

# Fachbericht

PROJEKT 6 COCKTAILMASCHINE - TEAM SCHENK & AEBI  
9. August 2020

<b>Betreuer Dozent:</b>	Prof. Dr. Schleuniger, Pascal
<b>Team:</b>	Schenk, Kim Aebi, Robin
<b>Studiengang:</b>	Elektro- und Informationstechnik
<b>Semester:</b>	Frühlingssemester 2020

## **Abstract**

In diesem Projekt wurde ein Konzept erstellt, um eine Cocktailmachine zu bauen. Dies reicht von der Analyse, was für Cocktailmaschinen es bereits gibt, über die Erstellung eines Grob- und eines Detailkonzeptes bis hin zur Evaluation der Komponenten. Der Aufbau wurde so gewählt, dass ein Glas mittels eines Linearantriebes auf einem Schlitten hin- und her gefahren wird und unter dem gewünschten Flüssigkeitsauslass stehen bleibt, wo es dann befüllt wird. Die Bedienung soll über ein Touch-Display geschehen. Die Verarbeitung der Daten wird ein Mikrocontroller übernehmen. Als mechanische Komponente wird pro Zutat eine Pumpe und ein Durchflusssensor verwendet sowie ein einzelner Motor, welcher den Linearantrieb mit dem Schlitten betreibt. Als Motor wurde ein bürstenloser Gleichstrommotor verwendet, da dieser ein sehr gutes Leistungs-/Gewicht-Verhältnis aufweist und in seiner Ansteuerung sehr interessant ist. Ziel des Projekt 5 war es, anhand des Konzeptes die einzelnen Teilsysteme aufzubauen und deren Funktion zu verifizieren und zu dokumentieren. Softwaremäßig wurde die Basis für den Mikrocontroller geschrieben. Dies bedeutet, dass die Teilsysteme kontrollierbar sind und im Projekt 6 ausgebaut und zusammengeführt verwendet werden können. Die Software wurde komplett in C geschrieben und ausgiebig dokumentiert. Das Resultat zeigt, dass die Komponenten zusammenpassen und der Cocktailmachine im Projekt 6 nichts im Weg steht.

## Inhaltsverzeichnis

<b>1 Einleitung</b>	<b>1</b>
<b>2 Ausgangslage</b>	<b>2</b>
2.1 Blockschaltbild . . . . .	3
<b>3 Neue Hardware</b>	<b>4</b>
3.1 Wirelessmodul . . . . .	4
3.2 USB-B . . . . .	6
3.2.1 Handshake ATMega2560 . . . . .	6
3.2.2 Handshake ESP32 . . . . .	7
3.3 RFID . . . . .	9
3.4 SD-Karte . . . . .	9
3.5 Beleuchtung . . . . .	11
<b>4 Printaufbau</b>	<b>12</b>
<b>5 Teilsysteme</b>	<b>13</b>
5.1 Speisungen . . . . .	13
5.1.1 48V Speisung . . . . .	13
5.1.2 12V Speisung . . . . .	14
5.1.3 5V Speisung . . . . .	15
5.1.4 3.3V Speisung . . . . .	17
5.2 Motor . . . . .	18
5.2.1 BLDC . . . . .	19
5.2.2 H-Brücke . . . . .	20
5.2.3 ABN-Encoder . . . . .	21
5.2.4 FOC-Treiber TMC4671 . . . . .	22
5.2.5 Gate-Treiber . . . . .	26
5.3 Flüssigkeitsbeförderung . . . . .	28
5.3.1 Pumpen . . . . .	28
5.3.2 Durchflussmessgeräte . . . . .	29
5.4 Programmierschnittstellen . . . . .	30
5.4.1 USB-B . . . . .	30
5.5 Benutzerschnittstellen . . . . .	32

5.5.1	Display	32
5.5.2	ESP	33
5.5.3	RFID	35
5.6	Beleuchtung	36
5.7	Mikrocontroller	37
5.8	SD-Karte	38
<b>6</b>	<b>Inbetriebnahme</b>	<b>38</b>
6.1	Speisungen	39
6.1.1	12V Speisung	39
6.1.2	5V Speisung	39
6.1.3	3.3V Speisung	39
6.2	Programmierschnittstellen	40
6.2.1	USB-B	40
6.3	Benutzerschnittstellen	45
6.3.1	Touch-Display	45
6.3.2	ESP	45
6.3.3	RFID	46
6.4	Mikrocontroller	47
6.4.1	Bootloader	47
6.4.2	Fuse-Bits	47
6.4.3	AVRdude in Atmel Studio einbinden	48
6.4.4	Vorgehen	48
6.4.5	Messungen	49
6.5	Datenspeicherung	50
6.5.1	mikroSD-Karte	50
6.6	Flüssigkeitsbeförderung	51
6.6.1	Pumpen	51
6.6.2	Durchflussmessgeräte	51
6.7	Beleuchtung	51
6.8	Motor	51
6.8.1	BLDC und H-Brücke	51
6.8.2	ABN-Encoder	51
6.8.3	FOC-Treiber	51
6.8.4	Gate-Treiber	52

<b>7 Software</b>	<b>54</b>
7.1 Atmega2560 . . . . .	54
7.1.1 Strukturplan . . . . .	55
7.1.2 Programmflussdiagramm . . . . .	56
7.1.3 Doubly Dynamic Linked Lists Allgemein . . . . .	58
7.1.4 Doubly Dynamic Linked Lists Cocktailmixer . . . . .	60
7.2 ESP32 . . . . .	60
7.2.1 Programmflussdiagramm . . . . .	61
7.3 Displaysoftware . . . . .	61
7.4 Android App . . . . .	61
<b>8 Evaluation</b>	<b>61</b>
<b>9 Fazit</b>	<b>61</b>
9.1 Zielerreichung . . . . .	61
9.2 Kosten . . . . .	61
<b>10 Schlusswort</b>	<b>61</b>
<b>11 Ehrlichkeitserklärung</b>	<b>61</b>
<b>Literatur</b>	<b>62</b>
<b>A TMC4671</b>	<b>63</b>
A.1 Standard-Schaltkreis TMC4671 . . . . .	63
A.2 Blockdiagramm TMC4671 . . . . .	63
A.3 Inbetriebnahme Gate-Control . . . . .	64
<b>B TMC6200</b>	<b>69</b>
B.1 Standard-Schaltkreis TMC6200 . . . . .	69
B.2 Blockdiagramm TMC6200 . . . . .	69
B.3 Verstärkungsfaktor, Strommessung, Strommesswiderstand . . . . .	70
B.4 Gate-Vorwiderstand . . . . .	70
B.5 Externe Gate-Spannungsversorgung . . . . .	70
<b>C H-Brücke</b>	<b>71</b>
C.1 Referenzschema . . . . .	71

<b>D Mikrocontroller</b>	<b>71</b>
D.1 Brown-out-Detection . . . . .	71
D.2 Full Swing Crystal Oscillator . . . . .	71
D.3 Bootloader-Speicherplatz . . . . .	72
D.4 Memory-Lock Bootloader . . . . .	72
<b>E USB-B</b>	<b>73</b>
E.1 Geräte-Manager . . . . .	73
<b>F Atmel Studio</b>	<b>73</b>
F.1 Fuse Bits . . . . .	73
F.2 Lock Bits . . . . .	73
F.3 Einbinden AVRdude und stk500v2 (wiring) . . . . .	74
F.4 Bootloader "Brennen" . . . . .	74

## 1 Einleitung

Eine gelungene Party auf die Beine zu stellen verlangt einem einiges ab. Vor allem kostet es eine Menge Aufwand und Zeit. Dies gilt besonders, wenn es darum geht mit vielen Freunden zusammen zu feiern. Neben der gelungenen Musikauswahl und den Snacks darf eines auf gar keinen Fall fehlen, die Getränke. Um diese sicherzustellen, gibt es mehrere Möglichkeiten. Einerseits könnte jeder seine eigenen Getränke mitbringen, was jedoch bedeutet, dass es unter Umständen eine riesige Sauerei gibt oder viele Flaschen in der Gegend rumstehen. Anderseits könnte man als Gastgeber selber anbieten Cocktails zu mixen und so den Getränkenachschub zu gewährleisten. Da gibt es jedoch ein grosses Problem. Als Gastgeber möchte man nicht den ganzen Abend hinter der Bar stehen müssen, sondern lieber bedenkenlos mitfeiern. Damit genau dies möglich ist haben wir uns dazu entschieden eine automatisierte Cocktailmaschine zu entwerfen. Diese soll vollkommen autonom arbeiten und sollte problemlos von jeder beliebigen Person und in fast jedem Zustand bedient werden können.

In den folgenden Kapiteln ist dokumentiert, wie die Cocktailmaschine im Detail aussieht. Dazu gehören die elektronischen Teilsysteme, das dazugehörige Printdesign, die Software, die Mechanik und die Evaluierung.

## 2 Ausgangslage

Die Basis für das Projekt 6 hat das Projekt 5 gebildet, in welchem ein Konzept erstellt wurde, welches die Hauptkomponenten der Maschine festlegte und dessen Arbeitsweise. Aus dieser Entscheidungsfindung wurde dann ein Blockschaltbild erstellt, welches in Abbildung 2.1 zu sehen ist. Ein weiterer Teil des Projekt 5 war es, die gewählten Komponenten und die daraus entstandenen Teilsysteme in einem Testaufbau aufzubauen und zu evaluieren. Dazu gehörten die Speisungen (48V, 12V, 5V und 3.3V), der Mikrocontroller, das Display, der Motor, die Pumpenansteuerung und die Durchflussmessgeräte.

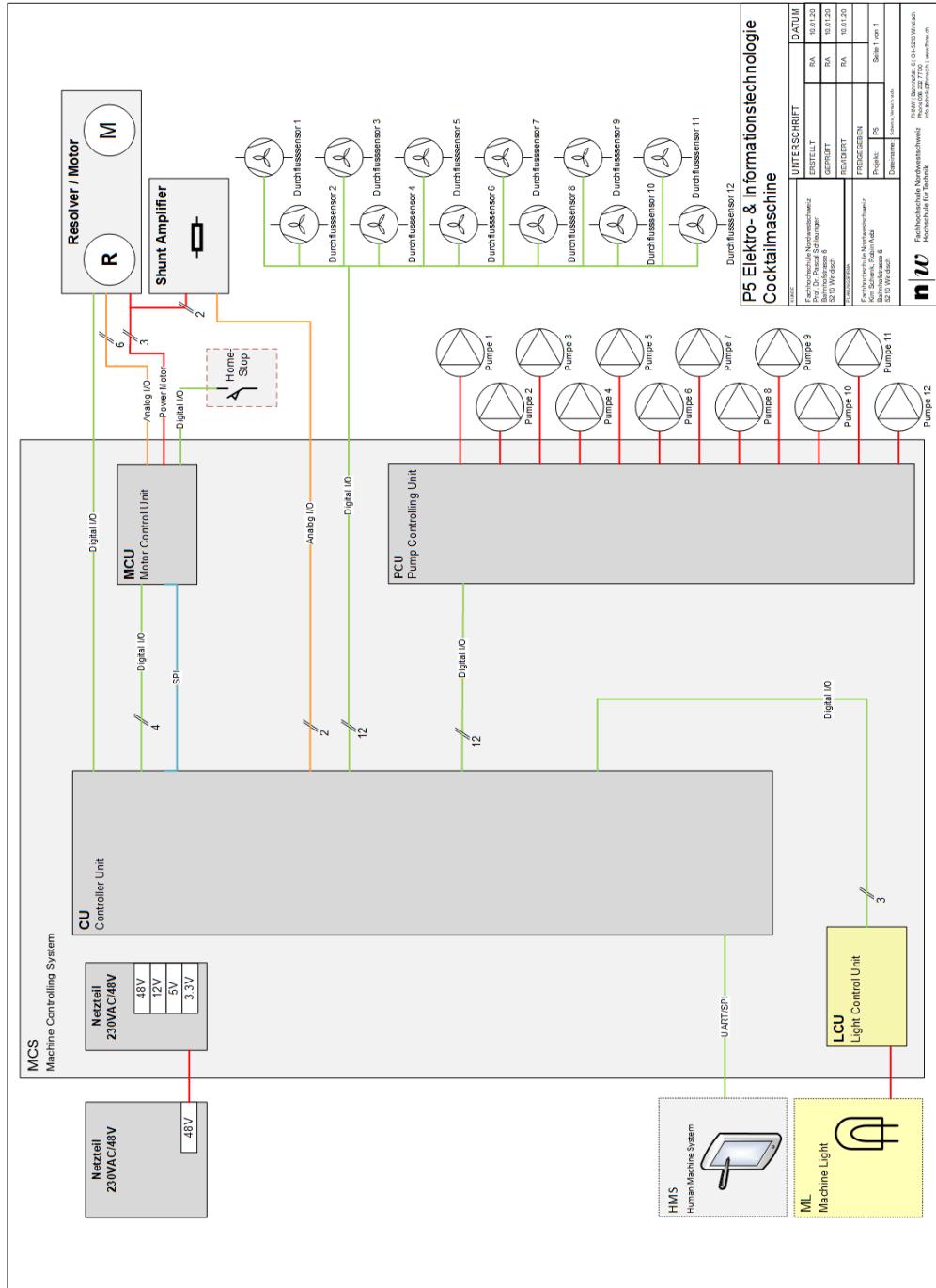


Abbildung 2.1: Blockschaltbild des CocktailMixer's gemäß Projekt 5.

## 2.1 Blockschaltbild

Das in Projekt entstandene Blockschaltbild ist nun um weitere Teilsysteme ergänzt worden, auf welche im folgenden Kapitel 3 eingegangen wird. Zu den neuen Systemen gehört ein Bluetooth- / Wirelessmodul, welches eine externe Ansteuerung per Web-Server oder Android App ermöglicht, eine dazugehörige Programmierschnittstelle, ein RFID Lesegerät, ein SD-Karten Slot und eine Maschinenbeleuchtung.

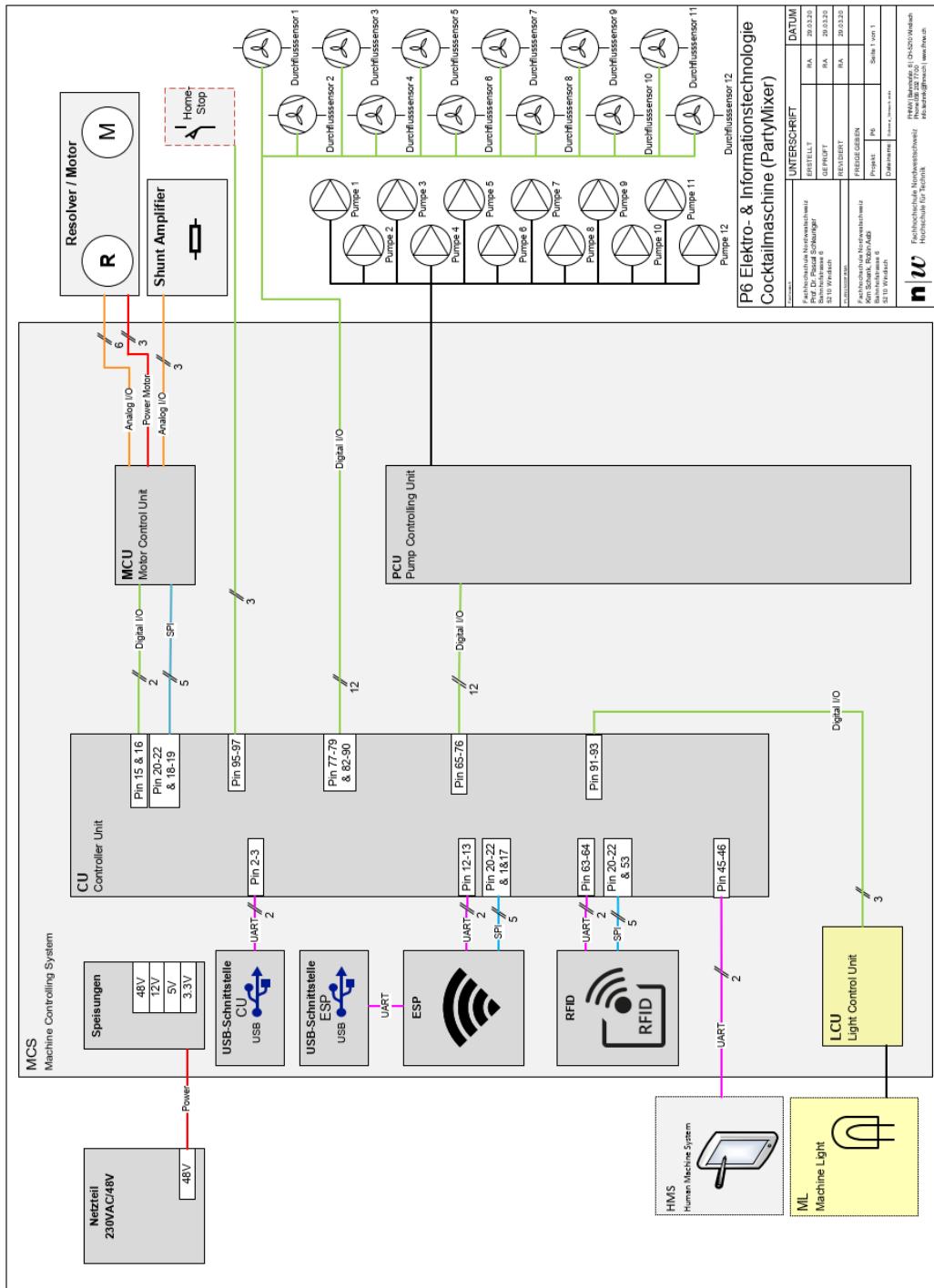


Abbildung 2.2: Blockschaltbild des CocktailMixer's gemäß Projekt 6.

### 3 Neue Hardware

Im folgenden Kapitel werden die Hardware-Teile beschrieben, welche im Projekt 5 noch nicht erarbeitet wurden. Zuerst werden grundlegende Anforderungen und Inhalte beschrieben, welche zur Auswahl der Bauteile geführt hat.

#### 3.1 Wirelessmodul

Über einen Web-Host soll der User die Möglichkeit haben, Getränke auszuwählen und seinem RFID-Chip zuzuordnen, sowie diverse kleinere Einstellungen an der Maschine vorzunehmen. Dazu wird ein WiFi-Modul benötigt, welches auch den Webhost bereitstellen kann.

Aufgrund schon bestehender Erfahrungen wurde ein Espressif ESP-Modul ausgewählt. Grundsätzlich standen zwei Modelle zur Auswahl. Das ESP8266 und das ESP32. Für die Cocktailmachine wurde das ESP32 ausgewählt, da dies Leistungsstärker ist. Die genauen Datenvergleiche sind in Tabelle 3.1 ersichtlich. Das ESP32 unterstützt das Protokoll nach ISO 802.11 b/g/n und kann somit auf 2.4GHz sowie 5GHz arbeiten. Mit dem n-Protokoll und einer Antenne kann so bis zu 150MBit übertragen werden bei einer Bandbreite von 20MHz.

MCU	Xtensa Single-Core 32-bit L106 (ESP8266)	Xtensa Dual-Core 32-bit LX6 (ESP32)
802.11 b/g/n	HT20	HT40
Bluetooth	No	Bluetooth 4.2 and BLE
Arbeitsfrequenz	80 MHz	160 MHz
SRAM	No	Yes
Flash	No	Yes
GPIO	17	36
SPI/I2C/I2S/UART	2/1/2/2	4/2/2/2
ADC	10-bit	12-bit
Ethernet Interface	No	Yes
Touchsensor	No	Yes
Temperatursensor	No	Yes
Hall-Sensor	No	Yes
Arbeitstemperatur	-40°C to 125°C	-40°C to 125°C
Price	\$ (3\$ - \$6)	\$\$ (\$6 - \$12)

Tabelle 3.1: Vergleich ESP8266 zu ESP32.

Wichtig ist, dass ein ESP ausgewählt wird, welches einen Anschluss für eine abgesetzte Antenne hat, da die Leiterplatte, worauf das Modul verbaut wird, im Gehäuse platziert wird. Dazu eignet sich der Espressif ESP32-32U.

In Tabelle 3.2 sind die Strapping-Pins aufgelistet, welche während dem Startvorgang einen Einfluss auf die Konfiguration des ESP32 haben. Mit diesen Pins kann die Ausgangsspannung des internen Spannungsregler für VDD\_SDIO, der Bootmodus, der Debug-Log-Print während dem Booten sowie das SPI-Timing der Kommunikation mit der SDIO-Schnittstelle des ESP32 konfiguriert werden.

