

Auftraggeber: gibb

Betreuer: Burri Matthias

Autor: David Jäggli, Nils Jäggi

Datum: 11.11.2020

Der Auftrag dieses Projektes ist es einen Fidget-Spinner zu realisieren, der mit dem PoV-Prinzip (Persistance of Vision) einen Text/Symbole anzeigen kann.

BüP

Hot-Board

Probelauf IPA

Inhalt

[1 Einleitung 1](#_Toc56254481)

[2 Hauptteil 2](#_Toc56254482)

[2.1 Blockschaltbild Hardware 2](#_Toc56254483)

[2.2 Schema 2](#_Toc56254484)

[2.2.1 Taster 2](#_Toc56254485)

[2.2.2 USB-Interface 2](#_Toc56254486)

[2.2.3 Spannungsregler 3](#_Toc56254487)

[2.2.4 Mikrocontroller 3](#_Toc56254488)

[2.3 Software 5](#_Toc56254489)

[2.3.1 Hauptsoftware 5](#_Toc56254490)

[3 Schlussteil 6](#_Toc56254491)

[3.1 Fazit 6](#_Toc56254492)

[3.2 Probleme 6](#_Toc56254493)

[3.3 Vergleich mit Aufgabenstellung 6](#_Toc56254494)

[3.4 Quellenverzeichnis 7](#_Toc56254495)

Anhang

[A Anhang 11](#_Toc528075874)

[A.1 Gesamtschema 11](#_Toc528075875)

[A.2 Blockschaltbild 12](#_Toc528075876)

[A.3 Pflichtenheft 13](#_Toc528075877)

[A.4 Bestückungsplan und Stückliste 14](#_Toc528075878)

[A.5 Bedienungsanleitung 16](#_Toc528075879)

[A.6 Zustandsdiagramm Hauptsoftware 17](#_Toc528075881)

[A.7 Zeitplan 18](#_Toc528075882)

