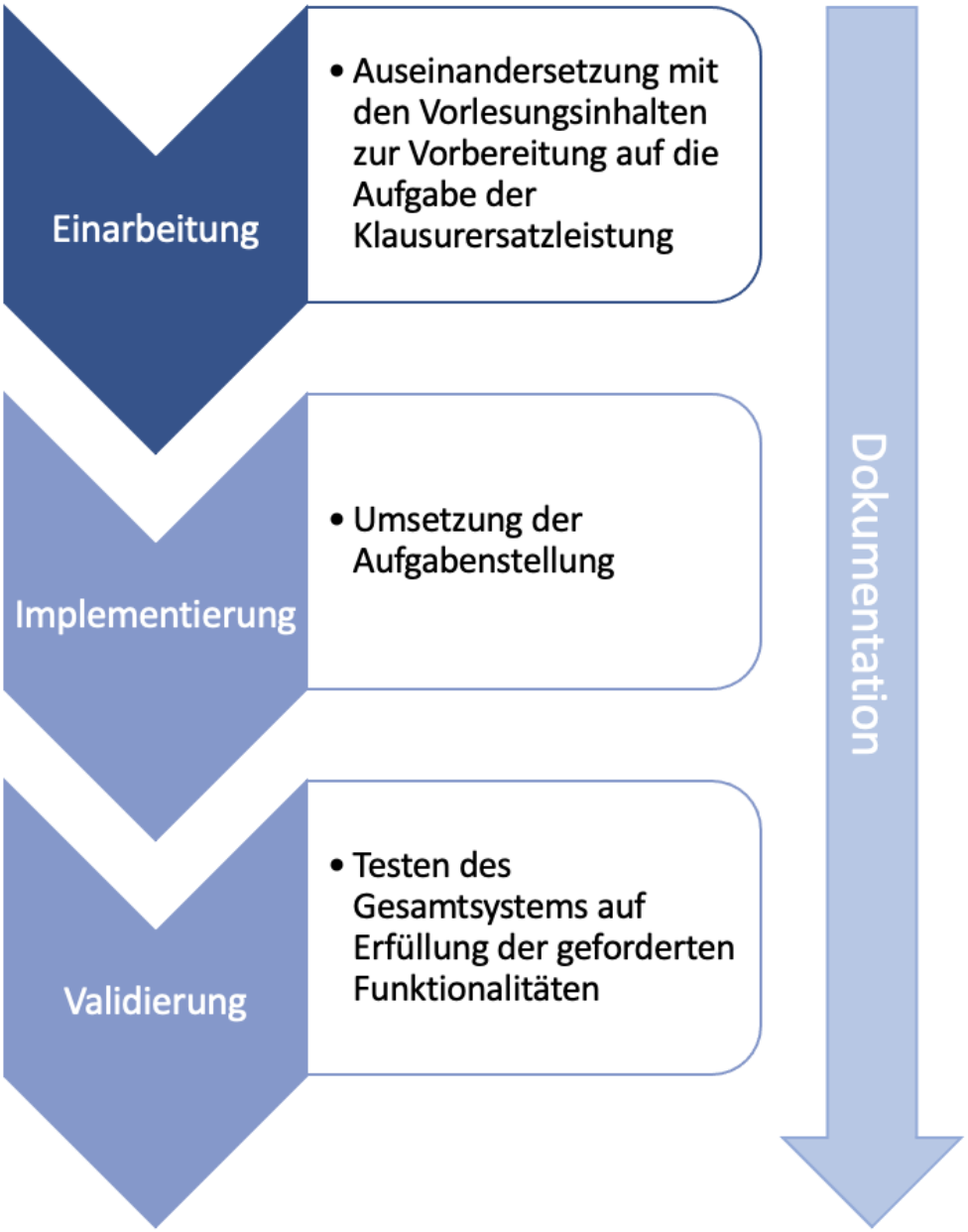

Abbildung 1.1: Vorgehensweise zur Klausurersatzleistung [ED]

2 Hardware

2.1 Allgemeines

Die Ampelsteuerung erfolgt hardwareseitig über ein STM32F401 Nucleo Board. Programme können mit Hilfe der IAR Embedded Workbench auf dieses Board geflasht werden. Die Entwicklungsumgebung stellt eine Hardwareschnittstelle bereit; darüber kann der ST-Link-Debugger und -Programmierer ausgewählt werden.