<b>Ausgangsspannung internen Spannungsregler (VDD_SDIO)</b>						
RS232	ESP	default	<b>3.3V</b>	<b>1.8V</b>		
RTS	IO12	Pull-down	0 <sup>1</sup>	1		
<b>Boot-Modus</b>						
RS232	ESP	default	<b>SPI-flash Boot</b>	<b>Download Boot</b>		
DTR	IO0	Pull-up	1	0		
-	IO2	Pull-down	Egal	0		
<b>Debugging Log Print über U0TXD während Booten</b>						
RS232	ESP	default	<b>U0TXD Active</b>	<b>U0TXD Silent</b>		
RTS	IO12	Pull-down	1	0		
<b>Timing<sup>2</sup> des SDIO</b>						
RS232	ESP	default	FF Sampling FF Output	FF Sampling SF Output	SF Sampling FF Output	SF Sampling SF Output
CTS	IO15	Pull-up	0	0	1	1
-	GPIO5	Pull-up	0	1	0	1

**Tabelle 3.2:** Tabelle Pinkonfiguration für Strapping-Pins.

### Spannung des internen Spannungsgeglers (VDD\_SDIO)

Das ESP32 hat einen eingebauten host controller für SD/SDIO/MMC-Speichergeräte. Dieser kann mit 3.3V (IO12 = 0) oder 1.8V (IO12 = 1) betrieben werden. Wird der Pin auf 1 gesetzt, könnte ein Brown-out der IO-Versorgungsspannung VCC\_IO ausgelöst werden.

**Booting Mode** Der Pin IO0 gibt vor, ob das ESP32 vom internen SPI-flash bootet (IO0 = 1) oder ob neuer Code in den flash Speicher geschrieben wird (IO0 = 0). Soll das ESP vom Flash-Speicher booten, so hat der Pin IO2 kein Einfluss, um jedoch den Code speichern zu können, muss der Pin auf 0 sein.

### De-/Aktivieren vom Debug-Log über U0TXD während dem Bootvorgang

Über den Pin IO12 kann konfiguriert werden, ob während dem Booten ein Debug-Log über die Serielleschnittstelle gesendet wird (IO12 = 1) oder nicht (IO12 = 0).

### Timing der Kommunikation mit dem SDIO Slave:

Über die Pins IO15 und IO5 kann das Übertragungsprotokoll des SDIO-Slaves festgelegt werden. Dabei kommt es darauf an, ob das Sampling auf eine fallende Flanke (IO15 = 0) oder steigende Flanke (IO15 = 1) geschehen soll, und ob der Output auf eine fallende Flanke (IO5 = 0) oder steigende Flanke (IO5 = 1) geschehen soll.

<sup>1</sup>Da das ESP32-WROOM-32U einen 3.3V SPI flash integriert hat, kann der MTDI nicht auf 1 gesetzt werden, wenn die Module aufgestartet sind.

<sup>2</sup>FF = Fallende Flanke, SF = Steigende Flanke

## 3.2 USB-B

Auf der Leiterplatte des PartyMixer's gibt es zwei Komponenten, welche programmiert werden müssen. Der Mikrocontroller und das WiFi-Modul. Um diese zu programmieren braucht es eine entsprechende Schnittstelle, welche mit einer USB-B-Schnittstelle realisiert wird.

Die USB-B-Schnittstelle benötigt nur zwei Kommunikationsleitungen (D+ und D-). Die zu programmierenden Komponenten (ESP32 und ATMega2560) benötigen zusätzliche Steuerleitungen um in einen Programmiermodus zu kommen und statt einem differenziellen Verfahrens ein serielles Verfahren. Deswegen wird ein USB-UART-Converter verwendet, welcher das Signal wandelt und die Steuerleitungen zur Verfügung stellt.

Die beiden Programmierschnittstellen (ATMega2560 und ESP32) benötigen unterschiedlich viele Steuerleitungen, da sie sich im Verfahren zum Aufruf des Download-Boot-Modus unterscheiden. In Tabelle 3.3 und 3.4 wird dargestellt, welche Leitungen für das jeweilige Flash-Interface benötigt werden. Die Signalfolge, um den Programmier-Modus zu starten, wird in den Kapiteln 3.2.1 und 3.2.2 beschrieben.

Mikrocontroller			USB-Flash-Device
RX	<==	direkt	==== TX
TX	====	direkt	==> RX
Reset	<==	Kondensator	==== DTR

**Tabelle 3.3:** Verbindung zwischen USB und Mikrocontroller.

ESP			USB-Flash-Device
RX	<==	direkt	==== TX
TX	====	direkt	==> RX
EN	<==	über Transistor	==== RTS
IO_0	<==	über Transistor	==== DTR
IO_13	<==	über Widerstand	==== RTS
IO_15	<==	über Widerstand	==== CTS

**Tabelle 3.4:** Verbindung zwischen USB und ESP.

### 3.2.1 Handshake ATMega2560

Der Handshake zwischen Computer und ATMega2560 ist einfacher zu verstehen als der Handshake zwischen Computer und ESP32. Da für den ATMega2560 ein stk500v2-Bootloader verwendet wird, erwartet dieser nur ein Flankenwechsel am Reset-Pin. Dadurch wird automatisch der im Programmspeicher abgelegte Bootloader gestartet und auf einkommende Daten gewartet. Nachdem das ProgrammierTool AVRdude die Device-Signatur (0x1E9801) verifiziert hat, sendet es das kompilierte HEX-File an den ATMega2560.

Um die Daten hochladen zu können, muss sichergestellt werden, dass avrdude mit dem stk500v2 programmer hochlädt und die DTR-Leitung toggelt, bevor der Programmervorgang stattfindet. Die DTR-Leitung hängt über einen Kondensator an der Reset-Leitung und zieht diese auf GND, womit der Bootloader des ATMega2560 aktiv wird. Deswegen wird beim Hochladen in AVRdude als programming-id der *wiring*-programmer angegeben, sichtbar in Kapitel 6.4.3 Tabelle 6.2 (-c wiring).

### 3.2.2 Handshake ESP32

Nach einem Neustart des ESP32 werden die Zustände der Strapping-Pins gelesen. Anhand dieser Zustände werden die Grundzustände des ESP32 gesetzt, bevor der Programmcode gestartet wird. Anhand der Strapping-Pins wird auch der Download-Boot-Modus gestartet. Um den Handshake zwischen Computer und ESP32 zu verstehen, muss zuerst Tabelle 3.5 betrachtet werden, worin die Strapping-Pins aufgelistet sind und welchen Einfluss diese auf denn Boot- oder Download-Modus haben. Aufgrund der defaultmässigen Pull-up und -down Widerstände kann aus dieser interpretiert werden, dass wenn U0TXD, IO2 und IO5 "floating" sind, IO0 den Boot-Modus bestimmt.

Boot-Mode Konfiguration			
Pin	Default	Boot	Download
IO0	1	1	0
U0TXD	1	1	Don't care
IO2	0	Don't care	0
IO15	1	Don't care	Don't care
IO5	1	1	Don't care

**Tabelle 3.5:** Wenn U0TXD, IO2, IO5 floating sind, bestimmt IO0 den Boot-Modus.

Im Getting Started Guide vom ESP32 steht geschrieben, dass wenn während die beiden Pins IO0 und EN GND gezogen werden, das Modul in den Download-Modus versetzt wird, die Firmware über den UART-Port herunterlädt und diese in den Flash-Speicher schreibt. Um dies über die DTR- und RTS-Leitung machen zu können, braucht es eine zusätzliche Programmierlogik.

Mit der Schaltung, wie sie in Abbildung 5.22 ersichtlich ist, kann das ESP32-Modul mit einer bestimmten Signalabfolge an den Leitungen DTR und RTS in diesen Download-Boot-Modus gesetzt werden. Die Auswirkung der Schaltung auf IO0 und den EN-Pin kann in Tabelle 3.6 entnommen werden.

Auto Program Cirquit			
DTR	RTS	EN	IO0
1	1	1	1
0	0	1	1
1	0	0	1
0	1	1	0

**Tabelle 3.6:** Strapping pins des ESP32 und deren Auswirkung auf den Boot- und Download-Modus beim aufstarten.

Die Programmierlogik wird aus dem Programmiertool angesteuert. Es befindet sich in der ESP-Bibliothek, welche verwendet wird, um Code aus der Arduino-Umgebung hochzuladen. Aus dem Python-Skript kann entnommen werden, wie die Pins angesteuert werden. Der Code ist in Abbildung 3.1 dargestellt.

Bei der Interpretation des Codes ist darauf zu achten, dass die beiden Pins DTR und RTS Active-Low sind. (True = 0V = 0 = LOW und False = VCC = 1 = HIGH). Der Bezug zwischen dem Upload des Programms, den Pins des ESP32 und dessen Download-Boot-Modus sowie den Leitungen DTR und RST wird in Tabelle 3.7 aufgezeigt.

Referenz  
einfügen:  
<http://loboris.com>

```

# issue reset-to-bootloader:
# RTS = either CH_PD/EN or nRESET (both active low = chip in reset
# DTR = GPIO0 (active low = boot to flasher)
#
# DTR & RTS are active low signals,
# ie True = pin @ 0V, False = pin @ VCC.
if mode != 'no_reset':
    self._setDTR(False) # IO0=HIGH
    self._setRTS(True) # EN=LOW, chip in reset
    time.sleep(0.1)
    if esp32r0_delay:
        # Some chips are more likely to trigger the esp32r0
        # watchdog reset silicon bug if they're held with EN=LOW
        # for a longer period
        time.sleep(1.2)
    self._setDTR(True) # IO0=LOW
    self._setRTS(False) # EN=HIGH, chip out of reset
    if esp32r0_delay:
        # Sleep longer after reset.
        # This workaround only works on revision 0 ESP32 chips,
        # it exploits a silicon bug spurious watchdog reset.
        time.sleep(0.4) # allow watchdog reset to occur
    time.sleep(0.05)
    self._setDTR(False) # IO0=HIGH, done

```

**Abbildung 3.1:** Codeausschnitt aus dem Programmier-Tool. Auf diese Weise werden der DTR- und der RTS-Pin angesteuert während dem Programmervorgang.

Schritt	Beschreibung	DTR	RTS	EN	IO0
1	Im ersten Schritt wird der Chip mit $EN = 0$ und $IO0 = 1$ in den Reset-Modus geschaltet und im Programm 0.1ms gewartet.	1	0	0	1
2	Im zweiten Schritt werden die Zustände geändert, jetzt ist $EN = 1$ und $IO0 = 0$ . Aufgrund der Kapazität am EN-Pin wird die Spannung für einen kurzen Moment tief gehalten. Dies hat Schritt 3 zur Folge.	0	1	1	0
3	Die durch den Kondensator C7 tief gehaltene Spannung bewirkt, dass sich die Zustände über den Pins wie folgt verhalten: $EN = 0$ und $IO0 = 0$ . Im Programm wird 0.05ms gewartet. Danach ist das ESP32 im Download-Boot-Modus und das zu speichernde Programm kann hochgeladen werden.	0	1	0	0
4	Nachdem das Programm hochgeladen wurde, werden beide Zustände auf 1 gesetzt. Das ESP32 startet nun den neu hochgeladenen Programmcode.	1	1	1	1

**Tabelle 3.7:** Abfolge der Schritte während dem Aufrufen des Download-Boot-Modus.

### 3.3 RFID

Eine vorgesehene Funktion der Maschine ist, dass der User ohne Suchen sein Lieblingsgetränk zubereiten lassen kann. Dafür wird auf ein System zurückgegriffen, was öfters für Zutrittskontrollen oder Ähnliches verwendet wird. Nämlich RFID<sup>3</sup>.

Um schnell ein funktionierendes Teilsystem testen zu können, soll ein Breakout-Board verwendet werden. Eine Recherche hat ergeben, dass es nicht allzu viele Produkte gibt. Ein Eingrenzungskriterium war, dass es für den Ausgewählten IC eine bestehende Library gibt, welche mit Arduino oder besser C kompatibel ist. Der Chip, welcher diese Anforderungen erfüllt, ist der **Mifare MFRC522**. Dieses Brakeout-Board kann außerdem an der Verschalung der Maschine angeschraubt werden. Somit kann er sich an einer Stelle befinden, welche weiter weg von der Hauptsplatine ist. Das Modul arbeitet auf 13.56MHz und gilt somit als kurzwelliges System (HF).

Der MFRC522 ist der Reader im RFID-System. Er kann Daten lesen und ggf. auch schreiben. Er erzeugt ein hochfrequentes, elektromagnetisches Wechselsignal mit einer Frequenz von 13.56MHz. Der RFID-Tag wird von der Energie des Wechselsignals gespiesen. Durch Kurzschliessen der Tag-Antenne wird ein Teil der Energie des vom Reader ausgehenden Wechselfeldes verbraucht. Diese Energiedifferenz kann ein Reader detektieren. Die Reichweite für ein solches System beträgt weniger als 10cm.

Alternativ könnte ein Tag mit einer Batterie gespiesen werden. Somit erhöht sich die Reichweite des Readers, die Latenzzeiten werden kürzer und der Anwendungsbereich würde grösser.

### 3.4 SD-Karte

Damit gespeicherte Getränke, Zutaten und Maschinenzustände bei einem Neustart der Cocktailmachine erhalten bleiben, werden diese auf der SD-Karte abgelegt. Neben dem Erhalten der Informationen ist die SD-Karte wichtig für den Betrieb der Maschine. Werden zu viel Getränke und Zutaten gespeichert, so führt dies zu einem Absturz des Mikrocontrollers. Es wurde entschieden, eine mikroSD zu verwenden. Im Folgenden wird erklärt, was es Hard- und Softwaretechnisch zu beachten gibt.

Abbildung 3.2 zeigt die Pinbelegung einer mikroSD-Karte. Es ist erkennbar, dass sie über SPI kommuniziert. Für den Anschluss wird nur ein mikroSD-Sockel benötigt. So kann die mikroSD über den Level-Shifter vom Mikrocontroller angesteuert werden.



Abbildung 3.2: Pinout des mikroSD-Sockels.

<sup>3</sup>Radio Frequency IDentification

Als Dateisystem wird FAT32 verwendet. FAT32 steht für File Allocation Table mit 32Bit Datenbreite. Es ist ein von Microsoft entwickeltes System, dessen Wurzeln bis ins Jahr 1977 zurückreichen und heute noch der Industriestandard unter den Dateisystemen ist.

Der Speicherbereich einer FAT32-formatierten Partition besteht aus fünf Bereichen.

1. Master Boot Record (LBA Sektor 0 des Laufwerks)
2. Volume Boot Record, Partition Boot Sektor (LBA Sektor 0 der Partition)
3. File Allocation Table (i.d.R 2 mal hintereinander vorhanden, direkt nach dem PBR)
4. Directory Table (mit den Ordner und Fileeinträgen)
5. Datenbereich (Fileinhalt)

Im **Master Boot Record** ist ein Startprogramm vorhanden, welches für BIOS baseierte Computer geschrieben wurde. Darin enthalten sind der Boot-Code in Assembler, Fehlermeldungen je nach Landessprache, Darstellungen der Fehlermeldungen, die Disk Signature, für die Organisation des Laufwerks wichtige Partitionstabellen und die Signature ID oder auch Magic Number 0x55 0xAA. Hier wird die aktive Partition ausgewählt und der Boot Sektor geladen. Da nicht über die SD-Karte gebootet wird, besteht dieser Teil nur aus Gründen der Kompatibilität.

Im **Volume Boot Record** befinden sich 3 Sektoren. Der erste Sektor enthält einen Sprungbefehl und einen "tu nichts" Befehl ("nop"), den Systemname (OEM ID), den BIOS Parameter Block, den Boot Code, der Bootloader File Name, Fehlermeldungen, Darstellung der Fehlermeldungen und ebenfalls die Magic Number 0x55 0xAA. Der zweite Sektor enthält den Anfang des erweiterten Volume Boot Records eines FAT32 Systems, den Anfang der Daten und das nächste verfügbare Cluster sowie die Magic Number. Hier kann kalkuliert werden, wieviel freier Speicher noch verfügbar ist. Der dritte Sektor ist eine Erweiterung des ersten Sektors, falls der Speicher für den Bootcode nicht ausreicht. Unter FAT32 wird für die drei Sektoren eine Sicherheitskopie abgelegt, welche zur Wiederherstellung des Volume Boot Records dient.

Im **File Allocation Table** wird organisiert, welche Cluster auf der Partition belegt oder frei sind und wo die Folgecluster eines Files sind. Die Clustergrösse kann selbst bestimmt werden und beträgt bei SD-Karten mit kleinen Dateien optimalerweise 512Bytes. Dies ist die minimale Grösse, da ein Cluster aus mehreren Sektoren besteht und ein Sektor 512 Bytes gross ist. Für die Cocktailmaschine weden pro File ca. 100 Bytes verwendet, wodurch pro File ca. 412 Bytes unnutzbar werden. Der Nachteil ist, dass die FAT so grösser wird und grosse Dateien mehr zerstückelt werden. Die Cluster sind fortlaufend nummeriert, wobei sich die Nummerierung in der FAT immer auf den Ort innerhalb der FAT bezieht, nicht auf den realen Ort auf der Partition. Ist ein File grösser als 512 Bytes, so ergibt sich eine Clusterkette, welche einen Verweis auf das nächste Cluster enthält und ein Verweis, falls die Kette zu ende ist. Jeder Eintrag der FAT ist 32 Bit lang (4 x 8 Bytes = 4 Doppelwörter). Ein Cluster kann z.B den Wert 0x0E 00 00 00 haben, wobei der Verweis 0E auf das nächste Cluster zeigt. Ein Endcluster ist mit dem Doppelwort 0xFF FF FF 0F gekennzeichnet. Ist eine Datei Fragmentiert, ist die Nummerierung nicht fortlaufend, sondern springt hin und her. Konklusion: Die Werte innerhalb des FAT zeigen an, "wo sich relativ gesehen die Daten eines Files auf der Partition finden lassen. Dabei sind die Werte entweder das Zeichen für ein Clusterende oder sie zeigen auf den nächsten Cluster der Clusterkette innerhalb der FAT". Um jedoch das Anfangscluster auf der Partition finden zu können, muss das Directory Table angeschaut werden.

cite: <https://www.ionos.de/.../aufbau-des-fat32-dateisystems.html>

cite: <https://docplanner.com/.../aufbau-des-fat32-dateisystems.html>

cite: <https://www.silabs.com/documents/public/.../aufbau-des-fat32-dateisystems.html>

cite: <https://docplanner.com/.../aufbau-des-fat32-dateisystems.html>

cite: <https://docplanner.com/.../aufbau-des-fat32-dateisystems.html>

Im **Directory Table** sind alle Files und alle Ordner der Partition beschrieben. Der Startsektor errechnet sich aus der letzten FAT und der Grösse der FAT (Addition). In diesem Bereich befindet sich das Root Directory, oder auch Wurzelverzeichnis genannt. Es hat eine hierarchische Baumstruktur, was bedeutet, dass "das Wurzelverzeichnis das oberste mögliche Verzeichnis ist unter dem sich weitere Verzeichnisse befinden können". Bis auf das Wurzelverzeichnis haben alle anderen Verzeichnisse ein einziges Elternverzeichnis, können aber mehrere Unterverzeichnisse haben.

cite:  
<https://docplata.com/Aufbau-des-fat32-dateisystems.html>

Der Name des Files ist in der Regel 8 Bytes lang, worauf 3 Bytes folgen für die Bezeichnung der Fileart (z.B .txt). Ein Byte steht für die Volume ID, welche Attribute das File oder der Ordner hat. Hier befinden sich der Zeitstempel des Files, das Datum des letzten Zugriffs und das Datum der letzten Änderung des Files. Vier Bytes enthalten das Startcluster des Files oder des Ordners, vier weitere Bytes enthalten die Grösse des Files. Die restlichen Bytes sind mit 0x00 beschrieben. Um den tatsächlichen Startsektor auf der SD-Karte zu finden, wird eine Umrechnungsformel benötigt. Diese und ein Beispiel wie die File-Organisation aussieht, kann dem File "Aufbau des FAT32 Dateisystems" entnommen werden.

cite:  
[http://www.informatik.uni-ulm.de/ni/LectureNotes/OS/OS\\_Lecture\\_07.pdf](http://www.informatik.uni-ulm.de/ni/LectureNotes/OS/OS_Lecture_07.pdf)

Die Ordner sind auf die selbe Weise organisiert, bis auf die Ausnahme, dass in FAT32 keine Größenangaben für Ordner gemacht wird. Die Bytes werden deswegen mit dem Wert 0x00 beschrieben.

cite:  
<https://docplata.com/Aufbau-des-fat32-dateisystems.html>

Für die Cocktailmaschine wird deswegen eine Library benötigt, welche die FAT32-Commands implementiert und gleichzeitig die Operationen auf der SD-Karte handelt. So ist die SD-Karte auch am Computer editierbar und die Befehlsliste reduziert sich auf readFile, writeFile und deleteFile.