# Einleitung

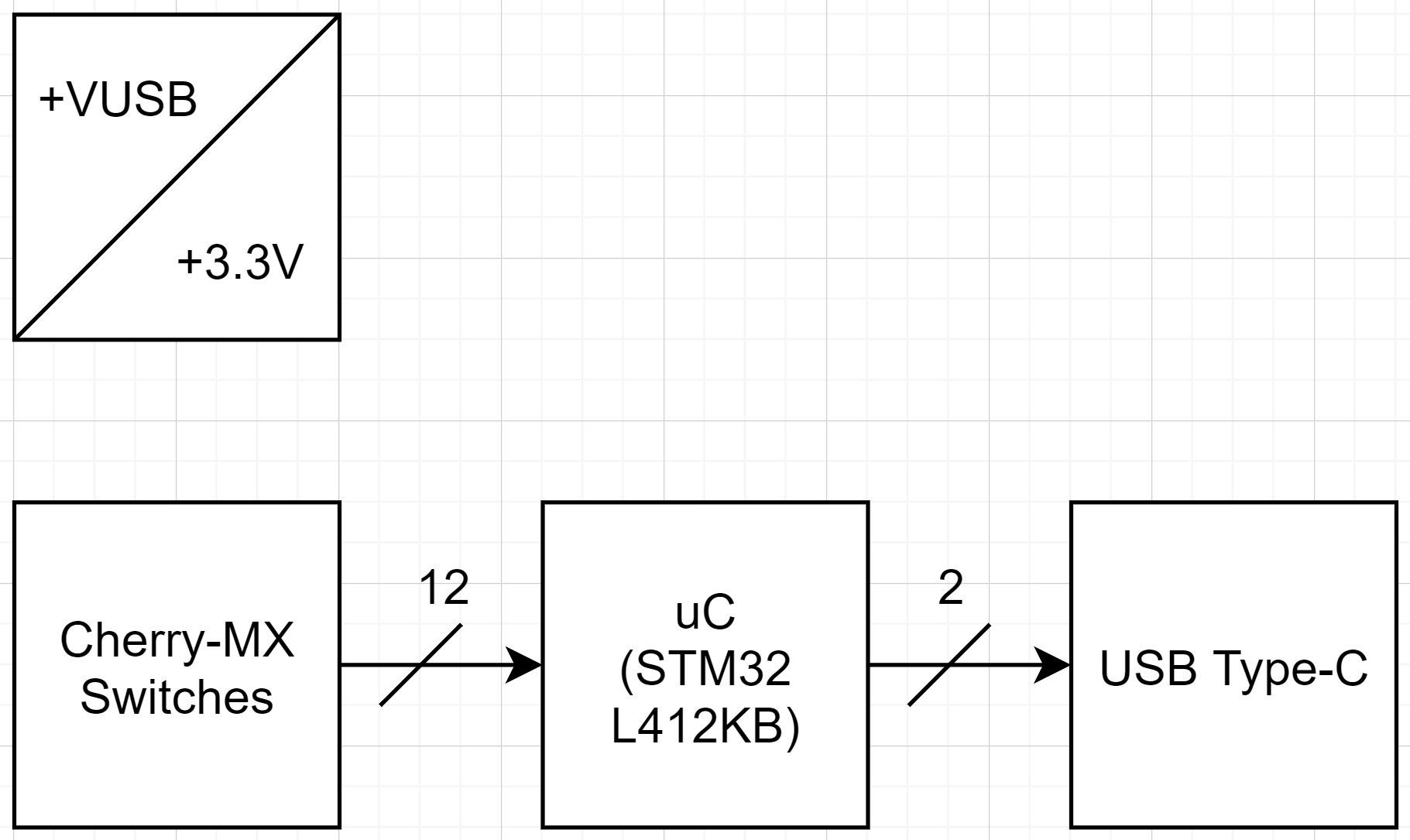
**Open-Source**

Das gesamte Projekt mit allen benötigten Dateien ist zur freien Benützung öffentlich erhältlich auf Github

# Hauptteil

## Blockschaltbild Hardware

Die Hardware beinhaltet die 12 Taster und einen Mikrocontroller, welcher die Tastenanschläge in einen von der Software interpretierbaren Command umwandelt. Dafür wurde ein STM32L412KB verwendet. Dieser bringt bereits ein Protokoll zur Kommunikation über Virtual-COM-Port mit sich.



## Schema

### Taster

Als Taster werden Mechanische von der Marke «Cherry-MX» verwendet. Diese sind auf dem Tastaturen-Markt weit verbreitet und werden oft eingesetzt. In diesem Fall werden sie direkt an einen GPIO des Mikrocontrollers angeschlossen und per Pull-Down abgeschlossen, welcher für einen Definierten Zustand sorgt.

### USB-Interface

Die Ausgewerteten Daten werden vom Mikrocontroller über die Serielle Schnittstelle an einen Computer gesendet. Als Anschluss wurde eine USB TYPE-C Buchse verwendet.

IC3 dient als Schutz gegen Überspannung auf den Datenleitungen.

### Spannungsregler

Mithilfe eines LDO-Spannungsreglers werden die 5V der USB-Schnittstelle auf 3.3V geregelt. Diese tiefere Spannung ist nötig für den Mikrocontroller, da dieser mit 3.3V arbeitet

