25.03.2025

Vario

Microcomputertechnik PStA von Bastian Herold und Leonard Rother

Inhaltsverzeichnis

[1. Aufgabenstellung 2](#_Toc193812316)

[2. Informationsbeschaffung 2](#_Toc193812317)

[3. Schaltplan 2](#_Toc193812318)

[4. Programmablaufplan 2](#_Toc193812319)

[5. Programmierung 3](#_Toc193812320)

[5.1. Cube MX 3](#_Toc193812321)

[5.2. Code 3](#_Toc193812322)

[5.2.1. LED blinken 3](#_Toc193812323)

[5.2.2. Analoge Spannung einlesen 3](#_Toc193812324)

[5.2.3. I2C Luftdrucksensor 4](#_Toc193812325)

[5.2.4. Buzzer 4](#_Toc193812326)

[5.2.5. Display 4](#_Toc193812327)

[6. Versionsverwaltung via Github 4](#_Toc193812328)

[7. Lessions learned 5](#_Toc193812329)

[8. x 5](#_Toc193812330)

# Aufgabenstellung

# Informationsbeschaffung

Datenblätter und sonstiges heraussuchen...

Microcontroller: STM32L476 Nucleo – NUCLEO-L476RG

Display: I2C 1602 LCD  
2 Zeilen mit je 16 Zeichen

Buzzer (Standard Aruduino)

Barometer – Aus letztem Semester E-Praxis: Bosch BMP280, i2c  
Hier zwar das Datenblatt von Adafuit und deren Sensormodul, allerings finde ich dieses viel besser und genauer beschrieben als das von Bosch selbst.

# Schaltplan

TODO: Leonard

# Programmablaufplan

TODO: Leonard

# Programmierung

Debuggen  
ST Link auf dem Nucleo erstmal updaten -> nur mit dem richtigen Kabel!

How to: printf  
<https://www.embedded-communication.com/en/misc/printf-with-st-link/>

## Cube MX

Die Pinbelegung wurde mithilfe von dem ST eigenen Tool Cube-MX gemacht, dort ist es sehr einfach und schön grafisch dargestellt welche Funktion ein bestimmter Pin belegen kann. Die jeweilige Funktion einem Pin zuweisen, geht mit einem Mausklick. So kann man sich sicher sein, dass keine Fehler bei der Pinbelegung gemacht werden, somit keine Überraschungen später bei der Programmierung.

## Code

Als Entwicklungsumgebung nutzen wir das von ST eigene Tool CubeIDE.

Eigene Setting.h schreiben, in die alle groben Einstellungen kommen, die später im Programm genutzt werden.

Um die anderen Funktionen nicht durch ein Delay zu verzögern, oder anders herum sich nicht von anderen Delays verzögern lassen, konnte kein Delay verwendet werden.  
Deswegen wurde die Methode gewählt, die aktuelle Systemzeit abzufragen und die jeweilige Funktion erst dann ausführen, wenn die Systemzeit um die gewünschte „Delay“ Zeit vorangeschritten ist. Ist die Systemzeit noch bevor der nächsten geplaten Ausführung, wird dieser Codeteil einfach übersprungen.

### LED blinken

Hier habe ich noch keine Hilfe benötigt, das habe ich aus eigenem Wissen gemacht.   
Sehr einfach, da nur ein Wert auf ein GPIO geschrieben werden muss und der Wert wird jedes mal invertiert.

Später habe ich noch eine Funktion hinzugefügt die Frequenz des Blinkens über die analoge Spannung zu steuern.

### Analoge Spannung einlesen

Da ich selbst ein begeisteter elektronik entwickler bin und in meiner Freizeit viele kleine Projekte gemacht habe, konnte ich diese Teilaufgabe aus einem alten Projekt von mir kopieren.

Anzupassen war das ADC Element, sowie die Parameter, welche in die Setting.h ausgelagert wurden.

Da das eigentliche Akkumodul keinen direkten Anschluss für das Nucleoboard hat, habe ich zum Testen ein Potetniometer angesteckt.

### I2C Luftdrucksensor

Zuerst nach einem I2C Scanner gegoogelt und diesen laufen lassen.   
Dieser hat auch sofot bei 0x76 einen Treffer gelandet. Somit wusste ich, dass die grundsätzliche Komminikation funktioniert und der Sensor antwortet.

Hier habe ich im Internet nach einer passenden Library gesucht und bin auch sehr schnell fündig geworden:

Allerdings gab es viele Compiler-errors was es erstmal zu lösen gab.   
Ebenfalls musste die i2c Library aus dem gleichen Git-Repository heruntergeladen werden.

Diese war allerdings wieder von sehr vielen anderen Librarys von „Ihm“ abhängig, weshalb ich mir den Sourcecode doch genauer angeschaut habe.   
Dort fand ich eine i2c\_init Funktion, welche diese vielen Abhängigkeiten benötigt. Diese konnte ich einfach löschen, da der i2c Bus bereits in unserer main.c initialisiert wird.

Noch die richtige I2C Addresse ausgewählt und der Sensor konnte angesprochen werden.

### Buzzer

Beim Vario ist es so, dass sich mit mehr vertikaler Geschwindigkeit der Ton auch in seiner Frequenz steigt. Gleichzeitig ist noch ein niederfeqentes signal aufmoduliert, welches den eigentlichen Ton 1 bis 5 mal pro sekunde „piepen“ lässt.

Hierfür wurde im Cube MX bereits der Timer 2 aktiviert und die PWM Generation für Channel 3, also den passenen GPIO aktiviert.

Um später den Timer mit einer updaterate von 1MHz benutzen muss der 80MHz takt mit einem Prescaler von 80 heruntergeteilt werden.   
Die restlichen Werte Zur PWM Generation wie auto-reload Wert und output-compare Wert, werden im code zur laufzeit verändert.

Ebenfalls wurde im Cube MX der Timer 6 aktiviert und die Interrupt-Funktion eingeschalten.  
Hier wird eine Updatefrequenz von 40kHz benötigt, darum in Prescaler von 2000.  
Wie auch schon bei TIM2 werden auch hier AR und OC zur der Laufzeit gesetzt.

### Display

TODO: Leonard

# Versionsverwaltung via Github

Um auch später, nach dem Studium weiterhin mit dem Projekt weitermachen zu können und auch das „eigentliche“ Git zu benutzen, haben wir uns entschieden dieses Projekt nicht mit Gitlab, sondern mit Github zu machen.

Wir haben das neue Repository eingerichtet und beide auf unsere Rechner geclont.

Anfangs wollten wir uns noch absprechen, wer wann arbeitet, allerdings haben wir das dann doch nicht gemacht.   
Trotzdem kam es nie zu „Merge“ problemen und das Arbeiten im GIT hat sehr gut funktioniert.

Mit der Zeit kam auch noch ein „.gitignore“ File hinzu, welches das unnötige hochladen von den Built-Daten weglässt.   
Damit der jeweils Andere immer weiß was zuletzt gemacht wurde, haben wir zu jedem „commit“ eine Nachricht geschrieben. Diese taucht im GIT neben dem Filenamen auf.

Alles was zu viel für eine einfache commit Nachricht ist, kam in die README des jeweiligen Ordners.  
Hier steht z.B. der Link zur printf Anleitung / der Pfad zu meinem lokalen Projektverzeichnis, um mit der commandozeile leichter hinzukommen.

# Lessions learned

# x