### 3.5 Beleuchtung

Damit die Maschine optisch etwas hergibt, wurde entschieden die ganze Maschinerie mit LED-Streifen zu verzieren. Dazu ist einerseits das LED-Band von nötig und die geeignete Ansteuerung dazu. Für die Cocktailmaschine wurde festgelegt, dass der LED-Streifen die benötigten Widerstände für die LED's schon aufgeklebt hat und die Ansteuerung folglich direkt über FET's geschehen kann. Der Aufbau ähnelt dann dem in Abbildung 3.3 gezeigten Schaltung. Der Streifen wäre Rot eingerahmt, die LED's und Widerstände befinden sich auf dem Band und die zu sehenden Fähnchen für R, G, B und W führen zum Mikrocontroller. Es können auch mehrere Bänder parallel geschaltet werden, wie in Abbildung 3.3 ersichtlich ist.

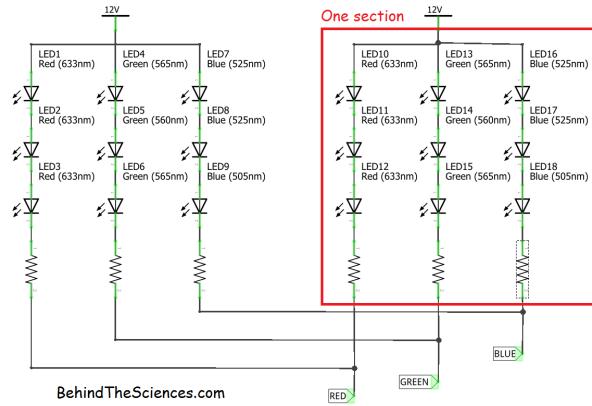


Abbildung 3.3: LED Beispiel.

## 4 Printaufbau

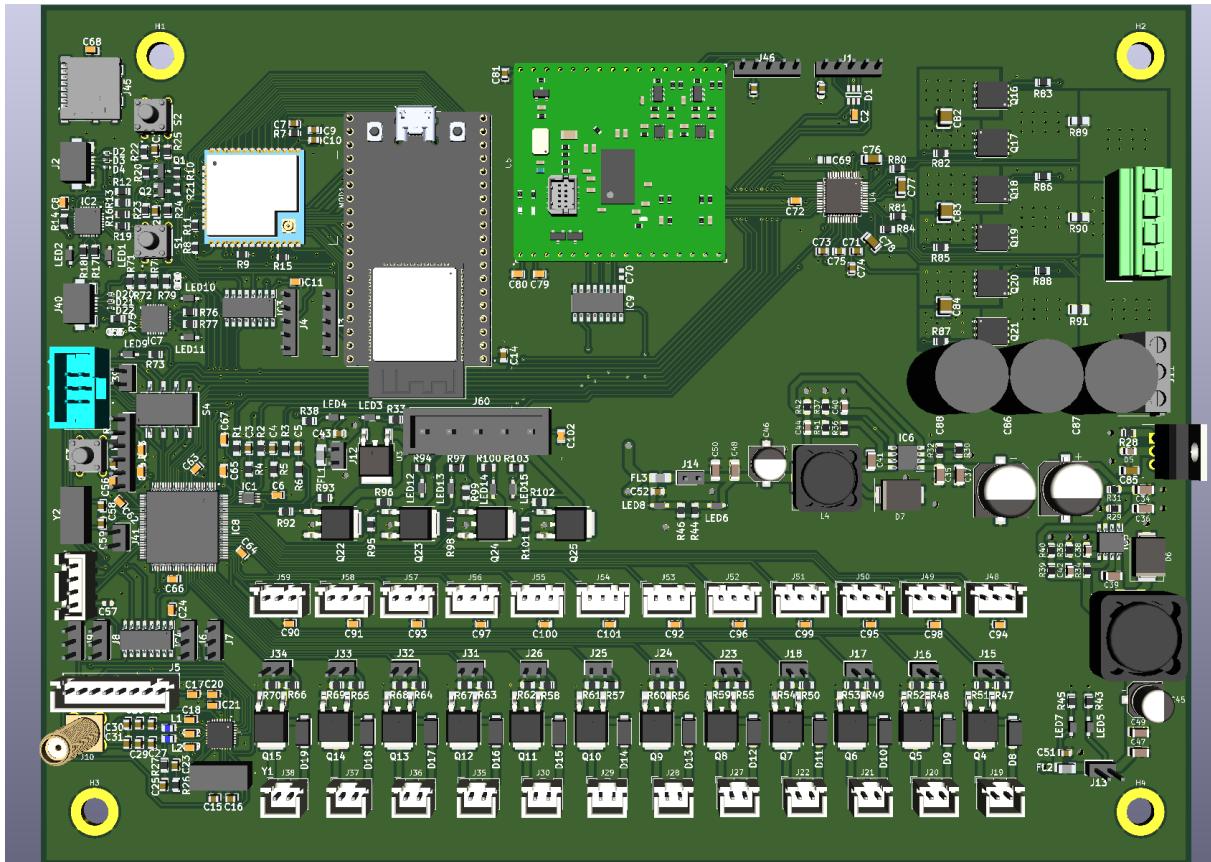


Abbildung 4.1: Print 3D

## 5 Teilsysteme

Da der Partymixer aus vielen kleineren und grösseren Teilsystemen besteht, werden diese in diesem Kapitel einzeln aufgelistet und im Detail angeschaut. Es wird dabei bei jedem Teilsystem auf drei Punkte eingegangen, die Problemstellung (Was ist der Zweck der Teilschaltung und weshalb wird sie benötigt?), das Schema und der Funktionsbeschrieb der Schaltung. Das komplette Schema kann in **Anhang Schema** begutachtet werden.

### 5.1 Speisungen

Ohne Speisung funktioniert keine elektronische Schaltung. Sie bildet daher einen essentiellen Bestandteil des Partymixer's. In dem System befinden sich vier verschiedene Speisungen. Die Ausgangsspannung für alle anderen Speisespannungen bildet dabei ein 48V Netzteil. Aus dieser Spannung wird mittels Step-Down Reglern eine 12V und eine 5V Speisung erzeugt. Bei der vierten Spannung handelt es sich um einen einfachen Linearregler, welcher aus den 5V eine 3.3V Speisung realisiert.

#### 5.1.1 48V Speisung

Der Motor wird mit einer Spannung von 48V betrieben. Dies ist zugleich auch die höchste verwendete Speisespannung. Um diese Speisung gewährleisten zu können, wird ein fertiges Netzteil gemäss **Fachbericht 5** eingesetzt.

Es wurde im Projekt 5 entschieden, dass die 48V Speisung extern als fertiges Netzteil eingekauft wird. Somit entfällt das Schema für diesen Speisungsteil. Ein Anschauungsbild des eingesetzten Netzteils kann in Abbildung 5.1 begutachtet werden.



**Abbildung 5.1:** Anschauungsbild des 48V Netzteils

Es musste jedoch unbedingt eine Leistungsabschätzung gemacht werden. Auch diese wurde im Projekt 5 durchgeführt. Unter Berücksichtigung der Schaltungsteile welche noch im Projekt 6

ergänzt werden, wurde dieses dann ausgewählt und eingekauft. Die Leistungsabschätzung kann im **Fachbericht 5** eingesehen werden.

### 5.1.2 12V Speisung

Die Pumpen werden mit 12V betrieben, was zur Folge hat, dass eine 12V Speisung implementiert werden musste. Dazu wird ein Schaltspannungsregler verwendet. Dieser wandelt mittels Step-Down Prinzip die 48V des Netzteils in eine Konstantspannungsquelle von 12V. Es handelt sich hierbei um einen Regler von Monolithic Power Systems. Genauer gesagt um den MP24943DN-LF. Die Auswahl ist auf dieses Bauteil gefallen, da mit 48V eine relativ hohe Eingangsspannung verarbeitet werden muss. Der MP24943DN-LF kann am Eingang mit Spannungen von 4.5-55V arbeiten und dabei eine Ausgangsspannung von 0.8-45V erzeugen. Dies bei einem maximalen Strom von bis zu 3A. Die Realisierung der 12V Speisung kann in Abbildung 5.2 betrachtet werden.

### Schema

Das Schema in Abbildung 5.2 kann in fünf Teile unterteilt werden. Da wäre zuerst der Eingangsfilter, welcher mit C32, C34 & C36 realisiert ist. Dieser Eingangsfilter wird gefolgt von einem Spannungsteiler, welcher den Enable auf aktiv setzt. Der eigentliche Regler wird mittels des IC7, D6 & L3 realisiert. Mittels zweier Spannungsteiler, wird die gewünschte Ausgangsspannung, sowie die Overvoltage-Protection eingestellt. Vor dem Ausgang der Schaltung ist dann erneut eine Filterstufe implementiert, welche das Ausgangssignal glättet.

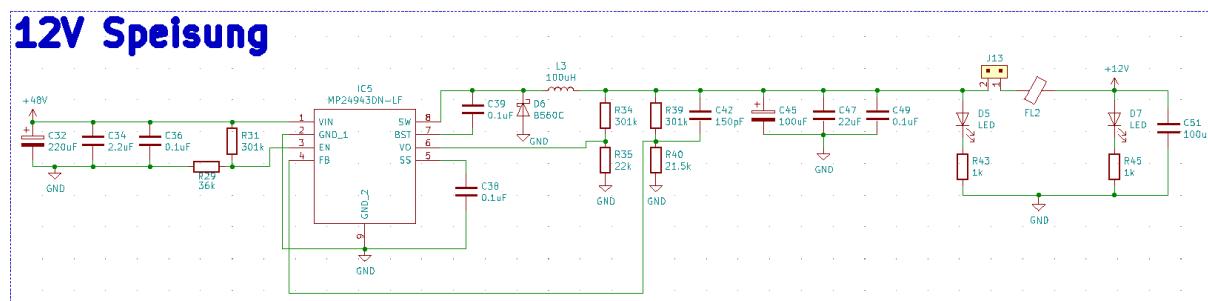


Abbildung 5.2: Schema der 12V Speisung

### Funktionsbeschrieb der Schaltung

Um den MP24943DN-LF auf aktiv zu setzen, wird eine minimale Spannung von 1.8V vorausgesetzt. Fällt diese unter 0.4V, so wird dieser auf inaktiv gesetzt. Damit der Spannungsregler immer eingeschaltet ist, wird mittels zweier Widerstände R29 & R31 ein Spannungsteiler realisiert, welcher den Enable (EN) Pin auf 5V und somit auf aktiv setzt. Dieser Spannungsteiler musste implementiert werden, da alle Eingangspins außer dem  $V_{in}$  einen maximalen Eingangspiegel von 6.5V verkraften können.

Die gewünschte Ausgangsspannung wird mittels Spannungsteiler R39 & R40 eingestellt, welche auf den Feedback Eingang (FB) rückgekoppelt werden. Diese berechnet sich laut Datenblatt gemäß Formel 5.1.

$$R40 = \frac{R39}{\frac{V_{out}}{0.8} - 1} \quad (5.1)$$

Bei einem Widerstandsverhältnis von  $R39=301\text{k}\Omega$  &  $R40=21.5\text{k}\Omega$  entspricht dies einer Ausgangsspannung von 12V.

Um einer Überspannung vorbeugen zu können, wird am Eingang Voltage-Overshoot (VO) ein Spannungsteiler implementiert. Diese wird am VO-Eingang mit einer Referenzspannung von 0.9V verglichen. Übersteigt die Spannung an VO die Referenzspannung von 0.9V, so wird der Regler ausgeschaltet, bis die Spannung wieder unter 0.9V fällt. Als maximale Ausgangsspannung wurde hierbei eine Spannung von 13V gewählt. Diese Wahl wurde getroffen, da die 12V ausschliesslich für die Ansteuerung der Pumpen verwendet wird und diese eine Spannung von 13V verkraften können ohne Schaden zu nehmen. Der Spannungsteiler wird gemäss Datenblatt mit der Formel 5.2 berechnet.

$$R35 = \frac{R34}{\frac{V_{ovp}}{V_{ovref}} - 1} \quad (5.2)$$

Bei einem Widerstandsverhältnis von  $R34=301\text{k}\Omega$  &  $R35=22\text{k}\Omega$  entspricht dies einer Überspannungsschutzschwelle von 13.21V.

Der Rippel des Spulenstroms lässt sich gemäss Formel 5.3 berechnen. Dieser sollte gemäss Datenblatt ca. 30% des maximalen Ausgangsstroms von 3A betragen.

$$L3 = \frac{V_{out} \cdot (V_{in} - V_{out})}{V_{in} \cdot \Delta I_L \cdot f_{osc}} \quad (5.3)$$

Der interne Oszillator läuft dabei bei einer Frequenz von 100kHz. Bei der ausgewählten Spule von  $100\mu\text{H}$  erhalten wir ein  $\Delta I_L$  von 0.9A. Ausserdem wird im Datenblatt darauf hingewiesen, dass die gewählte Spule auf mindestens 125% des maximalen Ausgangsstroms von 3A ausgelegt werden soll. Auch der Gleichstromwiderstand der Spule sollte  $\leq 200\text{m}\Omega$  sein.

Mit den Kondensatoren C45, C47 & C49 wird die Ausgangsspannung zum Abschluss noch geglättet. Bei den Eingangskondensatoren, sowie den Ausgangskondensatoren sollte es sich um low ESR Typen handeln. Um die restlichen hochfrequenten Störungen herauszufiltern, ist zum Abschluss ein Ferrit implementiert worden.

Um die Schaltung einfacher in Betrieb nehmen zu können, wurde mittels Jumper sichergestellt, dass die Speisung vom System abgekoppelt werden kann. Ein LED zeigt an, ob die Speisung funktioniert oder nicht. So ein LED ist jeweils vor und nach dem Jumper implementiert worden. Der Ferrit FL2 soll noch letzte Störungen herausfiltern.

### 5.1.3 5V Speisung

Der Mikrocontroller, sowie die Durchflussmessgeräte und das Display werden mit 5V betrieben. Aus diesem Grund wurde eine 5V Speisung implementiert. Dazu wird der selbe Schaltspannungsregler wie bei der 12V Speisung in Kapitel 5.1.2 verwendet. Die Realisierung der 5V Speisung kann in Abbildung 5.3 betrachtet werden.

## Schema

Das Schema in Abbildung 5.3 kann wie bei der 12V Speisung gemäss Kapitel 5.1.2 in fünf Teile unterteilt werden. Da wäre zuerst der Eingangsfilter, welcher mit C31, C33 & C35 realisiert ist. Dieser Eingangsfilter wird wiederum gefolgt von einem Spannungsteiler, welcher den Enable auf aktiv setzt. Der eigentliche Regler wird auch hier mittels des IC6, D5 & L4 realisiert. Mittels zweier Spannungsteiler, wird die gewünschte Ausgangsspannung, sowie die Übervoltageschutz eingestellt. Vor dem Ausgang der Schaltung ist dann erneut eine Filterstufe implementiert, welche das Ausgangssignal glättet.

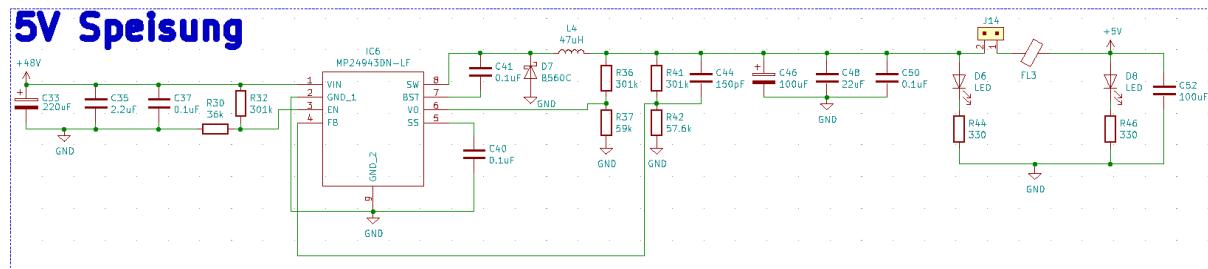


Abbildung 5.3: Schema der 5V Speisung

## Funktionsbeschrieb der Schaltung

Auch bei der 5V Speisung wurde mittels R29 & R31 ein Spannungsteiler realisiert, welcher das IC gemäss Kapitel 5.1.2 auf aktiv setzt.

Das Widerstandsverhältnis von R41 & R42, welches die Ausgangsspannung definiert, wurde gemäss Formel 5.1 berechnet. Somit ergeben sich für  $R41=301\text{k}\Omega$  und für  $R42=57.6\text{k}\Omega$ , Was einer Ausgangsspannung von 4.98V entspricht.

Beim Überspannungsschutz musste darauf geachtet werden, dass der Mikrokontroller AtMega2560-16AU nur in einem Spannungsbereich von 4.5V-5.5V betrieben werden darf. Die maximal verträgliche Eingangsspannung liegt laut Datenblatt bei 6V. Somit muss der Überspannungsschutz so gestaltet werden, dass die Schwelle von 6V nicht überschritten werden kann. Um dies erreichen zu können, wurde für  $R36=301\text{k}\Omega$  und  $R37=53\text{k}\Omega$  gewählt. Gemäss Formel 5.2 erhält man so eine Überspannungsschutzwelle von 6V.

Der interne Oszillator läuft wiederum bei einer Frequenz von 100kHz. Bei der ausgewählten Spule L4 von  $47\mu\text{H}$  erhält man mittels Formel 5.3 ein  $\Delta I_L$  von 0.953A. Auch hier gilt gemäss Datenblatt, dass die gewählte Spule auf mindestens 125% des maximalen Ausgangsstroms von 3A ausgelegt werden soll. Auch der Gleichstromwiderstand der Spule sollte  $\leq 200\text{m}\Omega$  sein.

Mit den Kondensatoren C46, C48 & C50 wird die Ausgangsspannung zum Abschluss auch noch geglättet. Bei den Eingangskondensatoren, sowie den Ausgangskondensatoren sollte es sich um low ESR Typen handeln. Auch hier wurde noch zum Abschluss ein Ferrit implementiert, welcher allfällige hochfrequente Störungen herausfiltern soll.

Auch bei der 5V Speisung wurde ein Jumper zu Testzwecken und zwei LED's implementiert. Außerdem findet sich auch hier wieder ein Ferrit FL3, welcher letzte Störungen beseitigen soll.

### 5.1.4 3.3V Speisung

Um die Treiber der Motorenansteuerung, das Wirelessmodul und die RFID-Schaltung betreiben zu können, wird zusätzlich eine 3,3V Speisung verbaut. Da es sich dabei nicht um enorm Leistungstreibende Elemente handelt, wurde entschieden einen einfachen Linearregler einzusetzen, welcher von der 5V Speisung aus betrieben wird.

#### Schema

Bei dem Linearregler handelt es sich konkret um den LF33CDT-TRY von STMicroelectronics. Dieser hat eine fixe Ausgangsspannung von 3,3V, bei einem maximalen Strom von 1A. Das dazugehörige Schama kann in Abbildung 5.4 begutachtet werden.

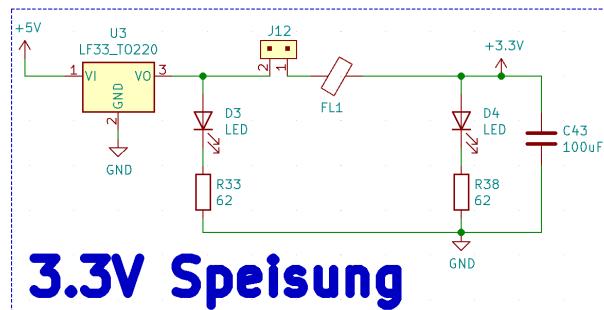


Abbildung 5.4: Schema der 3.3V Speisung

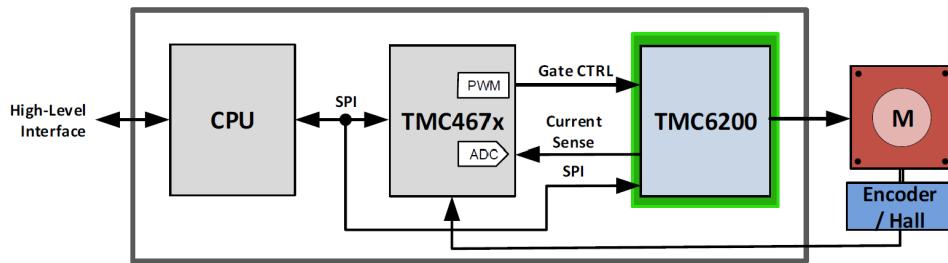
#### Funktionsbeschrieb der Schaltung

Der Linearregler benötigt keine spezielle Beschaltung. Er wird lediglich an die 5V Speisung angeschlossen. Am Eingang und am Ausgang ist ein Filterkondensator implementiert.