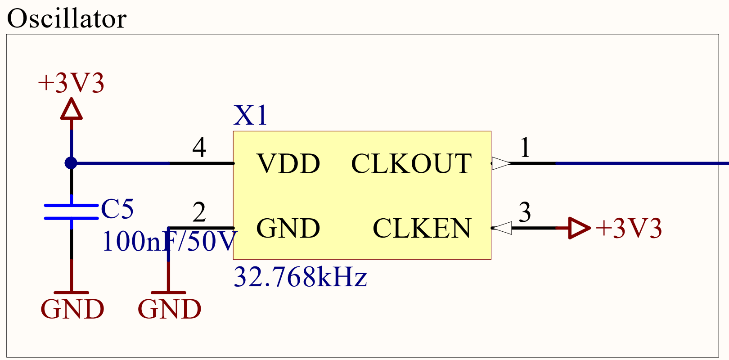
### Mikrocontroller

Bei der Auswahl des Controllers wurden die folgenden Mindestanforderungen gesetzt:

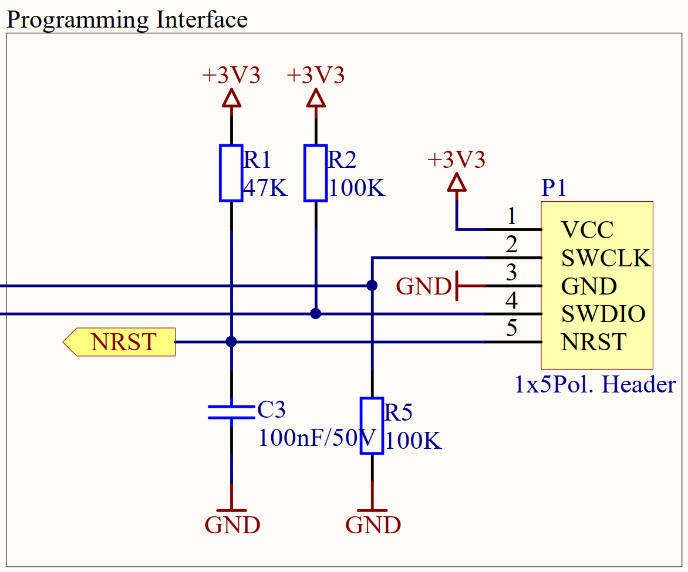
* Integrierte USB-Schnittstelle
* STM32L-Familie (Low-Power Familie)
* Mindestens 12 GPIO’s
* Handlötbarer Formfaktor

Das gewählte Modell ist der STM32L412KBT6. Dieser bietet genügend Pins und eine USB-Schnittstelle. Das Gehäuse ist ein LQFP mit 32 Pins.

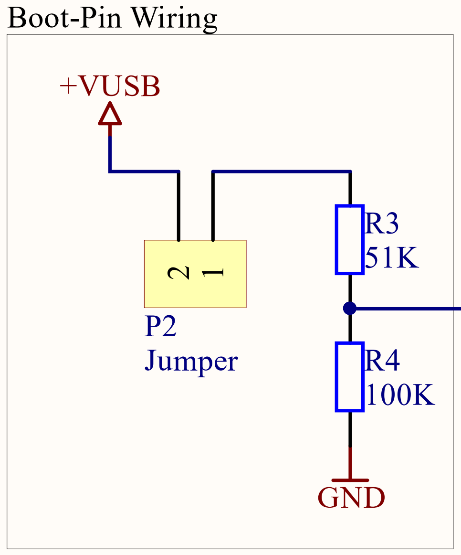
Die Abbildung stellt die Beschaltung des IC’s dar. Die drei Kondensatoren dienen als Abblockkondensatoren an der Speisung.



Getaktet wird der Controller durch einen internen Oszillator, genannt MSI. Dieser versorgt die interne Peripherie mit 48MHz. Die Genauigkeit dieser Takt ist jedoch ungenau und durch Temperatur sehr schwankend. Deshalb ist extern ein Taktgenerator (X1) angeschlossen, welcher den Controller mit einem 32.768kHz Signal speist. Mithilfe dieses Taktes wird danach der interne 48MHz Clock kalibriert. Die Genauigkeit ist wichtig, da die USB-Schnittstelle einen möglichst genauen Takt benötigt, um eine Fehlerfreie Datenübertragung zu gewähren.

