1. Einleitung

In der Vorlesung „Embedded Systeme in der Automation“ geht es darum zu lernen, wie ein Mikrocontroller objektorientiert, effektiv und mit möglichst wenig Speicherauslastung programmiert werden kann. Dazu wurden in der Vorlesung einige nützliche Methoden übermittelt. Gegen Ende des Semesters wird in diesem Modul jedoch keine Klausur geschrieben. Stattdessen bekommen die Studierenden ein Projekt zur praktischen Anwendung sämtlicher Methoden. Das Ziel dieser Projektarbeit ist es eine Ampelsteuerung zu implementieren.

1.1 Aufgabenstellung

Die in diesem Projekt zu erstellende Ampel soll, wie auch die Ampeln im Straßenverkehr, zwei Betriebszustände haben. Diese sind „active“ und „flashing“. Ersterer Zustand soll eine normale Ampelsteuerung ermöglichen mit den Farbwechseln Rot -> Rotgelb -> Grün -> Gelb -> Rot. Der zweite Zustand soll signalisieren, dass die Ampel nicht in Betrieb ist. Dies soll dadurch erreicht werden, dass das gelbe Licht blinken soll. Weiterhin soll es für den Benutzer möglich sein die Schnittstellen zur Ampelsteuerung selbst zu wählen. Hier soll es zwei Möglichkeiten geben. Erstere soll die Hardwareschnittstelle mit zwei Buttons und drei LEDs mit den Farben rot, gelb und grün sein. Andernfalls soll es auch möglich sein die Ampel über das Terminal zu steuern. Dies wäre die Softwareschnittstelle.

1.2 Vorgehensweise

Die Vorgehensweise bei diesem Projekt orientiert sich an einem Leitfaden, welcher von der Dozentin Claudia Hess stammt. Zunächst werden die bereitgestellte Hardware analysiert und erste Vereinbarungen bezüglich der Pinbelegungen getroffen. Danach wird ein Styleguide angelegt. Dieser ist nötig um einen möglichst einheitlichen Code zu erhalten. Im nächsten Schritt wird dann eine Architektur entworfen um alle geforderten Funktionen zu erfüllen. Daraufhin werden verwendete State Patterns vorgestellt und erklärt. Zuletzt wird dann noch auf den Hardwarezugriff eingegangen, bevor gezeigt wird, wie der Wechsel zwischen Hardware- und Softwareschnittstelle erfolgt. Zum Schluss wird dann ein Fazit gezogen.

2. Hardware

Zunächst wird die bereitgestellte Hardware analysiert. Diese wurde von der Hochschule bereitgestellt.

2.1 Allgemeines

Verwendet wird das ‚STM32F401RE Nucleo Board‘ der Firma ‚STMicroelectronics‘. Weiterhin wird ein Shield verwendet, welches auf den Microcontroller aufgesteckt werden kann. So wird eine Verbindung zwischen einigen GPIOs des Boards und dem Shield ermöglicht. Des Weiteren werden LEDs und Buttons mitgeliefert. Zuletzt wird noch ein USB-Kabel zur Verbindung zwischen Laptop und Mikrocontroller benötigt. Dieses versorgt den Controller mit Spannung und ermöglicht eine serielle Kommunikation, welche später benötigt wird.

2.2 Aufbau

Im Folgenden wird nun auf den Aufbau der Hardware eingegangen. Das Shield wird auf das Board aufgesteckt. Die mitgelieferten LEDs und Buttons können nun einfach am Shield eingesteckt werden und erhalten damit eine leitende Verbindung zu Versorgungsspannung, Masse und einer Signalleitung. Die Verbindung über das Shield bietet eine höhere Festigkeit und Stabilität gegenüber einzelnen Jumperverbindungen, welche direkt am Board gemacht werden könnten.

2.3 Pinbelegungen

Da der Arbeitsaufwand dieses Projektes auf vier Studierende aufgeteilt wird, muss abgestimmt werden, welcher Pin jeweils den LEDs und den Buttons zugeordnet werden soll. Deshalb wird gleich zu Beginn des Projektes eine Vereinbarung der Pinbelegungen getroffen. Diese sind in Tabelle 1 zu sehen.