Wie bei der 12V Speisung in Kapitel 5.1.2 und der 5V Speisung in 5.1.3 wurde ein Jumper, sowie zwei LED's zu Testzwecken implementiert und auch hier Filtert ein Ferrit FL1 letzte Störungen heraus.

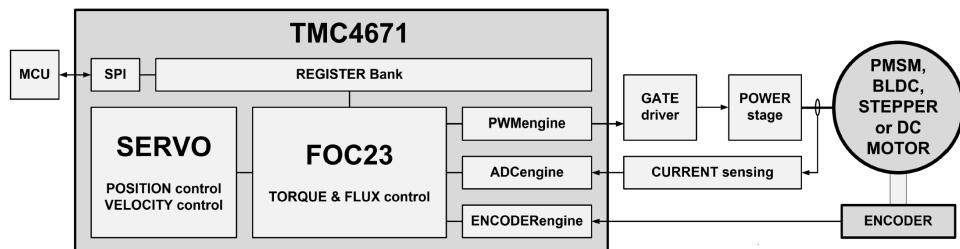
## 5.2 Motor

Um ein Glas während der Zubereitung hin- und her zu bewegen, wird eine Antriebsgruppe benötigt. Die Auswahl der Motorengruppe wurde mehr oder weniger vom Dozenten vorgegeben. Der Entscheid fiel dabei auf den Brushless DC-Motor AKM22h von Sigmatec. Auch dessen Ansteuerung ergab sich durch die schon vorhandenen EVAL-Boards mit TMC4671 und UPS 10A70V. Das benötigte Feedback wird vom ABN-Encoder AMT33 von CUI devices verwendet. Auf die erwähnten Komponenten wird im Folgenden eingegangen. Abbildung 5.5 zeigt, wie die ICs zusammenhängen.



**Abbildung 5.5:** Blockschaltbild Konfiguration IC's mit BLDC und Encoder

Es wurde darauf geachtet, dass der Aufbau des Prints so gut wie möglich dem Testaufbau entspricht. Im Folgenden wird deshalb ein detaillierteres Blockschaltbild gezeigt, welches den Aufbau eher nach Funktionen beschreibt, siehe dazu Abbildung 5.6. Darin eingezeichnet sind die verwendeten Referenceboards.



**Abbildung 5.6:** Blockschaltbild Konfiguration Motorengruppe nach Funktionen.

TMC4671	= Trinamic TMC4671	FOC-Treiber
GATE driver	= Trinamic TMC6200	Gate-Treiber
POWER stage	= Trinamic UPS 10A70V	H-Brücke
Current sensing	= UPS 10A70V ==> TMC6200	Messung Phasenströme
PMSM/BLDC	= Sigmatec AKM22h	BLDC
ENCODER	= CUI devices ATS33	ABN-Encoder

### 5.2.1 BLDC

Ein BLDC zeichnet sich dadurch aus, dass der Rotor mit Permanentmagneten bestückt ist. Somit ähnelt er vom Aufbau her einer Synchronmaschine mit permanenterregten Rotorwicklungen. Zur Ansteuerung hat der BLDC drei Phasenleitungen, welche auf die Spulen führen. Die Spulen sind innerhalb des BLDC in Stern geschaltet. Der Motor hat drei Polpaare, womit die magnetische Winkelgeschwindigkeit drei Mal schneller ist als die mechanische.

Ein Nachteil von BLDC-Motoren ist, dass bei Drehgeschwindigkeiten über Nenndrehzahl nur mit einer geeigneten Regelung der Statorwicklungen in Feldschwächung gefahren werden kann, anstelle dass der Erregerstrom verringert wird. Ansonsten zeichnet sich ein BLDC durch ein besseres Leistungs-/Gewichtsverhältnis aus als herkömmliche Motoren.

#### Schaltungsaufbau

Der Motor wird direkt auf die vorgesehenen Anschluss-Pins geführt. Obwohl die Versorgungsspannung unterhalb der maximalen Berührungsspannung liegt wurde im Falle eines Defekts im Motor zur Personensicherheit ein Pin für die Erdung des Motorengehäuses vorgesehen.

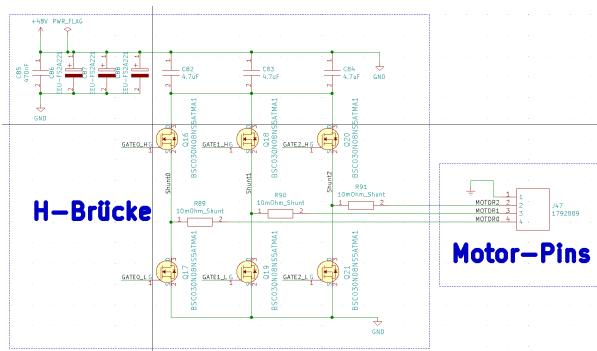


Abbildung 5.7: H-Brücke und Motor-Pins für BLDC

#### Funktionsbeschrieb der Schaltung

Hier gibt es nicht viel zu beschreiben, der Anschluss des Motors ist selbsterklärend. Lediglich die Phasenfolge ist zu beachten, damit das Drehfeld stimmt.

Der Motor an sich lässt sich wie folgt beschreiben:

Nennnetzspannung	$V_M$	=	48	[V]
Stillstanddrehmoment	$M_0$	=	0.88	[Nm]
Nenndrehmoment	$M_n$	=	0.85	[Nm]
Spitzendrehmoment	$M_{0max}$	=	2.8	[Nm]
Nenndrehzahl	$n_n$	=	1500	[min <sup>-1</sup> ]
Nennleistung	$P_n$	=	0.13	[kW]
Stillstandstrom	$I_0$	=	5.41	[A]
Nennstrom	$I_N$	=	5.21	[A]
Spitzenstrom	$I_{max}$	=	21.6	[A]
Drehmomentkonstante	$k_T$	=	0.1632	[Nm/A]
Rotorträgheitsmoment	$J$	=	0.16	[kg·cm <sup>2</sup> ]
Gewicht	$m$	=	1.1	[kg]

### 5.2.2 H-Brücke

Das Bindeglied zwischen Kraft (Motor) und Steuersignalen wird von der H-Brücke gebildet. Durch Laden-/Entladen der MOSFET-Gates wird eine Spannung an einer Spule angelegt oder weggenommen. Wie sich das Ein- und Ausschalten der einzelnen Phasenbrücken auf den Motor auswirkt, wird zu einem späteren Zeitpunkt erklärt.

#### Schaltungsaufbau

Für den Aufbau und die Dimensionierung wurde das Referenzschema des Evaluationsboard verwendet, mit dem getestet wurde (Trinamic UPS 10A70V). Es ist aufgezeigt im Anhang Kapitel C Abbildung C.1. Die Dimensionierung der Shunts wird in Kapitel 5.2.5 abgehandelt.

Der Schaltungsaufbau ergibt sich durch den dreiphasigen Aufbau des BLDCs. Es werden so drei Stränge gebildet, woran jeweils eine Spule verbunden wird. Der Energiefluss führt dabei über einen Strommesswiderstand. Die Eingänge der H-Brücke werden zusätzlich mit Stütz- und Filterkondensatoren bestückt, um eine saubere Netzspannung zu gewährleisten.

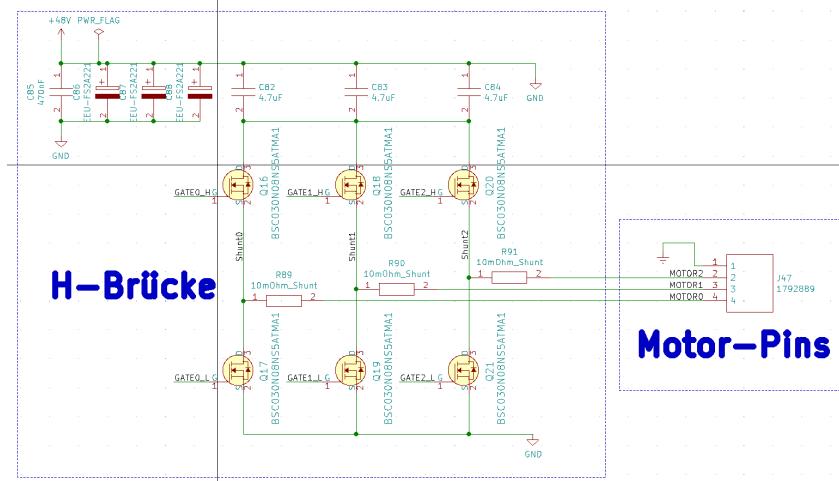


Abbildung 5.8: H-Brücke.

#### Funktionsbeschrieb der Schaltung

Wie erwähnt werden die Eingänge gefiltert. Dies geschieht mit den Kondensatoren C85-C88, um die Eingangsspannung konstant zu halten und hochfrequente Störspannungen nicht ins Netz gespeist werden. Bei C86-C88 handelt sich um Low-ESR Stützkondensatoren, jeweils ein Kondensator pro Phase. Sie befinden sich nahe am Spannungseingang und der H-Brücke.

Die Kondensatoren C82-C84 sind Kondensatoren, welche direkt zwischen Ein- und Ausgang eines H-Brücken-Strangs platziert wurden. Sie dienen auch zu Filterzwecken.

Die MOSFETS Q16-Q21 bilden die eigentliche H-Brücke. Sie Schalten den Leistungsfluss gemäss den Ansteuersignalen. Sie können 100A Nennstrom schalten und 100V standhalten. Die Gate-Source-Spannung beträgt  $\pm 20\text{V}$ .

Referenzierung  
auf mög-  
liches  
Kapitel  
FOC-  
Steuerung  
mit BLDC-  
Motor /  
Regleraus-  
legung

### 5.2.3 ABN-Encoder

Der ABN-Encoder wird verwendet, um dem FOC-Regler die momentane Lage des Rotors mitzuteilen. Dazu wurde der AMT33 im Verlaufe des Projektes 5 ausgesucht und hier beschrieben. Er ist unempfindlich auf Staub, Schmutz und Öl. Die Montage und Zentrierung gestaltet sich sehr einfach. Außerdem hat er eine einstellbar hohe Genauigkeit.

#### Schaltungsaufbau

Nebst der 5V-Spannungsversorgung sind die Signalleitungen auf den Encoder geführt. Nämlich A, B und N. Während A und B die Codierung für die relative Wegänderung sind, gibt N an, sobald eine gesamte Umdrehung erfolgte. Auf die Schaltung innerhalb des Encoders wird nicht eingegangen.

#### Funktionsbeschrieb der Schaltung

Der ABN-Encoder kann wie gesagt eine einstellbar genaue Genauigkeit ausgeben. Dies zeigt sich in einer Form der Bitanzahl pro Umdrehung. Die maximale Auflösung liegt bei 4096 Schaltvorgänge pro Umdrehung, was einer Auflösung von 12 Bit entspricht. Dies ist wichtig für die Implementierung in den FOC-Treiber. Die Ausgangspins sind normale Header-Pins. Der Spannungsausgang wurde mit einem Stützkondensator C89 versehen.

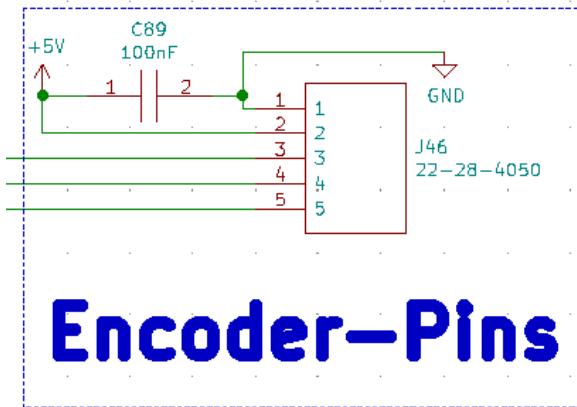


Abbildung 5.9: Schema ABN-Encoder.

### 5.2.4 FOC-Treiber TMC4671

Wie dem Grobkonzept in Kapitel 2.1 entnommen werden kann, wird für die Ansteuerung des BLDC-Motors eine Steuerlogik (Motor Control Unit) benötigt. Dabei handelt es sich um den TMC4671. Der FOC-Treiber berechnet den Modulationsindex für die H-Brücke anhand des Drehmoments, der Geschwindigkeit oder der Position, welche der Mikrocontroller dem FOC-Treiber vorgibt. Die vom FOC-Treiber ausgehenden PWM-Leitungen gehen direkt auf den Gate-Treiber, welcher dann wiederum die MOSFETs ansteuert. Als Feedback benötigt der FOC-Treiber zwei Phasenströme. Der dritte Strom wird mit dem Knotengesetz berechnet. Da der BLDC in Stern geschaltet ist gilt:

$$I_U + I_V + I_W = 0 \quad (5.4)$$

Auf den FOC-Treiber und weitere für diesen Schaltkreis benötigte Komponenten wird im folgenden eingegangen. Da das Gehäuse des IC's nur schwierig lötbar ist, wird ein Breakout-Board (BOB) verwendet. Auf dem Board sind die Eingänge des FOC-Treibers schon geschützt mit Filtern und Überspannungsdioden. Somit entfällt eine Dimensionierung der Filter und Schutzschaltungen. Es wird jedoch die jeweilige Funktion der Schaltung beschrieben.

#### Schema

Auf dem Breakout-Board sind diverse Anschlussmöglichkeiten vorhanden. Für den Cocktailmixer werden folgende Schnittstellen verwendet:

- SPI Input
- Phasenströme Input
- Encoder Input
- Motorspannung Input
- Steuersignale Output

Im Anhang A zeigt Abbildung A.2 anhand eines Blockdiagramms, welche Funktionen im Treiber vorhanden sind und wie diese zusammenhängen. Im wesentlichen kann daraus entnommen werden, dass der TMC4671 aus einer FOC-Logik<sup>4</sup>, einer Servo-Logik, einem SPI-Interface und diversen Engines (PWM, ADC, Encoder) besteht. In Abbildung A.1 wird eine Standard-Anwendungsschaltung gezeigt, worin erkennbar ist, dass der Treiber sehr universell aufgebaut ist und für mehrere Motorentypen verwendet werden können.

In Abbildung 5.10 ist erkennbar, welche Pins für den Cocktailmixer verwendet werden und welche Verbindungsleitungen zum Treiber führen. Es handelt sich dabei um die schon aufgelisteten Komponenten, nämlich Kommunikationsleitung zwischen Treiber und Mikrocontroller (SPI), Phasenströme (ADC), Encoder Input (ENC), Motorspannung (48V) und das PWM-Signal (PWM). Bei den SPI-Leitungen ist zu sehen, dass die Leitungen zuerst über einen Level-Shifter müssen, da der Treiber nur 3.3V verarbeiten kann ohne dabei kaputt zu gehen. Zu erkennen an der zusätzlichen Leitungsbezeichnung LV (Lower Voltage). Die Leitung SPI1 geht vom Treiber zum Mikrocontroller und muss deshalb nicht geshifftet werden.

---

<sup>4</sup>FOC= Field Oriented Control

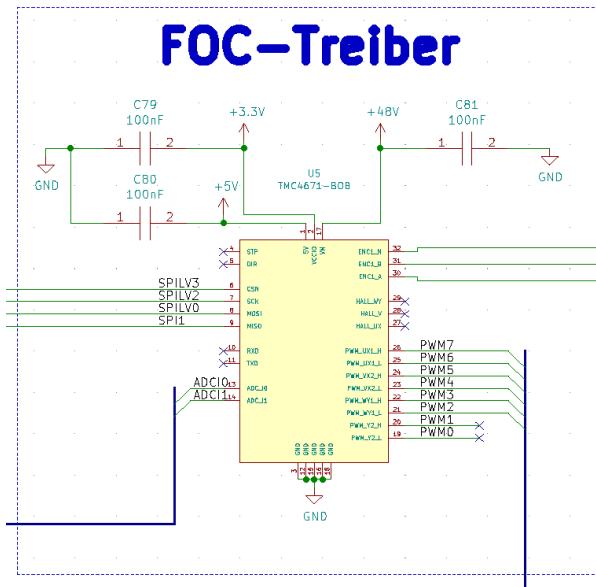


Abbildung 5.10: Schema FOC-Treiber.

### Funktionsbeschrieb der Schaltung

Im Folgenden werden kurz die Teilsysteme auf dem TMC4671-BOB beschrieben, welche für den Cocktailmixer gebraucht werden. Dazu wird auf das Datenblatt von Trinamic TMC4671 zurückgegriffen.

**Kommunikation Input (SPI)** Der Kommunikationseingang ist nicht speziell geschützt. Hier ist für das eigene Layout darauf zu achten, dass die angelegte Spannung nicht über 3.3V steigt. Dies wird mit einem Level-Shifter zwischen Mikrocontroller und TMC4671 gewährleistet.

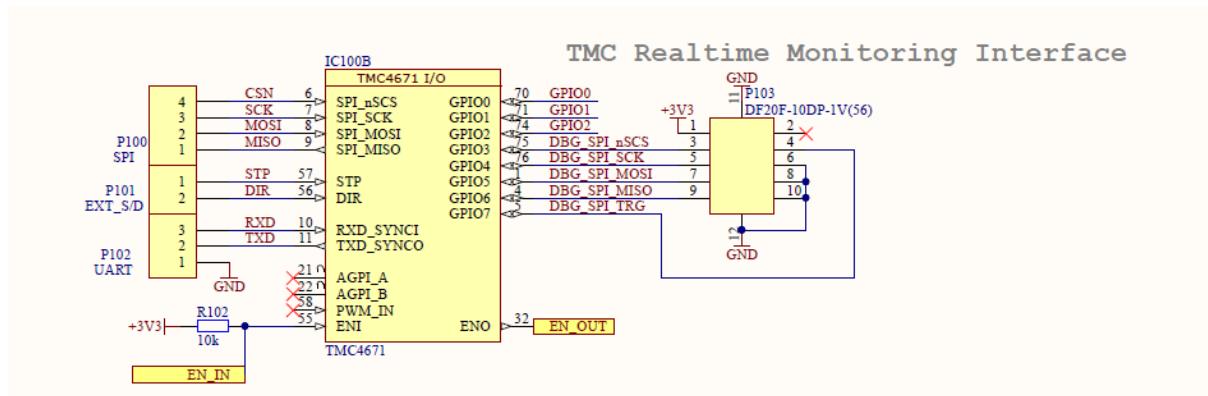


Abbildung 5.11: SPI-Input des TMC4671-BOB.

**Ströme Input (ADC)** Der Eingang zur Strommessung ist mit einem Tiefpass geschützt. Dieser hat die Zeitkonstante:

$$\tau = R308 \cdot C300 = 100\Omega \cdot 100pF = 10ns \quad (5.5)$$

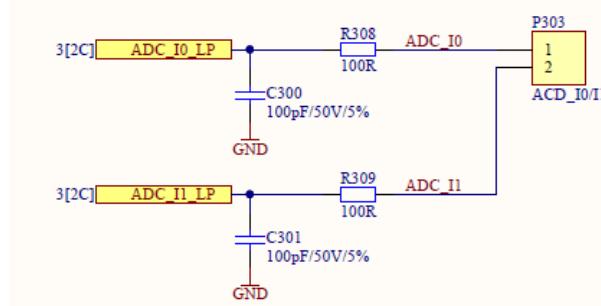


Abbildung 5.12: Phasenstroeme-Input des TMC4671-BOB.