Um den Mikrocontroller zu Programmieren oder zu Debuggen steht die SWD Schnittstelle zur Verfügung.

Sie beinhaltet zudem die Reset-Beschaltung (R1/C3), welcher auf VCC verbunden sein muss.

Der «BOOT0» Pin des Controllers bestimmt, wie er aufstartet. Standardmässig ist dieser mit einem Pull-Down Widerstand (R4) auf Masse gezogen, sodass er normal aufstartet und das Programm ausführt. Alternativ kann der Jumper P2 geschlossen werden und beim Startup wird dadurch der Bootloader gestartet. Dies erlaubt es, den Controller per USB-Schnittstelle zu Programmieren. R3/R4 dient als Spannungsteiler, da hier direkt die Spannung der USB-Schnittstelle verwendet wird.

## Software

### Hauptsoftware

# Schlussteil

## Fazit

## Probleme

## Vergleich mit Aufgabenstellung

## Quellenverzeichnis

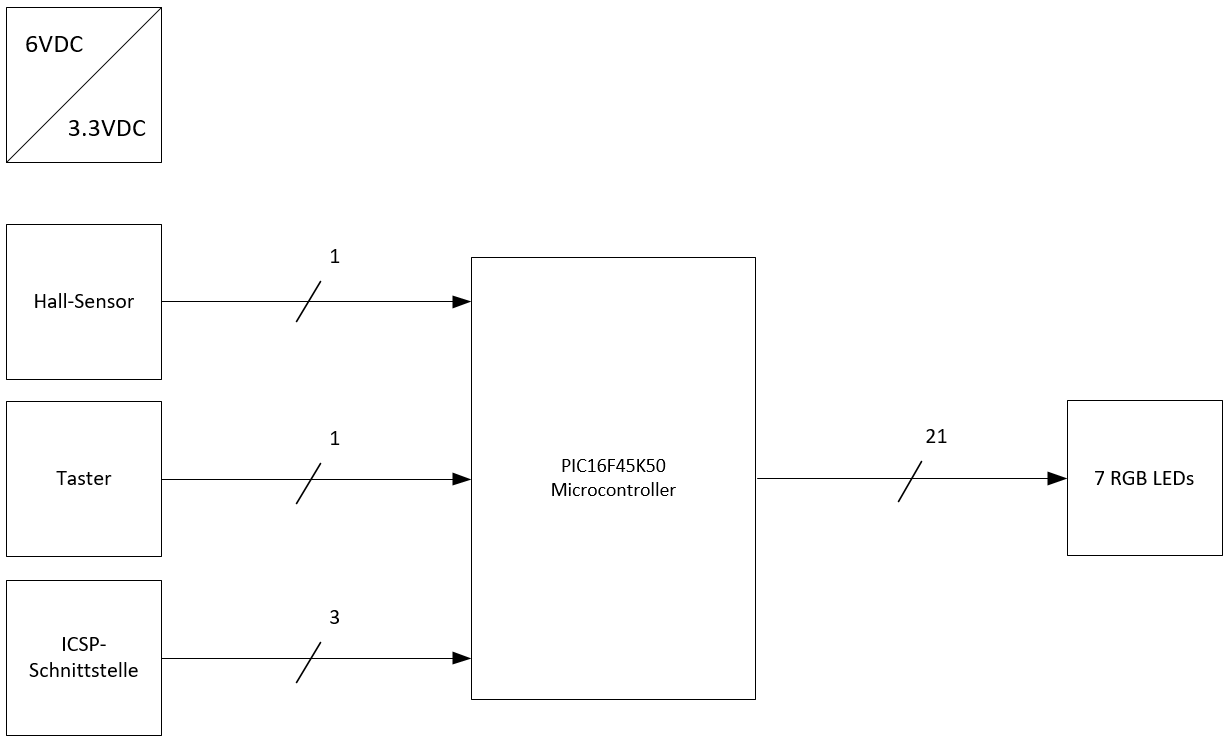
Microchip: PIC18(L)F2X/45K50, Microchip, 27.07.2018, <http://ww1.microchip.com/downloads/en/DeviceDoc/PIC18F2X_45K50-30000684B.pdf>