2.2 Aufbau

Auf das STM32F401 Nucleo Board wird wie in Abbildung 2.1 ersichtlich ein Base Shield aufgesteckt, an welches die Peripherie (LEDs und Taster) angeschlossen wird.

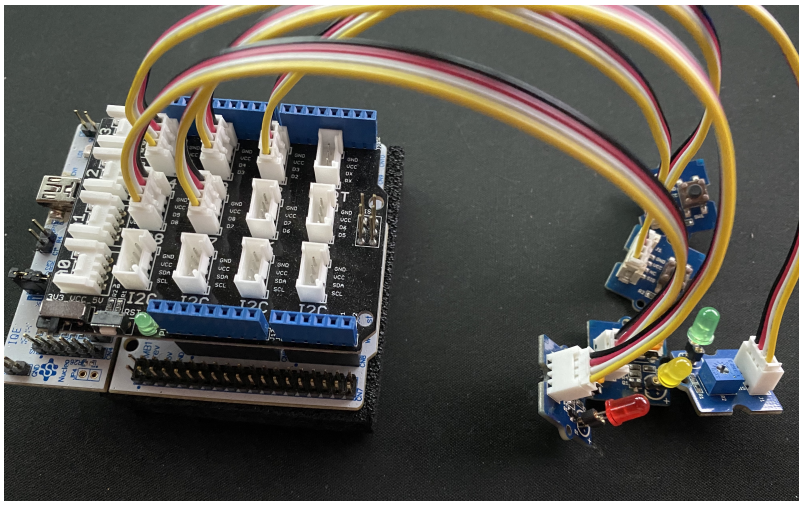

Abbildung 2.1: Hardwareaufbau der Ampelsteuerung [ED]

2.3 Pinbelegung

Die nachstehende Tabelle 2.1 zeigt die hardware- und softwareseitige Pinbelegung der angeschlossenen LEDs und Taster.

Hardware (Port Pin)	Software (Port Pin)	Peripherie
D7	A8	LED Rot
D8	A9	LED Orange
D2	A10	LED Grün
D4	B3	Taster 1
D3	B5	Taster 2

Tabelle 2.1: Pinbelegung (Hardware/Software)

3 Styleguide

3.1 Bezeichungen

3.2 Kommentare

4 Systemarchitektur

4.1 Klassendiagramm des Projekts

4.2 Zustandsautomat

5 State Pattern

5.1 Context, Interface und Concrete Klassen

5.2 Singleton

6 Hardwarezugriff - GPIO

7 Wechsel zwischen Hardware- und Softwarebetrieb

7.1 Definition

Wie bereits erwähnt, soll es dem Anwender möglich sein, die Ampelsteuerung sowohl mit Hardware, als auch komplett ohne Hardware nutzen zu können. Um die beiden Fälle näher zu erläutern, wird der Hardware- und Softwarebetrieb im Folgenden konkret spezifiziert.

- **Hardwarebetrieb:** Der Programmcode der Ampelsteuerung wird auf das Nucleo Board geflasht und auf diesem ausgeführt. Die Nutzereingabe erfolgt über zwei an das Board angeschlossenen Taster. Die Ausgabe der Ampelfarben erfolgt über die drei angeschlossenen LEDs.
- **Softwarebetrieb:** Das Programm der Ampelsteuerung wird auf dem PC ausgeführt. Die Nutzereingabe erfolgt über die Tastatur. Die Tasten „F“ und „B“ repräsentieren die beiden Hardwaretaster. Die Taste „X“ steht für das Betätigen beider Taster gleichzeitig. Das Einlesen der Tastatureingaben erfolgt über den Eingabestream `cin`, die Ausgabe der Ampelfarben über den Ausgabestream `cout` in der Kommandozeile.