**Encoder Input (ENC)** Der Encoder-Input ist ziemlich clever aufgebaut. Um Überspannungen zu verhindern, wird ein Schmitt-Trigger mit eingebautem Level-Shifter verwendet. Der Level-Shifter verhindert dabei ein Ansteigen der Eingangsspannung über dessen Eingangsspannung. Zudem wird mit dem Widerstandsnetzwerk ein entprellen der Encoder-Signale erreicht. Wird der Input auf 0V gezogen, so entlädt sich der Kondensator über die Widerstände R200-R202 mit der Zeitkonstante:

$$\tau = R200 \cdot C204 = 1k\Omega \cdot 100pF = 100ns \quad (5.6)$$

Fällt die Spannung über dem Kondensator während dem Entladen unter einen bestimmten Wert, so wird auch der Schmitt-Trigger aktiv und springt auf 0V. Sobald der Eingang nicht mehr vom Encoder auf 0V gezogen wird, so lädt sich der Kondensator über die Widerstände R200-R202 sowie R206,R208,R210. Somit ergibt sich eine längere Zeitkonstante:

$$\tau = (R200 + R210) \cdot C204 = (1k\Omega + 4.7k\Omega) \cdot 100pF = 570ns \quad (5.7)$$

Jetzt geht es bedeutend länger, bis der Kondensator wieder geladen wird. Sobald die Spannung wieder einen bestimmten Wert erreicht, wird der Schmitt-Trigger aktiv und springt auf Versorgungsspannung (3.3V).

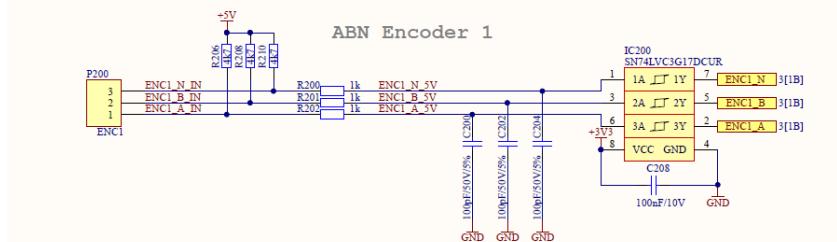


Abbildung 5.13: ABN-Encoder-Input des TMC4671-BOB.

**Motorspannung Input (48V)** Die Motorspannung ist wichtig, um einen Kommuntierungs-vorgang zu berechnen. Wichtiger als die Zeitkonstante ist hier ein Spannungsteiler, welcher die Motorspannung auf unter 3.3V bringt. Dazu wird folgende Formel angewendet:

$$U_{TMC} = U_M \cdot \frac{R310}{R310 + R311} = 48V \cdot \frac{1k\Omega}{1k\Omega + 100k\Omega} = 0.7V \quad (5.8)$$

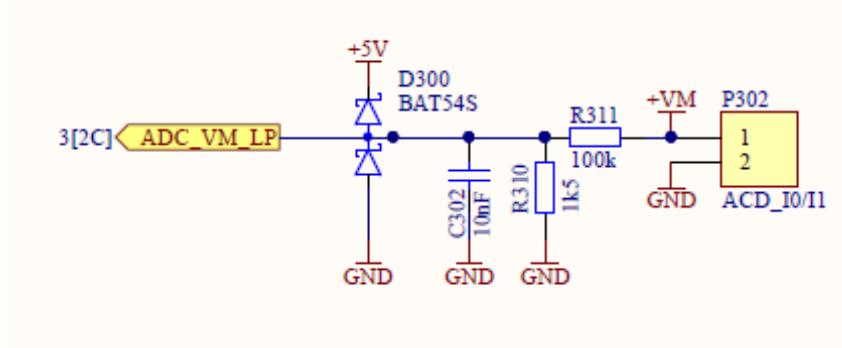


Abbildung 5.14: Motorspannung-Input des TMC4671-BOB.

**Steuersignale Output (PWM)** Die Ausgangssignale für den Gate-Treiber gehen direkt auf die Header-Pins des MC4671-BOB. Sie werden nicht geschützt.

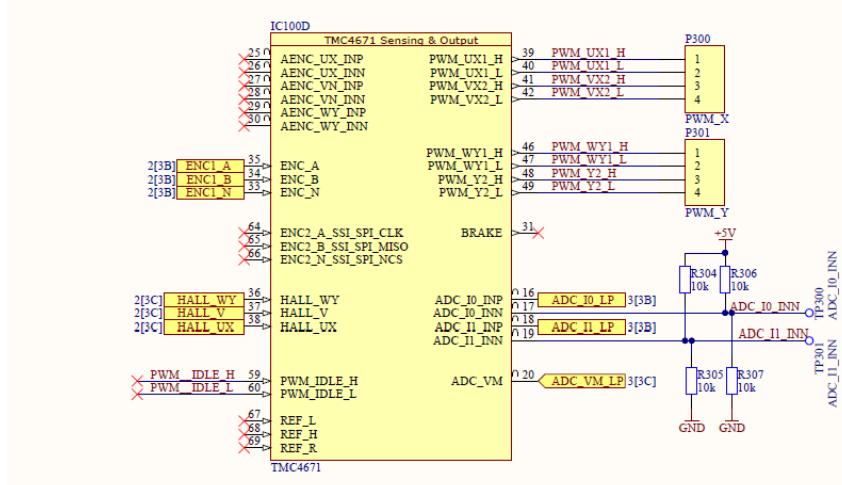


Abbildung 5.15: PWM-Output des TMC4671-BOB.

### 5.2.5 Gate-Treiber

Der Motor wird über eine H-Brücke gesteuert. Dies bedingt pro Spule zwei MOSFET's, um diese entsprechend magnetisieren zu können. Um einen MOSFET in einen leitenden Zustand zu bringen, muss das Gate des MOSFET's mit einer elektrischen Ladung gefüllt werden. Während diesem Vorgang hat am Gate ein kapazitives Verhalten, was bedeutet, dass bei jedem Schaltvorgang Ströme fliessen. Damit die Umschaltverluste und daraus folgende Abwärme verhindert werden kann, ist es vorteilhaft die Gates so schnell wie möglich laden und entladen zu können. Da kommt der Gate-Treiber ins Spiel. Dieser ladet und entladet das Gate schnell genug und stellt die dazu benötigte Energie für das Gate zur Verfügung.

### Schema

Damit ein komplizierter Aufbau vermieden werden kann und einige Zusatzfunktionen wie Strommessung, Messverstärkung etc. angeboten werden können, wurde während des Entwicklungsprozesses ein Gate-Treiber von Trinamic ausgewählt. Das entsprechende Bauteil ist der TMC6200. Das Blockdiagramm und eine Beispielschaltung befinden sich im Anhang Kapitel B Abbildung B.1 und Abbildung B.2. Das Blockdiagramm zeigt, dass der TMC6200 aus einer Treiber-Logik für den Motor, einem SPI-/Pinsettings-Interface, einer Diagnosenlogik, Strommessschaltung und diversen unterstützenden Schaltungen wie Spannungsversorgung besteht.

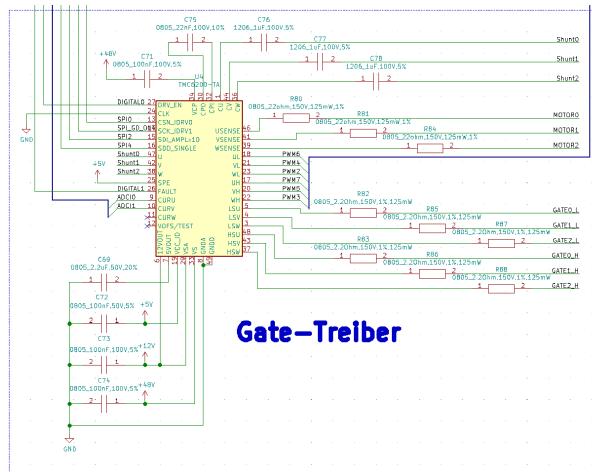


Abbildung 5.16: Schema Gate-Treiber.

### Funktionsbeschrieb der Schaltung

**Shunt-Widerstände  $R_{Sense}$**  Die Shunt-Widerstände ermöglichen einen geschlossenen Loop für die Stromsteuerung, so wie es für eine feldorientierte Steuerung (FOC) benötigt wird. Dazu wird solch ein Shunt in Serie mit der Spule des BLDC-Motors geschaltet (siehe R89 - R91, Abb. 5.8). Durch den fliessenden Strom liegt eine Spannung über dem Shunt, welche dann vom TMC6200 verstärkt wird und aufbereitet an den Kontroll-Chip TMC4671 ausgegeben wird. Die Verstärkung sowie Offset auf das Signal sind programmierbar.

Die Dimensionierung der Widerstände und die darauf folgende Programmierung des TMC6200 sind schon von Trinamic ermittelt worden. Es wird empfohlen, die Widerstände nach dem Maximalstrom des Motors auszuwählen. Im Falle des AKM22h sind dies 5A. Darüber hinaus wird

Kondensatoren  
im Text  
erwähnen

empfohlen, eine Überdimensionierung von 25-50% vorzunehmen. Aus Qualitätsgründen wird jedoch von 10A ausgegangen, was die Zuverlässigkeit und Lebensdauer erhöhen soll, zudem wird so die Verlustleistung des Shunts kleiner gehalten als bei einer Grösse ober-/unterhalb. Eine von Trinamic erstellte Tabelle (siehe Anhang B Tabelle B.3) ergibt folgende Grösse für die Shunt-Widerstände: [trinamic\\_tmc6200\\_datasheet\\_2013](#)

$$R_{Sense} = 10m\Omega$$

$$\text{Verstärkungsfaktor} = 10x$$

**Gate Vorwiderstände  $R_{Gate}$**  Da das Gate ein kapazitives Verhalten zeigt, ist der Strom, welcher zu Beginn ins Gate fliesst, sehr hoch. Der Gate-Vorwiderstand begrenzt diesen, um das Gate und den Pin des TMC6200 vor Überströmen zu schützen (R82 - R83 und R85 - R88, Abb. 5.16).

Die Dimensionierung der Gate-Widerständen sollte grundsätzlich an die MOSFET Gate-Drain-Ladung (Miller charge) angelehnt werden, um an angemessene Anstiegszeiten während des Schaltens zu erreichen. Auch in diesem Falle wurden von Trinamic schon einige Parameter ermittelt und im Anhang B in Tabelle B.4 dargestellt. Die Gate-Ladung des ausgewählten MOSFET's beträgt 61nC. Mit dem programmierbaren Register DRV\_STRENGTH wird der Strom ins Gate angepasst. [trinamic\\_tmc6200\\_datasheet\\_2013](#)

$$R_{Gate} = R_{Gate} \leq 2.5\Omega$$

$$DRV\_STRENGTH = 1 \text{ bis } 3$$

**Schutzwiderstände Messeingang  $R_{Protect}$**  Wird von einem High-Zustand in einen Low-Zustand gewechselt, kann aufgrund von Induktivitäten der Shunts oder deren Verbindungen die Spannung unterschiessen. Der Schutzwiderstand schützt den Messeingang des TMC6200 vor diesem Effekt (R80, R81 und R84, Abb. 5.16).

Die Dimensionierung dieses Widerstands wird im Datenblatt mit einem Widerstandswert zwischen  $10\Omega$  und  $22\Omega$  angegeben. [trinamic\\_tmc6200\\_datasheet\\_2013](#)

$$R_P = 10\Omega \text{ bis } 22\Omega$$

**Bootstrap Kondensatoren  $C_{Bootstrap}$**  Die Bootstrap-Kondensator werden benutzt, wenn eine Änderung des Potentials innert kurzer Zeit benötigt wird (C76 - C78, Abb. 5.16).

Die Dimensionierung dieses Widerstands wird im Datenblatt mit einem Kapazitätswert zwischen  $470nF$  und  $1\mu F$  angegeben, bei einer Nennspannung von 16V oder 25V. Weiter gilt gemäss Datenblatt, dass bei MOSFET's mit einem  $Q_G \geq 40nC$  die Gatekapazität  $1\mu F$  sein soll. Da die Kapazität  $61nF$  beträgt, ist dies der Fall. [trinamic\\_tmc6200\\_datasheet\\_2013](#)

$$C_{Bootstrap} = 1\mu F (25V)$$

**Externer Gate-Spannungsregler** Aufgrund der hohen Versorgungsspannung (48V), treten im IC über den Gate- und 5V-Spannungsreglern erhebliche Verluste auf. Um diese Verluste zu reduzieren, wird gemäss Datenblatt [trinamic\\_tmc6200\\_datasheet\\_2013](#) geraten, eine externe Gate-Spannung anzuhängen. Für beste Resultate wird eine Spannung von  $12V \pm 1V$  empfohlen.

Die Dimensionierung der benötigten Kondensatoren ist im Anhang B, aus Abbildung B.5 zu entnehmen.

$$C_{69} = 2.2\mu F$$

$$C_{73} = 100nF$$

### 5.3 Flüssigkeitsbeförderung

Für die Flüssigkeitsbeförderung sind zwei essentielle Komponenten erforderlich. Einerseits muss die Flüssigkeit befördert werden, andererseits muss diese auch dosierbar sein. Diese zwei Hauptaufgaben übernehmen die Pumpen und die Durchflussmessgeräte.

#### 5.3.1 Pumpen

Um die Flüssigkeiten sauber befördern zu können, werden Vakuummembranpumpen eingesetzt. Diese werden oft in der Lebensmittelindustrie verwendet, da diese einerseits hygienisch sind und zum andern einen relativ guten Durchfluss bieten können. Ein weiterer Vorteil dieser Pumpen ist, dass durch die Erstellung eines Vakuums auch Luft gepumpt werden kann. Dies ermöglicht auch einen «Kaltstart» auch ohne Flüssigkeit in den Schläuchen.

#### Schema

Die Schaltung zur Ansteuerung der Pumpen ist relativ simpel aufgebaut und in Abbildung 5.17 zu sehen. Sie besteht aus einer kleinen Leistungsstufe, welche mit einem Mosfet realisiert wurde, welcher die 12V für die Pumpen ein- und ausschaltet. Ausserdem beinhaltet die Schaltung einen Begrenzungswiderstand, eine Freilaufdiode und einen Stecker für den Anschluss der Pumpe. Die Schaltung wurde für alle zwölf Pumpen umgesetzt.

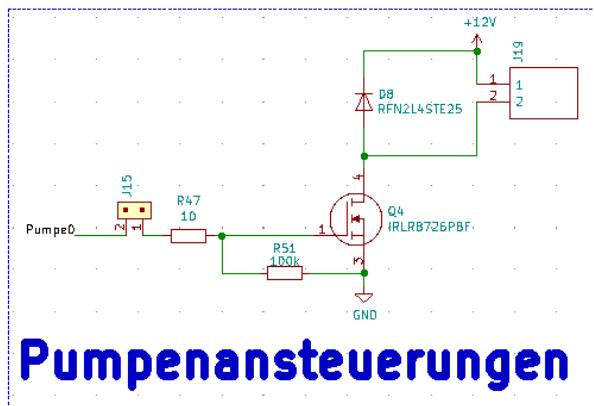


Abbildung 5.17: Schema der Pumpenansteuerung.

#### Funktionsbeschrieb der Schaltung

Die Pumpen werden jeweils direkt über den Mikrocontroller angesteuert. Dabei wird für jede Pumpe ein Digitalpin gemäss 5.7 verwendet. Da der Mikrocontroller nicht genügend Ausgangstrom bietet, um die Pumpen ansteuern zu können, wurde eine kleine Leistungsstufe mit einem Mosfet (hier Q4) implementiert gemäss Abbildung 5.17. Es handelt sich bei diesem Mosfet um einen Logic-Level-Mosfet IRLR8726 von Infineon Technologies. Dieser hat den Vorteil, dass er direkt über einen Mikrocontroller angesteuert werden kann und trotz niedrigem Spannungsniveau voll durchsteuern kann. Da das Gate jedes Mosfet's auch eine Kapazität darstellt, wird der Anfangsstrom beim einschalten des Pin's durch einen 10Ohm (hier R47) Widerstand begrenzt. Ein weiterer Grund für diesen Widerstand ist, dass manche Mosfet's beim ein- und ausschalten, mit der steilen Flanke dazu neigen kurzzeitig zu oszillieren. Dies wird durch diesen Widerstand auch unterdrückt. Zu Messzwecken wurde vor diesem Widerstand ein Jumper implementiert.

Dies ermöglicht es die Ansteuerung vom Mikrocontroller ab zu koppeln. Da es sich bei dieser Pumpe um einem DC-Motor und somit um eine Induktion handelt, wurde parallel zur Pumpe am Stecker eine Freilaufdiode implementiert. Diese schützt den Mosfet vor Spannungsspitzen beim Ausschalten.

### 5.3.2 Durchflussmessgeräte

Um die Flüssigkeiten wie gewünscht abfüllen zu können, müssen diese sauber dosiert werden können. zu diesem Zweck wurden Durchflussmessgeräte eingesetzt. Bei diesen Durchflussmessgeräten handelt es sich um volumetrische Durchflussmessgeräte von Sea, welche bei Durchfluss eine Pulsfolge ausgeben. Diese können dann mittels Elektronik ausgewertet werden.

#### Schema

Das Schema für diese Schaltung ist in Abbildung 5.18 zu sehen. Es beinhaltet lediglich einen Filterkondensator an der Speisung und einen Stecker, an welchem die Durchflussmessgeräte angeschlossen werden können.

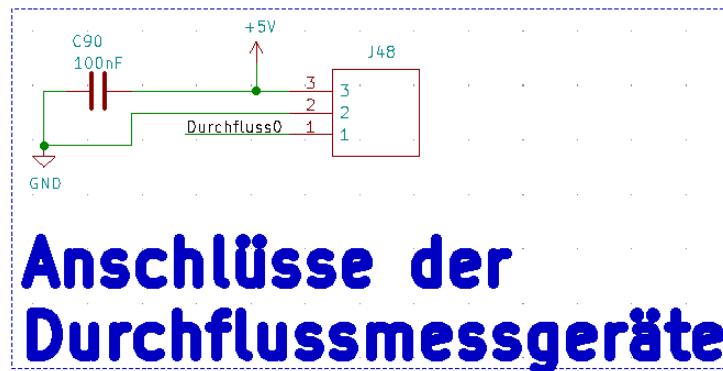


Abbildung 5.18: Schema der Durchflussmessgeräte.

#### Funktionsbeschrieb der Schaltung

Die Durchflussmessgeräte haben drei Anschlüsse. Zum einen ist dies einen Speisungseingang von 5V und eine Groundleitung. Der dritte Pin der Durchflussmessgeräte ist ein Signalpin, welcher bei Durchfluss ein gepulstes Signal mit 5V und 50% Duty-Cycle ausgibt. Da dieses Signal direkt vom Mikrocontroller ausgewertet werden kann, ist keine weitere Elektronik erforderlich. Zur Auswertung der Signale werden gemäss Kapitel 5.7 wiederum Digitalpins als Eingänge verwendet.

## 5.4 Programmierschnittstellen

Die beiden Devices, welche programmiert werden müssen, werden über die USB-Schnittstelle programmiert. Im Folgenden ist in Abbildung 5.19 nur das Schema des Flash-Interfaces für das ESP32 ersichtlich. Es unterscheidet sich zum Flash-Interface des ATMega2560 in der Anzahl Steuerleitungen. Auf das Flash-Interface des Atmega2560 wird nicht eingegangen, da die Funktionen die selben sind.

### 5.4.1 USB-B

Im Folgenden wird der in Kapitel 3.2 erwähnte USB-Anschluss beschrieben. Es handelt sich dabei um einen USB-UART-Converter mit zugehöriger Beschaltung wie z.B. USB-B-Buchse, ESD-Schutz und weitere passive Komponenten.

#### Schema

Auf dem Schema ist die USB-Buchse mit J2 gekennzeichnet. Der dazugehörige ESD-Schutz bildet die Diodenschaltung D2-4. Der Converter hat die Bezeichnung IC2. Die Widerstände R12 und R13 bilden einen Spannungsteiler, welcher an die 5V-Eingangsspannung der Buchse und an den VBUS-Eingang des Converters hängt. Die Widerstände R16, R19, R10 und R11 reduzieren den Kurzschlussstrom zwischen dem USB-UART-Converter und dem ESP32. Der Widerstand R14 ist ein Pull-Up Widerstand, welcher verhindert, dass der Reset-Pin einen ungewünschten Zustand annimmt. Die Widerstände R17 und R18 reduzieren den Strom durch die LED's.