Würth Elektronik: WL-SFTW SMD TOP LED full-color waterclear, Würth Elektronik eiSos GmbH & Co. KG, 27.7.2018,<http://www.farnell.com/datasheets/1911491.pdf?_ga=2.146832631.1915304690.1540276737-1213338847.1534140400>

Infineon: TLE4946-2K, infenion, 10.10.2018, <http://www.farnell.com/datasheets/1932173.pdf?_ga=2.190443627.1915304690.1540276737-1213338847.1534140400>

Texas Instruments: TPS6220x High-Efficiency, Texas Instruments, 27.10.2018, <http://www.ti.com/lit/ds/symlink/tps62203.pdf>

1. Anhang
   1. Gesamtschema
   2. Blockschaltbild



* 1. Pflichtenheft

**Beschreibung**

Ziel des Projekts ist es einen Fidget-Spinner zu realisieren, der mit dem PoV (Persistance of Vision) - Prinzip einen Text anzeigen kann. Dazu ist es nötig LEDs auf einem selbst konstruierten Fidget-Spinner zu installieren und diese im richtigen Moment anzusteuern. Mit der Trägheit des Auges «sehen» wir dann einen Text. Der angezeigte Text und die Farbe sollen jederzeit über die Serielle, Virtual COM-Port Schnittstelle verändert werden können. Dazu soll, wenn möglich, eine eigene Windows Applikation geschrieben werden. Ansonsten wird der Fidget Spinner mit dem Programm TeraTerm konfiguriert.

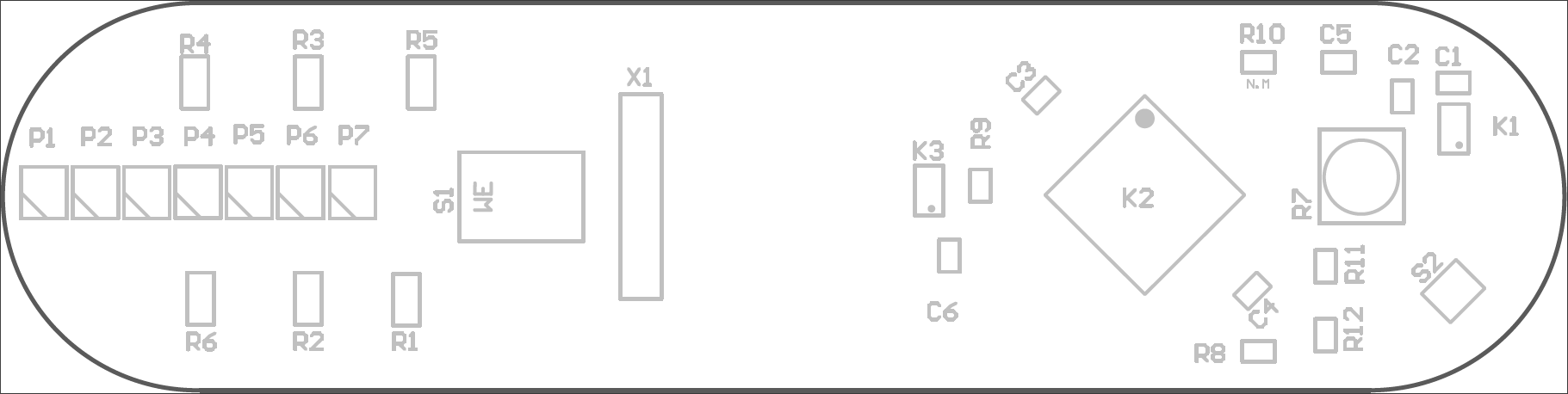
**Obligatorische Ziele**

* Einen Text anzeigen lassen (min. 15 Zeichen)
* Den Text verändern können (Tera Term)
* RGB LEDs für die Textanzeige verwenden
* Farben einstellbar (7 Farben)
* PIC-Mikrocontroller mit USB verwenden
* Gute Gewichtsverteilung auf den beiden Flügeln des Fidget-Spinners
* Speisung über Batterien
* Schrift bleibt kontinuierlich an derselben Stelle