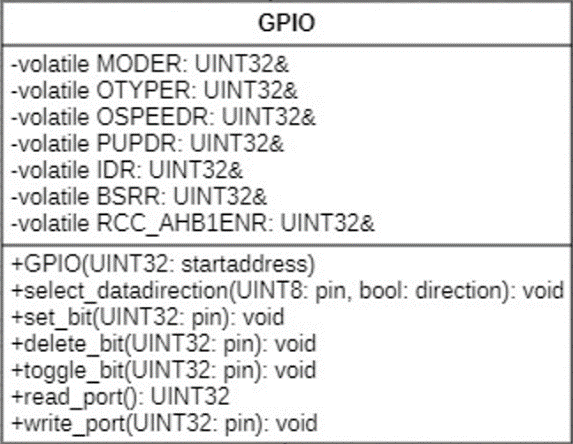
Tabelle 1: Bauteile und ihre dazugehörigen Pinbelegungen

|  |  |  |
| --- | --- | --- |
| Bauteil | Portzughörigkeit und Pinnummer | Steckbezeichnung Shield |
| LED rot | PA8 | D7 |
| LED gelb | PA9 | D8 |
| LED grün | PA10 | D2 |
| Button 1 | PB3 | D3 |
| Button 2 | PB5 | D4 |

6. Hardwarezugriff – GPIO

Der Hardwarezugriff erfolgt über drei Klassen. Die untergeordnete Klasse ‚GPIO‘ greift dabei direkt auf die Register des Mikrocontrollers zu und steuert somit sämtliche Hardwarezugriffe. Die zwei übergeordneten Klassen ‚UserLEDs‘ und ‚UserButtons‘ greifen dann auf die Klasse ‚GPIO‘ zu und steuern diese. Übergeordnet heißt hier jedoch nicht, dass es sich um Oberklassen einer Vererbung handelt. Die hier herrschende Beziehung ist eine Aggregation. Außer zu diesen beiden Klassen hat die ‚GPIO‘-Klasse keine weiteren Verbindungen. Eine Hardwareansteuerung kann somit nur durch die beiden genannten übergeordneten Klassen durchgeführt werden.

Nun wird etwas genauer auf die die ‚GPIO‘-Klasse eingegangen. Sie ist wie folgt aufgebaut:



Alle Attribute dieser Klasse sind als „private“ gekennzeichnet. Das liegt daran, dass es sich bei den Attributen um Zeiger handelt, mit welchen man auf die GPIO-Register des Mikrocontrollers zugreifen kann. Hier wird das Prinzip des Information Hidings angewendet, um zu verhindern, dass die Register von außen manipuliert werden könnten. Weiterhin wird den Attributen das Schlüsselwort volatile vorangestellt. Dieses gibt dem Compiler die Information, dass sich der Inhalt auf den die Zeiger zeigen, ändern kann, auch wenn das Programm nicht schreibend darauf zugreift.

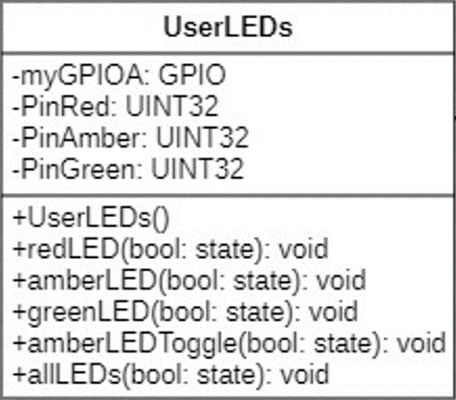
Der Konstruktor der Klasse konfiguriert einen GPIO-Port. Ihm muss lediglich die Startadresse des gewünschten Ports übergeben werden. Diese Startadresse kann dem Datenblatt auf Seite 53 entnommen werden. Auf Basis der Startadresse adressiert dieser dann die benötigten Register. Weiterhin wird dort das Clock Signal des ausgewählten Ports aktiviert.

Über die Methode ‚select\_datadirection()‘ kann dann für jeden Pin des Ports einzeln ausgewählt werden ob, dieser als Input oder als Output fungieren soll. Dazu muss der Methode nur die Pinnummer und die Art des GPIOs übergeben werden.