7.2 Umsetzung

Ob die Ampelsteuerung im Hardware- oder Softwarebetrieb arbeitet, wird über Pointer bestimmt, welche in der `main.cpp` des Programmes angelegt und bis zur relevanten Stelle im Programmcode durch alle Klassen übergeben werden. Ob die Pointer für den Hardware- oder Softwarebetrieb initialisiert werden, wird wiederum über eine Präprozessoranweisung festgelegt, welche der Nutzer setzen oder auskommentieren kann. Dies ist in Codeauszug 7.1 dargestellt.

```
1  ...
2  #define _HARDWAREPRESENT
3  int main(){
4    #ifdef _HARDWAREPRESENT
5      OutputFormat *myOutputFormat =
           HardwareOutput::GetInstance();
6      InputFormat *myInputFormat =
           HardwareInput::GetInstance();
7    #else
8      OutputFormat *myOutputFormat =
           SoftwareOutput::GetInstance();
9      InputFormat *myInputFormat =
           SoftwareInput::GetInstance();
10   #endif
11 ...
```

Codeauszug 7.1: main.cpp - Hardwarebetrieb oder Softwarebetrieb

Im weiteren Programmcode wurde die Umschaltung zwischen Hardware- und Softwarebetrieb nochmals aufgeteilt. Zum einen wird beim Ausgabeformat der Ampelfarben zwischen Hardware- und Softwareausgabe, zum anderen beim Eingabeformat der Nutzereingabe zwischen Hardware- und Softwareeingabe unterschieden.

7.2.1 Ausgabeformat

Abbildung 7.1 zeigt den Ausschnitt aus dem Klassendiagramm, welcher für die Umschaltung zwischen Hardware- und Softwareausgabe zuständig ist. Die Klasse „OutputFormat“ ist eine Interface Klasse und beinhaltet ausschließlich virtuelle Methoden. Die Unterklassen „SoftwareOutput“ und „HardwareOutput“ erben von der Oberklasse „OutputFormat“ und verwenden deren Interface. Die Concrete Klassen des übergelagerten State Patterns, welche die unterschiedlichen Ampelfarben repräsentieren, besitzen den erläuterten übergebenen Zeiger, welcher in der `main.cpp` initialisiert wurde und auf eine Instanz der Klasse „HardwareOutput“ oder „SoftwareOutput“ zeigt. Je nachdem werden die Methoden von „HardwareOutput“ oder „SoftwareOutput“ aufgerufen, um die Ampelfarben hardware- oder softwareseitig anzuzeigen.

Beim Hardwarebetrieb ruft die Klasse „HardwareOutput“ schließlich die Methoden der Klasse „UserLEDs“ auf und steuert so die LEDs an. Befindet sich die Ampelsteuerung im Softwarebetrieb, so sorgen die Methoden in der Klasse „SoftwareOutput“ für die entsprechenden Ausgabestreams in der Kommandozeile.