**Optionale Ziele**

* Farben dimmbar (volles Farbspektrum)
* Eigene Windows-Software (C#) zum definieren des Textes bzw. der Farbe schreiben
  1. Bestückungsplan und Stückliste

**Oberseite**



**Stückliste Oberseite**

|  |  |  |  |
| --- | --- | --- | --- |
| Menge | Position | Bezeichnung | Händler |
| 1 | C1 | Cap, Cer, 4,7uF, 0805 | Distrelec |
| 1 | C2 | Cap, Cer, 10uF, 0805 | Distrelec |
| 3 | C3, C4, C6 | Cap, Cer, 100nF, 0805 | Distrelec |
| 1 | C5 | Cap, Cer, 470nF, 0805 | Distrelec |
| 1 | K1 | Buck-Converter, TPS62203, SOT23-5 | Distrelec |
| 1 | K2 | Microcontroller, PIC18F45K50, TQFP44 | Farnell |
| 1 | K3 | Hall-Effekt-Sensor, TLE4946-2K, SOT23-3 | Farnell |
| 7 | P1 – P7 | RGB-LED, 3528 | Farnell |
| 2 | R1, R2 | Widerstandsnetzwerk, 220R, 0612 | Distrelec |
| 2 | R3, R4 | Widerstandsnetzwerk, 330R, 0612 | Distrelec |
| 2 | R5, R6 | Widerstandsnetzwerk, 560R, 0612 | Farnell |
| 1 | R7 | Spule, 10uH | Farnell |
| 2 | R8, R9 | Res, 10k, 0805 | Distrelec |
| 1 | R11 | Res, 4,7k, 0805 | Distrelec |
| 1 | R12 | Res, 6,8k, 0805 | Distrelec |
| 1 | S1 | Switch on-on, SW DPDT | Distrelec |
| 1 | S2 | Taster, B3U | Farnell |
| 1 | X1 | ICSP-Programmierschnittstelle, 5-Pin Stiftleiste | Distrelec |
| 1 | X2 | Micro-USB Buchse, USB AB | Distrelec |

**Unterseite**



**Stückliste Unterseite**

|  |  |  |  |
| --- | --- | --- | --- |
| Menge | Position | Bezeichnung | Händler |
| 2 | G1, G2 | 16mm Knopfbatterie 3V | Farnell |
| 1 | X2 | Micro-USB Buchse, USB AB | Distrelec |

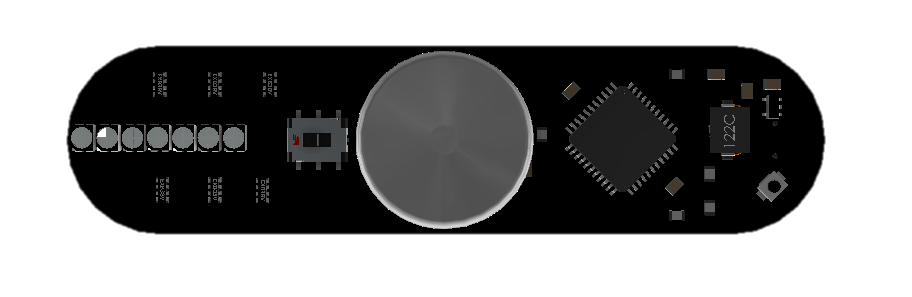
* 1. Bedienungsanleitung

**Text anzeigen**

Um einen Text anzeigen zu können muss nur der Schalter in Richtung Mitte gestossen werden und der Spinner gedreht werden.