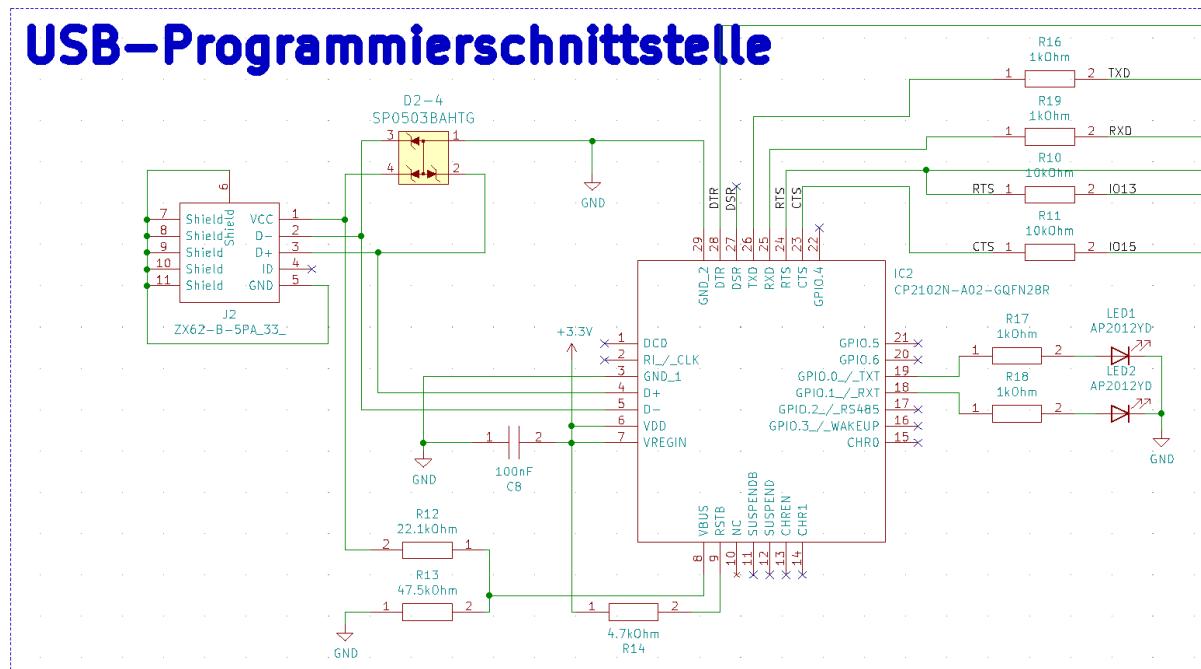


Abbildung 5.19: Schema USB-B.

### Funktionsbeschrieb der Schaltung

Sobald ein USB-Kabel vom PC zum Converter angeschlossen wird, liegt 5V am Spannungsteiler, was vom IC als bestehende Netzwerk-Verbindung interpretiert wird. Die USB-Kommunikation zwischen PC und Converter findet über die Leitungn D+ und D- statt. Aufgrund des differenziellen Verfahrens und Verwendung von verdrillten Drähten werden Störungen von ausserhalb zu einem grossen Teil eliminiert. Der Vorteil der asynchronen serielle Schnittstelle (UART) zeigt sich im Falle der Cocktailmachine vor allem im flexiblen Protokoll und dem einfach zu implementierenden Handshake, welcher benutzt wird, um in den Boot-Modus der Empfänger (ATMega2560 und ESP32) zu gelangen. Die LED's zeigen an, wenn eine Kommunikation stattfindet. Die Widerstände in den Kommunikations- und Steuerleitungen schützen die Pins vor hohen Strömen im Falle von Spannungsunterschieden.

## 5.5 Benutzerschnittstellen

Im Folgenden werden die Benutzerschnittstellen, welche sich auf der Leiterplatine befinden erklärt. Die vorkommenden Interfaces sind mit ihren Funktionen tabellarisch aufgelistet:

Interface	Funktion
Display	Direkte Steuerung zur Cocktailherstellung und Konfiguration der Maschine mit GUI auf dem Display.
ESP32	Indirekte Steuerung zur Cocktailherstellung und Konfiguration der Maschine mit GUI im Browser.
RFID	Abrufen des per Webhost oder Display hinterlegten Getränkes mit Badge. Men genauswahl per Display.

### 5.5.1 Display

Über das Display geschehen die Hauptinteraktionen mit dem Benutzer. Für die Bereitstellung des GUIs wird auf die Software Nextion Editor zurückgegriffen. Über diese Software lassen sich die Seiten, welche im Voraus mit viel Aufwand durchdacht wurden, sehr gut und einfach erstellen. Die Kommunikation, bestehend aus den Page- und Button-IDs, welche bei Drücken der Buttons gesendet werden, findet über UART statt.

#### Schema

Das Schema gestaltet sich eher einfach, da die Logik auf dem Display schon vorhanden ist. Es braucht lediglich einen Stecker, welcher mit J43 beschriftet ist und einen Stützkondensator nahe der Pins am Spannungsausgang.

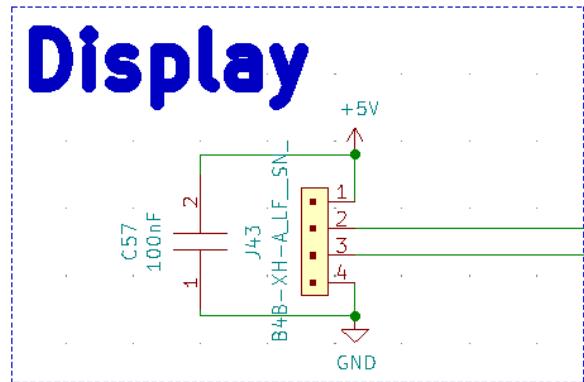


Abbildung 5.20: Schema Display-Stecker.

#### Funktionsbeschrieb der Schaltung

Die mehr oder weniger einzige Funktion, welche im Schema erkennbar ist, wird durch das Stützen der Versorgungsspannung mit dem Kondensator gegeben. Der Stecker ermöglicht den Anschluss des Displays an die Leiterplatine.

### 5.5.2 ESP

Für die Implementierung des WiFi's wird das ESP32 verwendet. Die Hauptfunktion im Schema ist die Kommunikation mit dem Mikrocontroller über die zweite serielle Schnittstelle. Die Ansteuerung zum Schreiben des Programmspeichers muss so gestaltet werden, dass der Boot-Modus automatisch gestartet werden kann, wenn ein Code hochgeladen werden soll. Dies hat eine zusätzliche Schaltung zur Folge.

#### Schema (WiFi-Modul)

In Abbildung 5.21 wird das Schema rund um das ESP32 an sich gezeigt. Es beinhaltet Stütz- und Filterkondensatoren sowie einige Pull-up- und Pull-down-Widerstände, welche verwendet werden, um einen vordefinierten Grundzustand beim Booten des ESP32-Moduls zu erreichen (Strapping-Pins). Weiter gibt es einen Kondensator, welcher dazu da ist, bei gewünschter Zeit in den Boot-Modus zu gelangen.

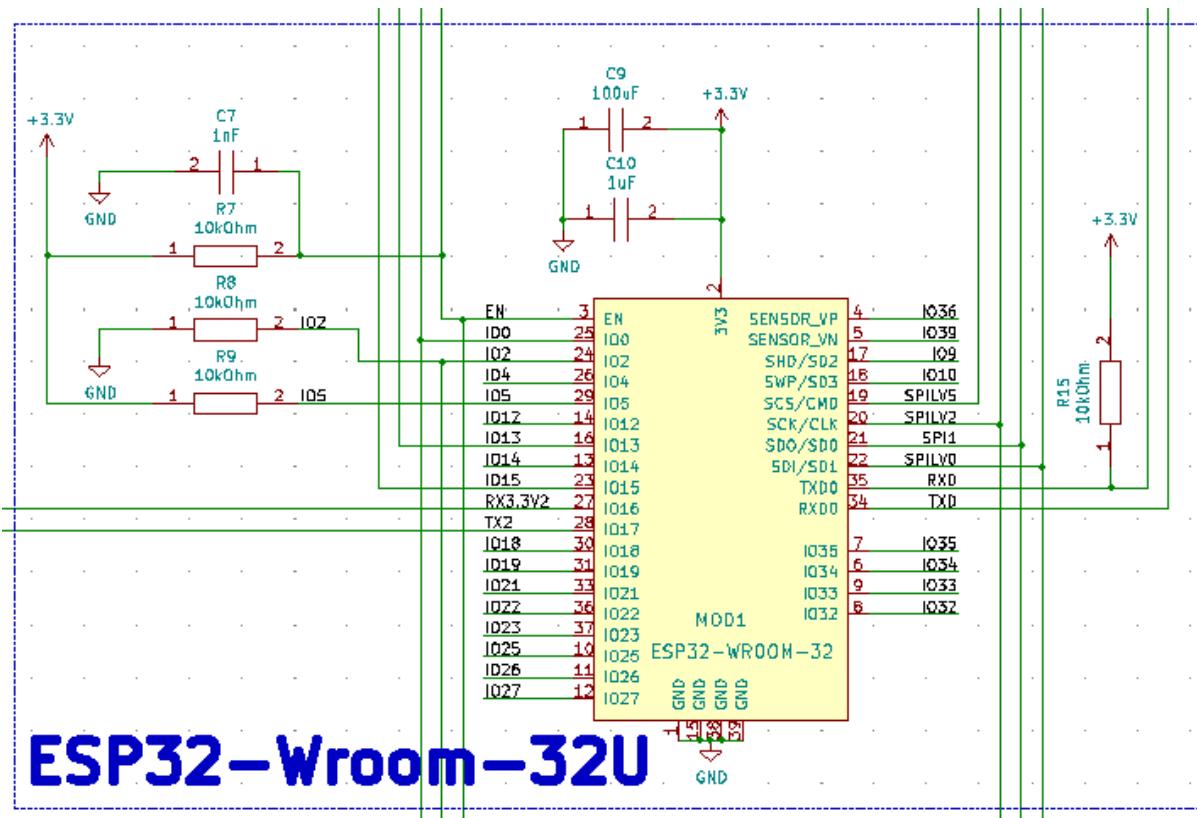


Abbildung 5.21: Schema ESP32-Wroom-32U.

#### Funktionsbeschrieb der Schaltung (WiFi-Modul)

Das ESP32 ist mit MOD1 beschriftet, es übernimmt die in Kapitel 3.1 beschriebenen Funktionen. Die Kondensatoren C9 und C10 dienen zu Stütz- und Filterzwecken am Spannungseingang. Über den EN-Pin wird das Modul ein- und ausgeschaltet (active high). Der Widerstand R7 ist ein Pull-Up für Chip-Enable. Die Widerstände R8, R9, R10 und R11 sind an die Strapping-Pins angeschlossen. Über diese werden beim Aufstarten des ESP32 der Boot-Modus, die Versorgungsspannung von VDD\_SDIO<sup>5</sup>-Slave (Erweiterung der SD-Spezifikationen) und andere Initialisie-

<sup>5</sup>Secure Digital Input Output

rungseinstellungen konfiguriert. Details zu den Konfigurationen sind in Tabelle 3.2 aufgelistet<sup>6</sup>. Der Widerstand R15 zieht U0TXD auf HIGH. Aus Tabelle 3.5 ist ersichtlich, dass dieser für den normalen Boot-Modus auf HIGH sein muss. Für den Download-Boot-Modus hat dieser kein Einfluss. Mit dem Kondensator C7 wird sichergestellt, dass nach einem Reset der Bootmodus gestartet werden kann. Wird der Reset-Pin au LOW gezogen, so entlädt sich der Kondensator. Sobald der Reset-Pin auf HIGH gezogen wird, dauert es länger, bis ein logic High-Zustand am CMOS-Eingang erkannt wird. Wird der Pin IO0 auf LOW gezogen, bevor der Enable-Pin nach einem Reset einen HIGH-Zustand erreicht, so wird das ESP32 in den Download-Boot-Modus gesetzt. Um automatisch in diesen Boot-Modus zu kommen benötigt es eine Logik, welche im Folgenden erklärt wird.

### Schema (Automatische Boot-Logik)

In Abbildung 5.22 wird die Schaltung gezeigt, welche verwendet wird, das ESP32-Modul in den gewünschten Boot-Zustand zu bringen. Für die Beschaltung der automatischen Boot-Logik benötigt es eine Schaltung mit DTR und RTS als Inputs vom USB-UART-Converter her und EN und IO0 als Outputs auf das ESP32. Die Buttons können bei Bedarf verwendet werden, sind für den automatischen Boot-Modus jedoch nicht zwingend nötig. Sie könnten dazu verwendet werden, manuell in den Bootmodus zu gelangen.

Die Widerstände R20 und R21 sind Vorwiderstände an der Basis der Transistoren Q1 und Q2. R22 und R23 sind Pull-Up-Widerstände für die EN- und IO0-Leitung. Die Kondensatoren C13 und C12 dienen zum entprellen. Die Widerstände R25 und R24 begrenzen den Strom bei Drücken der Buttons S1 und S2.

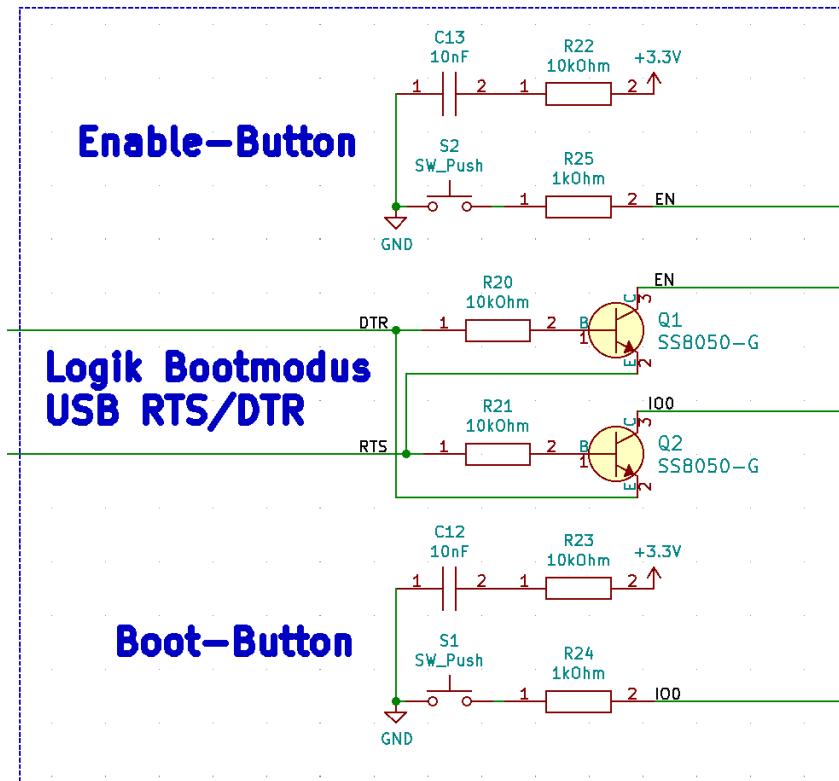


Abbildung 5.22: Schema ESP32-Wroom-32U.

<sup>6</sup>[https://www.espressif.com/sites/default/files/documentation/esp32\\_datasheet\\_en.pdf](https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf) S.13

## Funktionsbeschrieb der Schaltung (Automatische Bootlogik)

### 5.5.3 RFID

Ein Bestandteil der Cocktailmaschine ist, mittels einem persönlichen kontaktlosen Chip oder Badge das Lieblingsgetränk auswählen zu können. Dazu wird auf die RFID<sup>7</sup>-Technologie gesetzt. Sie ermöglicht es mittels RFID-Tags oder auch mit NFC<sup>8</sup> des Handys Daten zwischen Tag und Maschine auszutauschen. So kann der User identifiziert werden.

Damit die Datenübertragung stattfinden kann, wird an der Antenne ein Wechselfeld angelegt, welches im Folgenden Energiesignal genannt wird. Dieses induziert beim Empfänger eine Spannung, welche als Energieversorgung dient. Der Sender moduliert ein Datensignal über das Energiesignal. Die Informationen werden im Tag demoduliert und verwertet. Bei den Befehlen für den Tag geht es hauptsächlich um Lese- und Schreibmethoden. Bei einer Lesemethode des RFID wird ein Teil des Speichers abgefragt, bei Schreibmethoden wird der Speicher beschrieben.

### Schema

Bild 5.23 zeigt den Schaltungsaufbau des RFID-Moduls. Darin ersichtlich sind folgende Teilbereiche:

- MFRC522 RFID IC
- Antenne
- Anpassnetzwerk für Antenne
- Kommunikationsschnittstellen

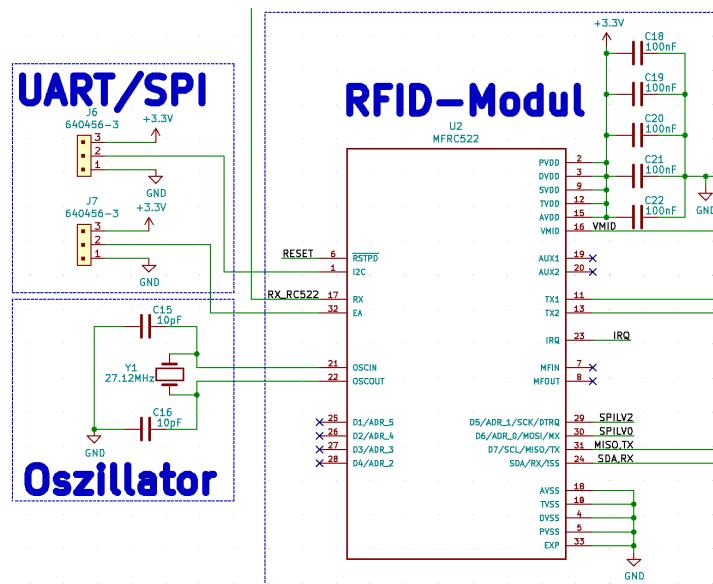


Abbildung 5.23: Schema des RFID Sender/Empfänger

<sup>7</sup>Radio-Frequency Identification

<sup>8</sup>Near-Field-Communication

## Funktionsbeschrieb der Schaltung

Der IC steuert den Ablauf, damit die Datenübertragung stattfinden kann. Er verbindet die Antenne mit dem Cocktailmixer und liest bzw. schreibt die Informationen auf das bzw. aus dem Trägersignal. Die Herausforderung besteht im Design des Anpassnetzwerkes an das IC und die Antenne. Wie die Bauteile dimensioniert werden, kann in einer Anleitung nachgesehen werden. Sie dazu Anleitung: [1].

Die Anschlüsse sind so gestaltet, dass falls eine einens gelayoutete Antenne verwendet wird, die Kommunikation zwischen UART und SPI gewählt werden kann. Aus Zeit- und Backupgründen wurde für die Cocktailmaschine ein Breakout-Board verwendet, welches nur einen SPI-Anschluss hat. In erster Linie damit keine Zeit gebraucht wird, sich mit einem Antennendesign auseinanderzusetzen zu müssen. Aber auch, weil der IC in ein TQFP28-Gehäuse gepackt ist, und deshalb Schwierigkeiten auftreten könnten beim Löten.

## 5.6 Beleuchtung

Damit die Maschine schon von Weitem erkennbar ist, wird ein auffallendes Mittel benötigt. Dazu eignet sich ein LED-Band hervorragend, am besten wenn es noch verschiedenfarbig ist. Dazu wird eine geeignete Ansteuerung benötigt. Es wird davon ausgegangen, dass das LED-Band die benötigten Widerstände schon eingebaut hat und die Anschlüsse somit direkt an die FET's gehangen werden können.

### Schema

Abbildung 5.24 zeigt den Schaltungsaufbau der LED-Steuerung. Damit die LED's angesteuert werden können, braucht es ein Bauteil, welches mit einer 5V-Ansteuerung 12V schalten können. Dazu wird ein MOSFET verwendet. Über die Widerstände an den Gates wird der Strom zum Schutz des Gates begrenzt. Die Leitungen führen direkt auf den Klemmblock für die LED-Streifen. Parallel dazu wurden noch für jede Lichtfarbe ein Kontroll-LED installiert, welche es ermöglicht auch ohne LED-Band etwas zu programmieren.