Mit ‚set\_bit()‘ kann ein ausgewählter Output gesetzt werden. Dazu muss lediglich die Pinnummer bekannt sein. ‚delete\_bit()‘ macht genau das Gegenteil. Dort wird ein Output zurückgesetzt. ‚toggle\_bit()‘ vereint beide Methoden. Dort wird der Status des Outputs abgefragt und danach umgekehrt.

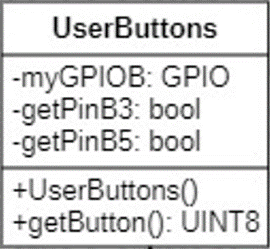
Mit ‚read\_port()‘ werden die Zustände jedes einzelnen Pins des gesamten Ports gelesen und als 32-Bit Wert zurückgeben. Bei der Ausführung von ‚write\_port()‘ hingegen, wird der gesamte Port beschrieben. Wirksam ist der Schreibzugriff jedoch nur auf definierte Outputs.

Nun wird auf die Klasse „UserLEDs“ betrachtet. Das dazugehörige Klassendiagramm kann folgender Abbildung entnommen werden:



Auch hier sind alle Attribute als private definiert, um das Information Hiding zu waren. Das erste Attribut ist eine Instanz der ‚GPIO‘-Klasse. Es dient als Schnittstelle zur Ansteuerung der Pins, an welchen die LEDs angeschlossen sind. In den restlichen drei Attributen werden die jeweiligen Pinnummern der LEDs hinterlegt. Dies erfolgt im Konstruktor über eine Initialisierungsliste. Danach wird dort noch, über die ‚select\_datadirection()‘-Methode der ‚GPIO‘-Klasse festgelegt, dass es sich bei den vorliegenden Pins um Outputs handelt. Mit den restlichen Methoden werden dann die LEDs angesteuert. Ihnen muss lediglich der gewünschte Zustand übergeben werden.

Zuletzt wird die Klasse ‚UserButtons‘ betrachtet. Auch hier wird ein UML Klassendiagramm zur Erläuterung herbeigezogen:



Dort wird auch wieder eine private Instanz der Klasse ‚GPIO‘ erstellt. Damit können die Buttons abgefragt werden. Weiterhin existieren zwei private Methoden, welche den Status des jeweiligen Buttons zurückgeben. Sie werden in der Methode ‚getButton()‘ aufgerufen. Dort erfolgt eine Entprellung. Dazu wird der Wert zunächst ein erstes Mal abgefragt. Danach wird 3000 Schleifendurchgänge gewartet und der Status erneut abgefragt. Liefern beide Abfragen das gleiche Ergebnis, so wird der Wert übernommen. Andernfalls wird die Entprellung erneut durchgeführt. Wie im Klassendiagramm zu sehen hat die Methode einen Rückgabewert vom Datentyp ‚UINT8‘, dies entspricht einem 8 Bit Charakter. Je nachdem, welcher Button nun gedrückt wird, wird ein bestimmter Rückgabewert zurückgegeben. Eine Auflistung aller möglichen Rückgabewerte kann folgender Tabelle entnommen werden:

Tabelle : Rückgabewerte der Methode 'getButton()' der Klasse 'UserButtons'

|  |  |
| --- | --- |
| Welche Buttons gedrückt? | Rückgabewert |
| Keiner | O |
| Nur Button an PB3 (D3) | B |
| Nur Button an PB5 (D4) | F |
| Beide Buttons | X |

8. Fazit

Schlusslegend kann gesagt werden, dass fast alle Ziele dieses Projektes erreicht wurden. Die erstellte Ampel kann, sowohl rein Software- als auch rein Hardwareseitig betrieben werden. Weiterhin ist es möglich den Betriebszustand der Ampel zu wechseln. Ist die Ampel in Betrieb, so wird in der korrekten Farbfolge geschalten, ist sie dagegen außer Betrieb, sollte die gelbe LED blinken. Dieses Ziel konnte leider nicht erreicht werden, da keine vernünftige Methodik gefunden wurde, ein delay einzubauen.