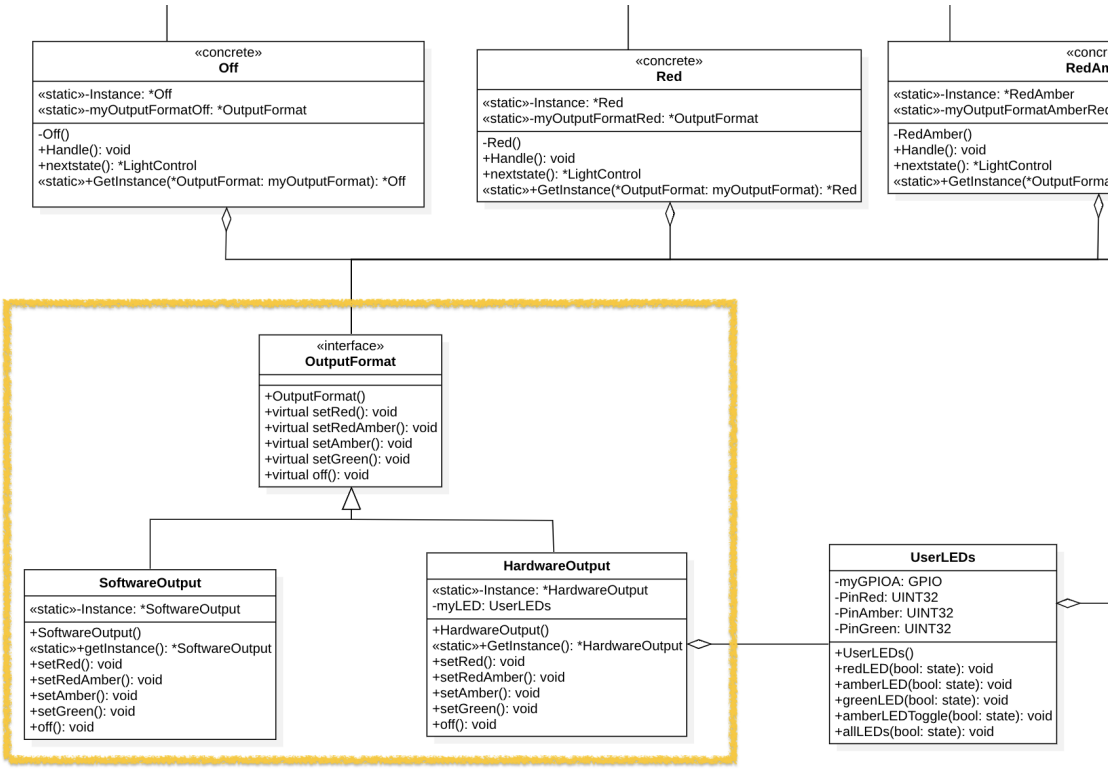

Abbildung 7.1: Auszug aus dem Klassendiagramm - Ausgabeformat [ED]

7.2.2 Eingabeformat

Abbildung 7.2 zeigt einen Auszug aus dem Klassendiagramm, welcher für den Wechsel zwischen Hardware- und Softwareeingabe zuständig ist. Auch hier sind eine Interface Klasse „InputFormat“ und zwei erbende Unterklassen „HardwareInput“ und „SoftwareInput“, welche das Interface verwenden, vorzufinden. Der Wechsel zwischen Hardware- und Softwareeingabe funktioniert dabei identisch zur Logik der Hardware- und Softwareausgabe, weshalb darauf an dieser Stelle nicht weiter eingegangen wird.

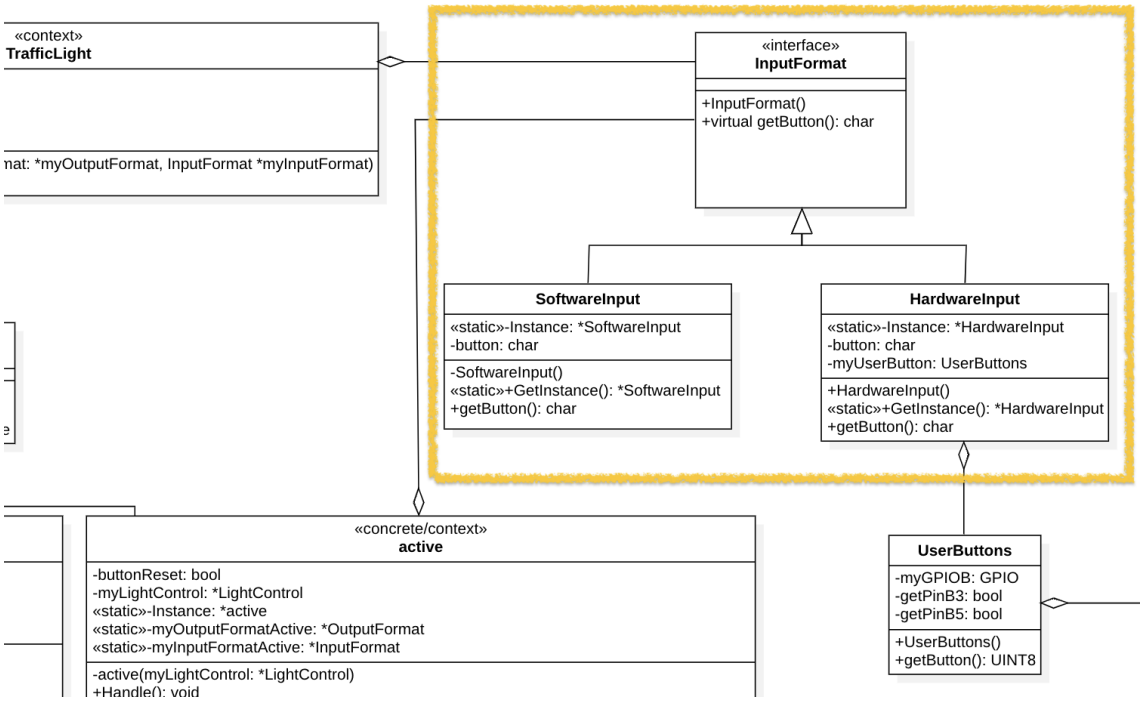

Abbildung 7.2: Auszug aus dem Klassendiagramm - Eingabeformat [ED]

8 Fazit

Abbildungsverzeichnis

Tabellenverzeichnis

Quellcodeverzeichnis

Literatur

[ED] *Eigene Darstellung.*