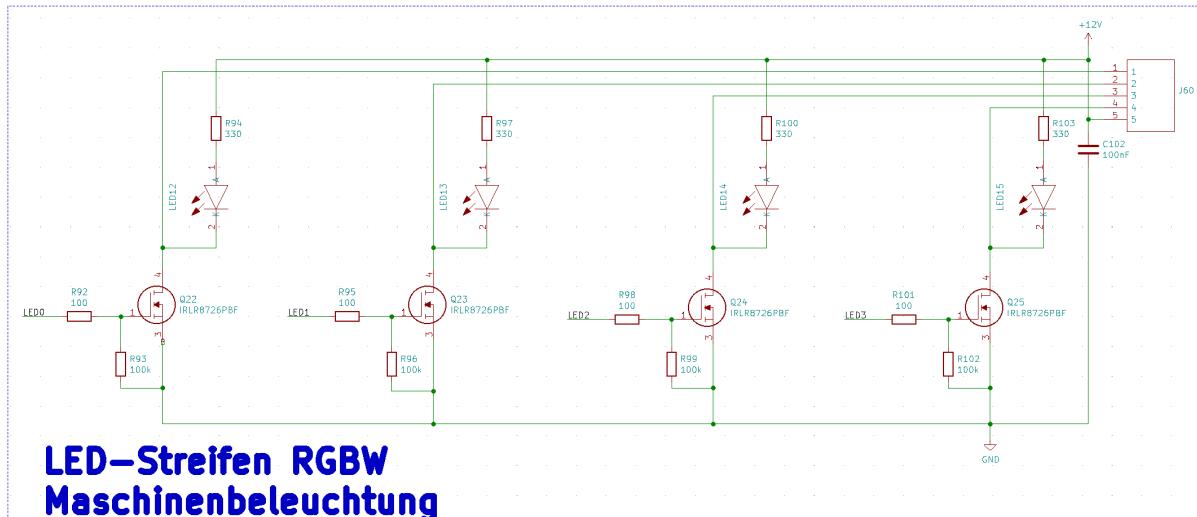


Abbildung 5.24: Schema der LED-Ansteuerung

## Funktionsbeschrieb der Schaltung

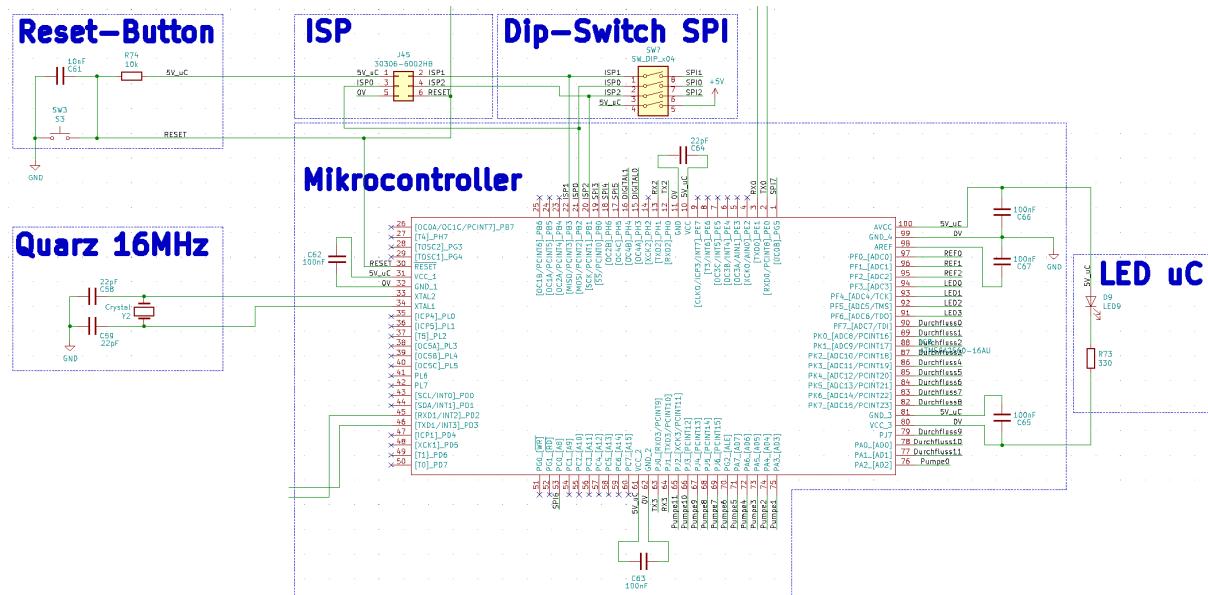
Licht sind bekanntlich elektromagnetische Wellen im sichtbaren Wellenlängen-Bereich. Dabei gibt es vier Hauptfarben: Rot, Grün, Blau und Weiss. Zwar kann Weiss auch aus einer Kombination aller drei Farben erstellt werden, kommt aber besser mit einer separaten Diode. Das resultierende Licht des Bandes ist eine Überlagerung der Wellenlängen. Diese Überlagerung kann vom Mikrocontroller über den Duty-Cycle eines PWM-Signals gesteuert werden. So lassen sich mit dem Licht praktisch aus den Grundfarben praktisch alle Farben mischen.

## 5.7 Mikrocontroller

Der Mikrocontroller ist die Seele, das Gehirn der Maschine. Er muss in der Lage sein mit allen Komponenten auf der Platine zu kommunizieren, die Tasks zu verarbeiten und abhandeln. Der ATmega2560 übernimmt diese Aufgabe.

## Schema

Das Schema besteht aus dem Mikrocontroller IC8 mit den fünf Stützkondensatoren C62, bis C66 an den Spannungseingängen. Mit dem Full Swing Oscillator Crystal Y2 und den Kondensatoren C58 und C59 wird eine Frequenz von 16MHz generiert. Der Reset-Button SW3 dient dazu, den Mikrocontroller manuell neu zu starten. Der dazugehörige Kondensator C81 entprellt den Button und der Widerstand R74 zieht die Leitung mittels Pull-up auf VCC. Mittels dem ISP-Stecker kann über einen Programmer, z.B AVR MKII, auf den Mikrocontroller zugreiffen werden. Der DIP-Switch ist dazu da, die SPI-Leitungen von der ISP-Schnittstelle zu trennen. Die LED D9 gibt Auskunft ob Spannung am Mikrocontroller anliegt.



**Abbildung 5.25:** Schema Mikrocontroller

## Funktionsbeschrieb der Schaltung

Die Stützkondensatoren halten die Spannung am Mikrocontroller konstant. Die Quarz-Schaltung wird gebraucht, da der Mikrocontroller nur eine 8MHz-RC-Clock integriert hat, jedoch mit 16MHz gearbeitet wird. Der Reset-Button zieht in gedrücktem Zustand die Reset-Leitung auf

GND. Dabei entlädt sich der Kondensator schnell, da er kurzgeschlossen wird. In offenem Zustand lädt sich der Kondensator über den Widerstand R74 mit einer Zeitkonstante von  $\tau = R_{74} \cdot C_{81} = 100\mu s$ . Die Trennung von SPI und ISP wird gemacht, sodass gewährleistet ist, dass bei der Inbetriebnahme des Mikrocontrollers die restlichen SPI-kommunikationsfähigen Komponenten nicht mitreden können. Zudem kann die 5V-Versorgungsspannung des Mikrocontroller vom Rest der Platine getrennt werden, sodass dieser auch bei ausgeschalteter Board versorgt werden kann, ohne die anderen Komponenten zu speisen. Deswegen auch die Betriebs-LED.

## 5.8 SD-Karte

Auf der mikroSD-Karte werden die Getränke-, Zutaten, Maschinenfiles abgelegt. Die mikroSD-Karte ist FAT32 formatiert und benötigt ein mikroSD-Adapter um an das System angeschlossen zu werden.

### Schema

In Abbildung 5.26 ist das Schema der SD-Karte zu sehen. Darin erkennbar ist der mikroSD-Adapter J45 mit einem Stützkondensator C68 am Spannungseingang.

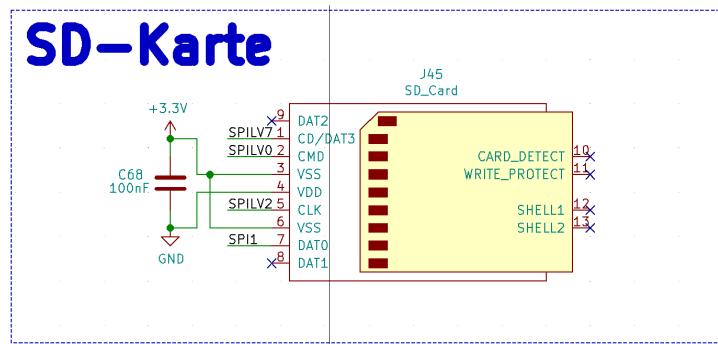


Abbildung 5.26: Schema SD-Karte.

### Funktionsbeschrieb der Schaltung

In der Schaltung ist erkennbar, dass die SPI-Leitungen an die Pins des Adapters führen, genauso die Spannungsleitungen. Da die SD-Karte stets vom Mikrocontroller angesteuert wird, gibt es keine weiteren erwähnenswerte Funktionen zu beschreiben.

## 6 Inbetriebnahme

Ein essentieller Teil der Entwicklung ist die Inbetriebnahme. Dabei sollen Teilsysteme nacheinander in Betrieb genommen werden. Die einzelnen Systeme werden dann in ihren wichtigsten Funktionen geprüft und für den Einsatz vorbereitet. In den folgenden Kapiteln werden die einzelnen Teilsysteme der Cocktailmachine in Betrieb genommen.

## 6.1 Speisungen

Um die Speisungen in Betrieb nehmen zu können ist gemäss Kapitel 5.1 ein Jumper in die Schaltung implementiert worden, welcher die Speisungen von der übrigen Hardware trennt. Ausserdem sind Status-LED's verbaut, welche den korrekten Betrieb anzeigen sollen. Da es sich bei der 48V-Speisung um ein fertiges Netzteil handelt, musste dieses nicht grossartig in Betrieb genommen werden. Die einzige einstellung, welche vorgenommen werden musste, ist die Feinjustierung der Ausgangsspannung mittels Drehregler am Netzteil.

### 6.1.1 12V Speisung

Die 12V Speisung wurde für die Inbetriebnahme vom übrigen System abgekoppelt. Da diese aus der 48V-Speisung generiert wird, wurde lediglich diese vorgeschaltet. Beim einschalten des Netzteils war der korrekte Betrieb am Status-LED sichtbar. Eine Messung mit dem Voltmeter bestätigte dies.

### 6.1.2 5V Speisung

Simultan zur 12V-Speisung wurde die 5V-Speisung in Betrieb genommen. Auch bei dieser Speisung wurde lediglich das Netzteil vorgeschaltet und das restliche System abgeschnitten. Auch hierbei leuchtete das Status-LED sofort auf. Um zu verifizieren ob die richtige Spannung anliegt, wurde dies auch hier mit einem Voltmeter geprüft.

### 6.1.3 3.3V Speisung

Die 3.3V-Speisung wurde als letzte Speisung in Betrieb genommen, da diese von der 5V-Speisung gemäss Kapitel 5.1.4 abhängig ist. Der Linearregler hat am Ausgang auch eine Status-LED zur Verifizierung verbaut. Auch diese leuchtete nach dem Start des Netzteils und dem Zuschalten der 5V-Speisung wie gewünscht auf. Das Voltmeter bestätigte anschliessend die Richtigkeit der Ausgangsspannung.

## 6.2 Programmierschnittstellen

Die Programmierschnittstellen sind werden jeweils mithilfe einen USB-UART-Controller gesteuert. Im Folgenden wird die Inbetriebnahme dieser Schnittstelle für den Mikrocontroller ATmega2560 und das WiFi-Modul ESP32 beschrieben.

### 6.2.1 USB-B

Für die Inbetriebnahme der USB-B-Schnittstelle wurde vorerst geprüft, ob der USB-UART-Converter über die USB-Buchse und Kabel vom Computer erkannt wird. Danach wurde geprüft, wie der Handshake beim Programmieren der Komponenten funktioniert. Die Inbetriebnahme der Programmierung wird in den Kapitel 6.3.2 (WiFi-Modul) und Kapitel 6.4.4 (Mikrocontroller) beschrieben. Dies geschieht ein wenig parallel, denn die Programme zum Schreiben, Kompilieren und Hochladen des Codes müssen schon vor dem Prüfen der USB-B-Schnittstelle installiert sein. Für die Inbetriebnahme wurde entschieden, dass der Handshake noch zum USB-B-Teil gehört und als Voraussetzung zählt, sodass das Programm überhaupt hochgeladen werden kann. Ausserdem wird dieser Handshake unabhängig davon gemacht, ob ein Device angehängt ist oder nicht. Erst nach einer Überprüfung der Device-Nummer kann bestimmt werden, ob das Target (ESP32 und ATmega2560) vorhanden ist oder nicht. Um den Handshake zu kontrollieren sind die in Abbildung 6.1 eingekreisten Punkte interessant:

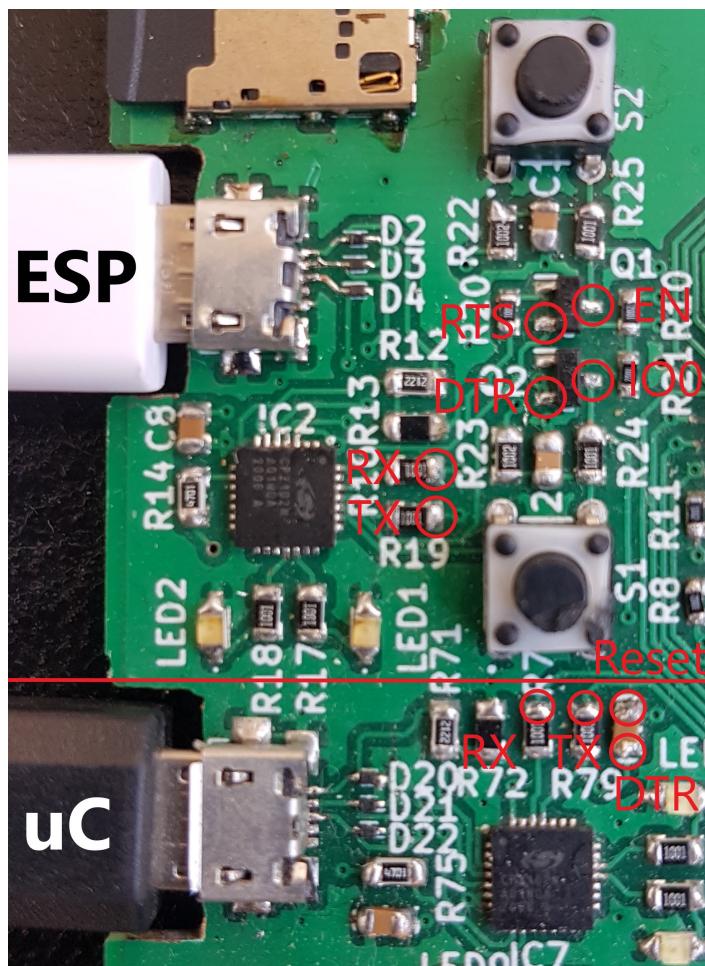


Abbildung 6.1: Ausschnitt aus Platine (USB-Schnittstellen) mit eingekreisten Messpunkten.

Folgende Schritte wurden befolgt:

1. Platine mit Spannung versorgen und Computer mit zwei USB-Kabeln an die USB-B-Buchsen anschliessen.

[Systemeinstellungen → Geräte-Manager](#) (siehe Abbildung E.1)

Dort sind die Devices sichtbar unter dem Name: *Silicon Labs CP210x USB to UART Bridge*

2. Überprüfen der Handshakes, welche definiert sind in den Tools zum Hochladen der Software.

- (a) ATMega2560

Um herauszufinden, ob und wie der Handshake durchgeführt wird, wurde beim Hochladen eines Programms an den folgenden Leitungen gemessen:

1 Gelb DTR

2 Blau Reset

3 Violett RX In Abbildung 6.2 ist die Kommunikation mit den ersten Bytes

4 Grün TX

zu sehen, in Abbildung 6.3 wird der Handshake genauer aufgezeigt. Sobald die DTR-Leitung auf GND gezogen wird, fällt die Spannung über dem Reset-Pin aufgrund des Kondensators zwischen DTR und Reset nur kurzzeitig. Dies reicht jedoch aus, dass der Mikrocontroller in den Boot-Modus fällt. Nach ca. 50ms beginnt die Datenübertragung vom Computer in den Flash-Speicher des uC.

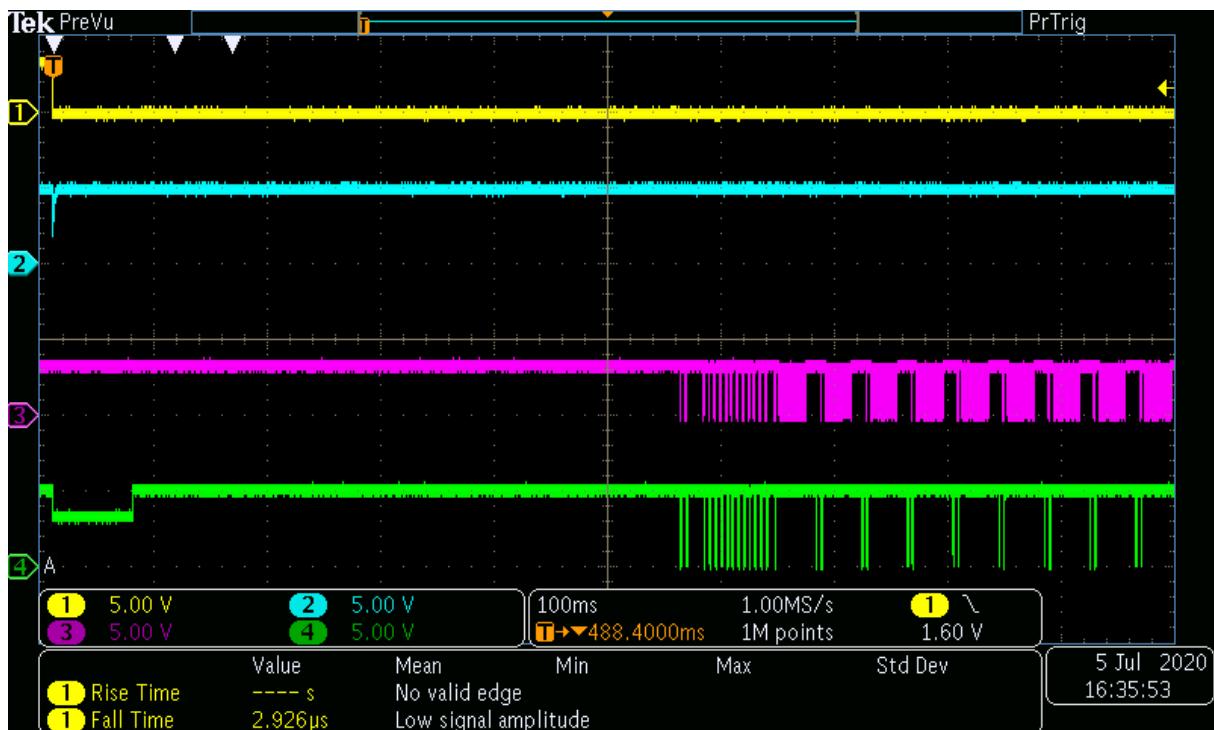
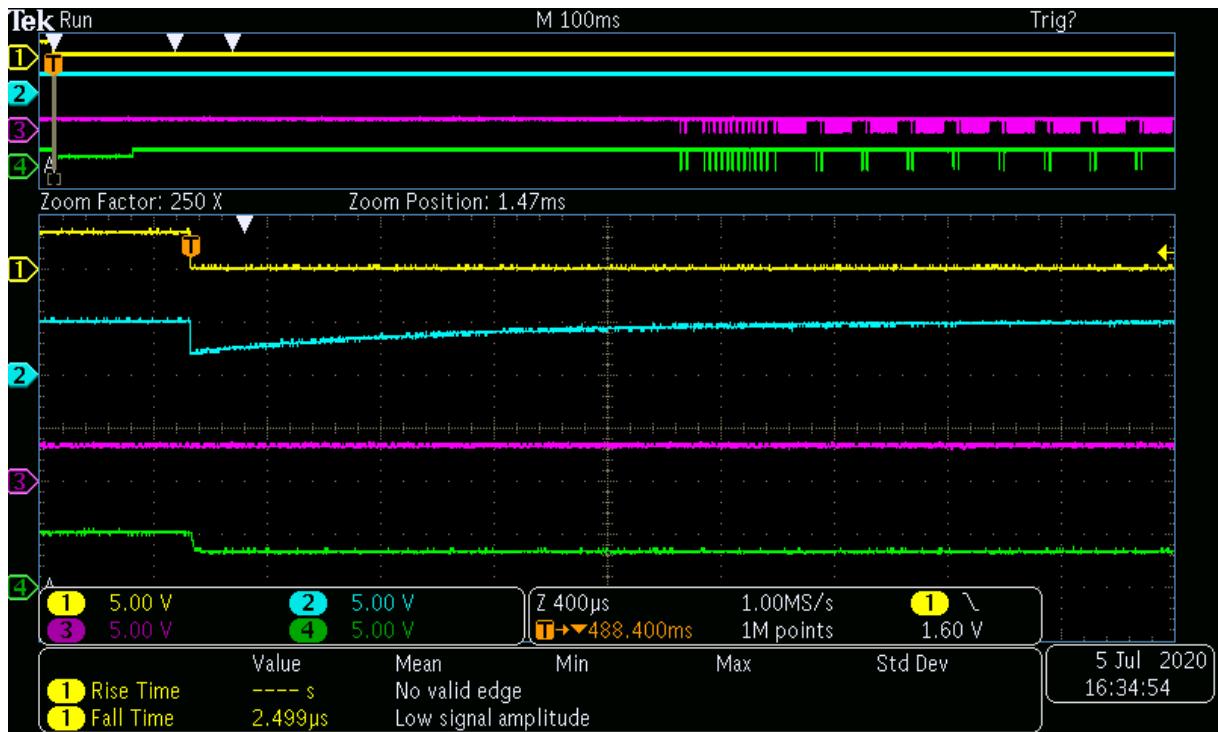


Abbildung 6.2: Hochladen des Programmcodes auf den ATMega2560.