**Farbe wechseln**

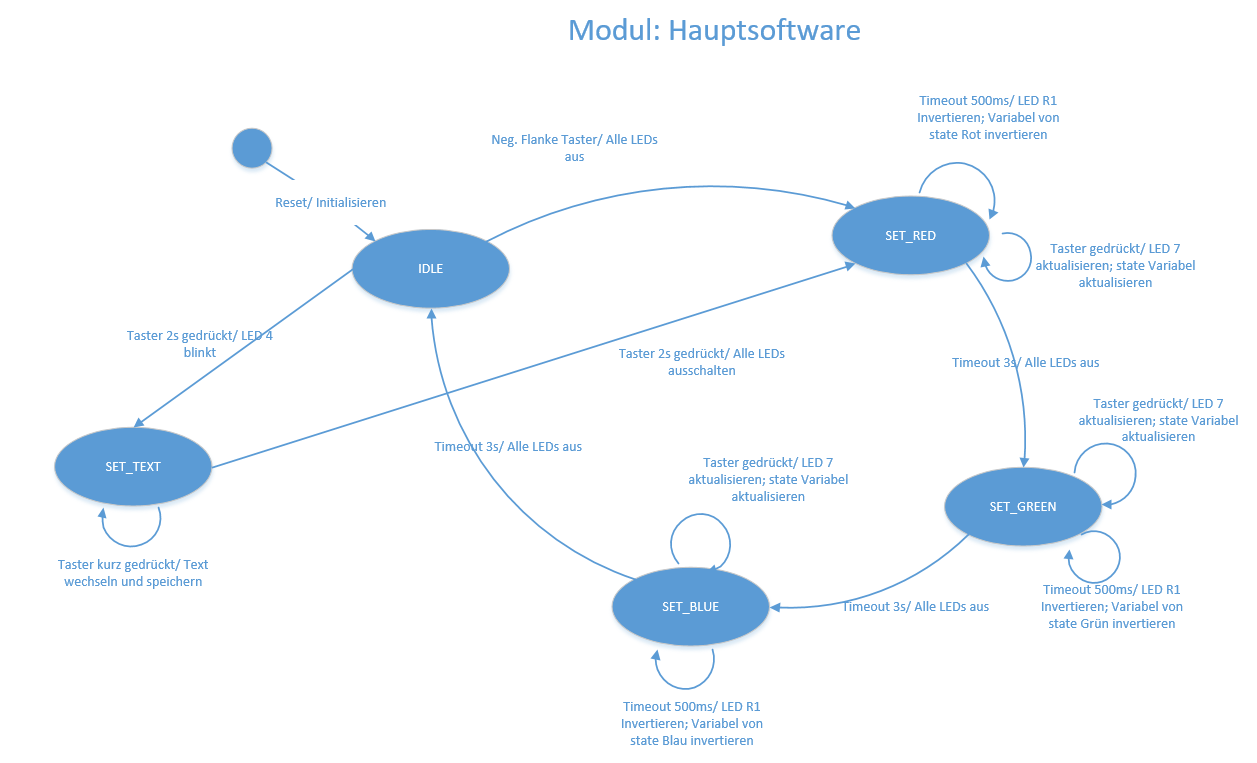
Um eine Farbe zu wechseln muss zuerst der Taster kurz gedrückt werden. Anschliessend blinkt die 1. LED in den Farben Rot, Grün und Blau. Bei jeder Farbe kann mit einem kurzen Tastendruck entschieden werden ob die Farbe leuchten soll oder nicht (Farbmischung ist möglich). Standartmässig leuchten die LEDs in Grün.



Taster

Power-Schalter OFF/ON

* 1. Zustandsdiagramm Hauptsoftware



* 1. Zeitplan