**Abbildung 6.3:** Handshake zum Hochladen des Programmcodes auf den ATMega2560. Zoom auf den Moment des Resets.

(b) ESP32

Um herauszufinden, ob und wie der Handshake durchgeführt wird, wurde beim Hochladen eines Programms an den folgenden Leitungen gemessen:

- |   |         |     |
|---|---------|-----|
| 1 | Gelb    | RTS |
| 2 | Blau    | DTR |
| 3 | Violett | EN  |
| 4 | Grün    | IO0 |

In Abbildung 6.4 wurden die ersten 100ms des Hochladen eines Programms auf das ESP32 gemessen. Gesucht wird nach einem gleichzeitigen Flankenwechsel der RTS-Leitung von 0 auf 1 und der DTR-Leitung von 1 nach 0. Abbildung 6.5 zeigt eine genauere Aufnahme zum Zeitpunkt des Flankenwechsels.

Was auffällt, ist dass entgegen der Erwartung der Flankenwechsel der beiden Leitungen leicht verzögert ist (um ca. 80  $\mu$ s). Auch das Signal des EN-Pins erreicht wesentlich schneller einen HIGH-Zustand als erwartet. Denn gemäss der Theorie müssen die Pins EN und IO0 zum gleichen Zeitpunkt auf LOW sein, um in den Download-Boot-Modus zu kommen, und trotz der Abweichung der Praxis zur Theorie funktioniert die Übertragung des Codes. Ein Fehler in der Matrix?

1 Gelb RTS | 2 Blau DTR | 3 Violett EN | 4 Grün IO0

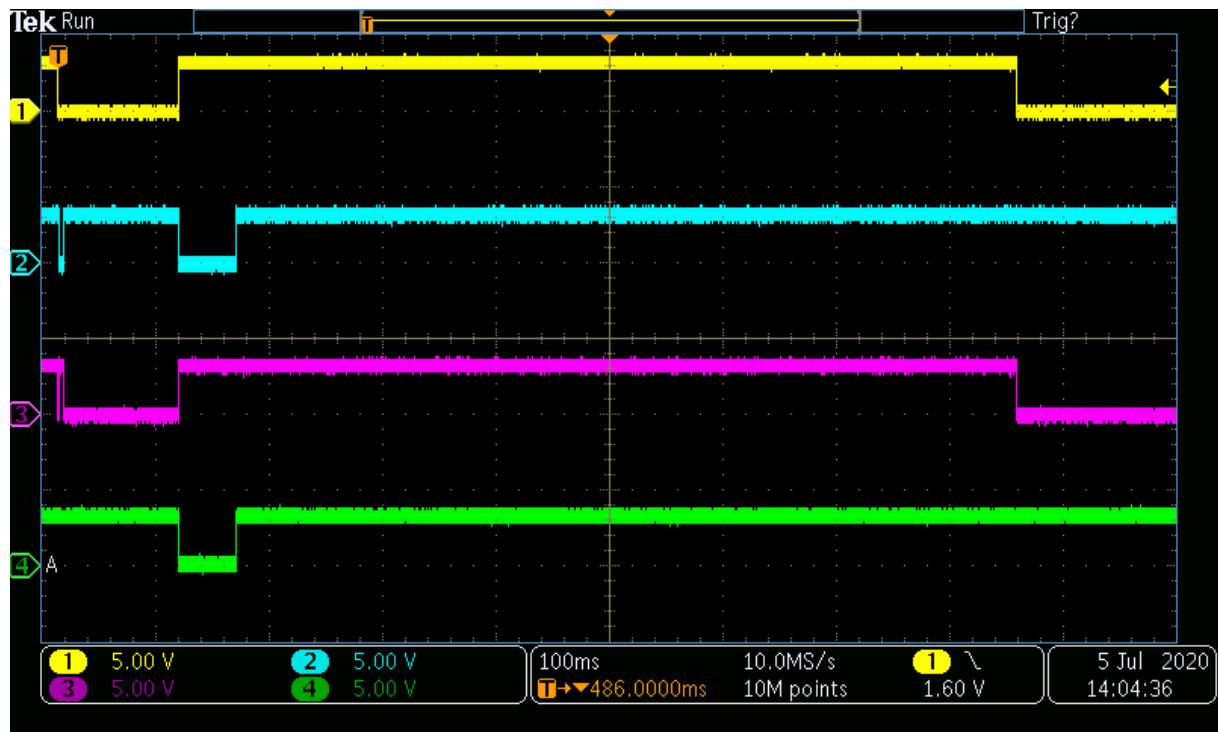
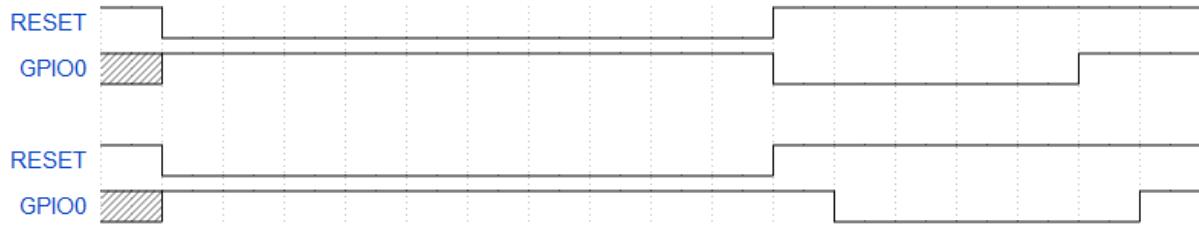


Abbildung 6.4: Handshake zum Hochladen des Programmcodes auf das ESP32.



Abbildung 6.5: Handshake zum Hochladen des Programmcodes auf das ESP32. Zoom auf gleichzeitiger Flankenwechsel RTS und DTR.

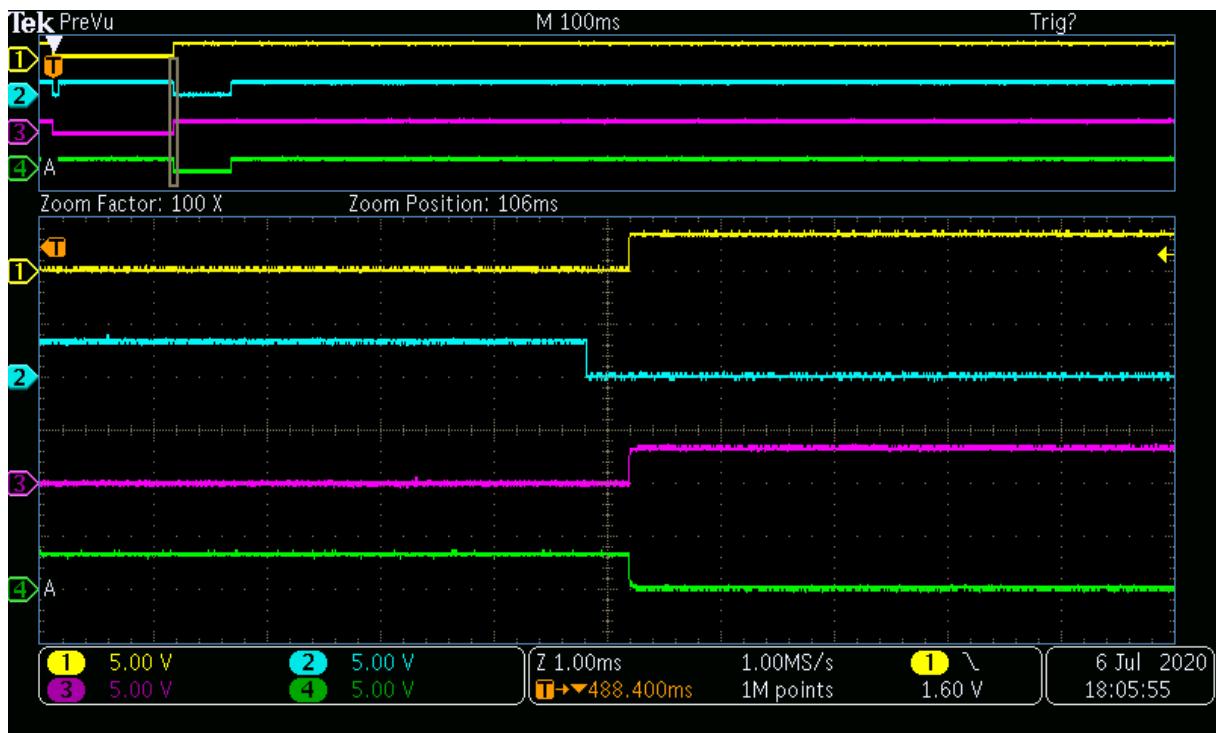
In einem Forum<sup>9</sup> wurde diese Auffälligkeit auch schon besprochen. Die Diskussion führte zum selben Ergebnis wie bei Inbetriebnahme. Nämlich, dass es nach dem Reset eine Zeit dauert, bis im Startprozess die Pins geprüft werden. Dazu gehört auch der Pin IO0. Somit ist es möglich, den Pin IO0 kurz nach dem Reset auf 0 zu ziehen. Im selben Forum wurde auch die in Abbildung 6.6 gezeigte Darstellung gefunden. Ein Kommentar weist darauf hin, das mit dem esptool.py der EN-Pin des ESP32 direkt auf RTS gehängt werden kann. So lassen sich die Zeitpunkte, zu der die Pins auf 0 sind, näher zusammenschieben.



**Abbildung 6.6:** Handshake zum Hochladen des Programmcodes auf das ESP32. Zoom auf gleichzeitiger Flankenwechsel RTS und DTR.

Die Messung nach dem Einlöten der Brücke bestätigt dies, wie in Abbildung 6.7 ersichtlich ist. Allerdings spielt der Kondensator jetzt nicht mehr so eine grosse Rolle, was auch nicht nötig ist. Auf das Hochladen des Codes hat die Brücke keinen Einfluss. Die Funktioniert wie bei der Schaltung ohne Brücke einwandfrei.

1 Gelb RTS | 2 Blau DTR | 3 Violett EN | 4 Grün IO0



**Abbildung 6.7:** Handshake zum Hochladen des Programmcodes auf das ESP32. Zoom auf gleichzeitiger Flankenwechsel RTS und DTR.

<sup>9</sup><https://forum.micropython.org/viewtopic.php?t=4607>

### 6.3 Benutzerschnittstellen

Die Benutzerschnittstellen sind sehr Unterschiedlich. Sie gemeinsames haben sie indem sie praktisch alle über die UART-Schnittstelle funktionieren.

#### 6.3.1 Touch-Display

Der Test für das Display gestaltet sich am schnellsten. Die Schnittstelle wird mit dem schon im P5 geschriebenen Code getestet. Für den Test wird erwartet, dass ein Druck auf das Bild in der Mitte mit dem Cocktail eine Funktion auslöst. In dieser Funktion wird auf dem Display ein anderes Bild geladen und die Texte in den Buttons geändert. Eine andere Funktion simuliert eine Zubereitung eines Cocktails. Beide Funktionen haben gleich wie im P5 einwandfrei funktioniert.

#### 6.3.2 ESP

Die Inbetriebnahme des ESP32-WROOM-32U geschieht mit der Programmierumgebung Arduino IDE. Um diesen in Betrieb nehmen zu können, waren einige Einstellungen und Downloads nötig, welche dann in der Arduino IDE eingebunden werden können. Es wird getestet, ob das ESP sich im vorgegebenen Netz anmeldet und als Webhost dient, ob bei Klicken auf ein Button im Explorer ein Event auslöst, ob die Kommunikation zwischen µC und Mikrokontroller funktioniert.

Folgende Schritte wurden befolgt:

1. Benötigte Daten von Github<sup>10</sup> herunterladen.
2. Dateien Entpacken und speichern unter:  
`C:\Users\”Benutzer”\Documents\Arduino\hardware\espressif\esp32`  
Damit die Arduino IDE die Files findet.
3. Arduino IDE starten und folgenden Reiter öffnen:  
`Arduino IDE → Werkzeuge → Board`  
ESP32 Dev Module auswählen.
4. Unter demselben Reiter können noch weitere Einstellungen getätigt werden.  
`Arduino IDE → Werkzeuge →...`  
Upload Speed : 921600  
CPU Frequency : 240MHz  
Flash Frequency : 80MHz  
Flash Mode : QIO  
Flash Size : 4MB (32Mb)  
Partition Scheme : Default 4MB mit spifss  
Core Debug Level : none  
PSRAM : disabled  
Port : **COMx**  
Wobei der Port **COMx** im Geräte-Manager ermittelt werden muss. Das ESP32 ist jetzt flashbar.

---

<sup>10</sup><https://github.com/espressif/arduino-esp32>

5. Testprogramm herunterladen<sup>11</sup>, leicht modifizieren und hochladen.

**Arduino IDE → Verify and Upload Button**

Für die Inbetriebnahme wurde das Testprogramm so modifiziert, dass das ESP32 gleich getestet werden kann wie das Touch-Display. Dies bedeutet, dass das ESP eine Page-ID, eine Button-ID und das Abschlusszeichen 0xFF 0xFF 0xFF sendet. Der Unterschied ist, dass das ESP über einen anderen UART-Port des µC kommuniziert. Die Testsoftware ist bei den anderen Firmwares auf dem USB-Stick zu finden.

6. Die "Debug Kommunikation" über den ersten seriellen Port des ESP32 testen.

**Arduino IDE → Tools → Serial Monitor**

Im Testprogramm wird hier angezeigt, wenn ein neuer Client die IP-Adresse aufruft und wenn im Internetexplorer ein Button gedrückt wird. Erste Versuche nach dem hochladen waren erfolgreich, das ESP hat eine IP-Adresse zugeordnet bekommen.

Über die zweite serielle Schnittstelle findet die Kommunikation zwischen ESP32 und Atmega2560 statt. Sobald über den Webserver eine Aktion ausgelöst wird, sendet das ESP die gleiche Zeichenfolge, welche auch das Display ausgibt. Der Test zeigt, dass das Drücken auf den Button im Internetexplorer die gleiche Aktion auslöst, wie das Drücken auf das Touch-Display. Es werden Bilder neu gesetzt und Texte geändert. Die Implementierung des EPS ist folglich erfolgreich.

### 6.3.3 RFID

Robin: Inbetriebnahme RFID beschreiben

---

<sup>11</sup><https://randomnerdtutorials.com/esp32-web-server-arduino-ide/>

## 6.4 Mikrocontroller

Um das Anwenderprogramm auf dem Mikrocontroller ( $\mu$ C) speichern zu können, ist es am angenehmsten, wenn dies direkt aus der Programmierumgebung geschehen kann. Als Programmierumgebung wird aufgrund des AVR-Chips die Software Atmel Studio 7.0 ausgewählt.

### 6.4.1 Bootloader

Ein Bootloader (BL) ermöglicht seitens  $\mu$ C den Programmierungsvorgang über USB und stellt sicher, dass der frisch kompilierte Code in den Programmspeicher des  $\mu$ C gelangt. Im Grunde ist der BL ein Code am Beginn des Programmablaufs, welcher entsprechend nach einem Reset aufgerufen wird. Innerhalb des BL-Codes wird für zwei Sekunden auf eingehende Daten der UART0-Schnittstelle gewartet. Wird innerhalb dieser zwei Sekunden ein Anwenderprogramm in Form eines HEX-Files an den Mikrocontroller gesendet, wird es gemäss stk500v2-Protokoll in den Flash-Speicher geladen. Aufgrund des BL geht es nach einem Reset des  $\mu$ Cs zwei Sekunden, bis das eigentliche Programm startet.

Für den  $\mu$ C des Cocktailmixers wird ein stk500v2-BL verwendet. Dieser kann über Github<sup>12</sup> heruntergeladen werden. Im File stk500v2.c befinden sich einige Anweisungen, welche Fuse- und Lock-Bits wann gesetzt werden müssen. Achtung, die Angaben haben im Falle des Cocktailmixers nur mit Abweichungen funktioniert (4096 words anstelle 1024). Durch Anpassen des start-Vektors auf den Programmspeicher und Setzen der korrekten Bootloader-Fuses wäre es möglich, den BL-Speicherplatz zu verkleinern um den Programmspeicher zu vergrößern.

cite:  
<http://www.m...>

### 6.4.2 Fuse-Bits

Da während dem Entwickeln die Möglichkeit bestehen soll, den Flash-Speicher per USB zu beschreiben, muss beim Aufstarten der BL aufgerufen werden. Dazu muss das BOOTRST-Bit aktiviert werden. Der Speicherplatz für den BL wird auf 4096 words gesetzt (anders als stk500v2 Erklärung). Das EEPROM soll beim Löschen des  $\mu$ C geschützt bleiben, weshalb das EESAVE-Bit aktiviert wird. Da die ISP-Schnittstelle benötigt wird, um den BL zu schreiben und Fuse-Bits zu setzen, wird das SPIEN-Bit gesetzt. Für den  $\mu$ C verwenden wir einen 16MHz Full-Swing-Crystal-Oszillatator, weshalb die Bits CKSEL3:1 auf 111 stehen müssen. Die Aufstartzeit wird vorsorglich auf die längst mögliche Zeit eingestellt. Dies führt dazu, dass das Register CKSEL0 auf 0 und die Register SUT0:1 auf 00 gesetzt werden. Der Brown-out-Detektor setzt den internen Reset, sobald die Versorgungsspannung unter einen Wert fällt. Wenn der Mikrocontroller ausfällt, während der TMC4671 noch aktiv ist, wird der Motor weiter gefahren. Dieses Risiko soll eingeschränkt werden, indem dieser Modus ausgeschaltet wird.

Die Lock-Bits müssen gesetzt werden, nachdem der BL in den Speicher geschrieben wurde. "SPM ist nicht erlaubt, in den Anwenderbereich zu schreiben und LPM ist nicht erlaubt aus dem Applikationssktor zu lesen, wenn LPM aus dem Urlader-Bereich ausgeführt wird."

cite :  
<https://www...>  
<https://www...>  
<https://www...>  
<https://www...>

---

<sup>12</sup><https://github.com/arduino/Arduino-stk500v2-bootloader/blob/master/stk500boot.c>

Daraus folgt für die Fuse- und Lock-Bits die Einstellungen gemäss Tabelle 6.1. Siehe Anhang D und F für Details.

Extended	High	Low	Lock
0xFF	0xD0	0xF7	0xCF

**Tabelle 6.1:** Tabelle Fuse- und Lock-Bits.

#### 6.4.3 AVRdude in Atmel Studio einbinden

AVRdude ist eine Software, mit der Atmel AVR Controller programmiert werden können. Sie schreibt den bereits kompilierten HEX-Code der Programmierumgebung über den Bootloader in den Flash-Speicher des Controllers. Hier gäbe es auch eine Methode, die Fuse- und Lock-Bits des µC zu setzen.

<https://www.mikrocontroller.net/thread/113777>

Über einen Link<sup>13</sup> kann eine Datei heruntergeladen werden in Form von [avrduude-6.3-mingw32.zip](http://download.savannah.gnu.org/releases/avrduude-6.3-mingw32.zip). Der gleichnamige Ordner wird im Ordner **C:\Tools** gespeichert. Danach wird in AtmelStudio der Reiter **”Tools→External Tools”** ausgewählt und ein neues Tool hinzugefügt. Im Falle des Atmega2560 geben wir die Commands gemäss Tabelle 6.2 ein:

Title	:	Cocktailmixer
Command	:	C:\Tools\avrduude-6.1-mingw32\avrduude.exe
Arguments:	:	-D -P <b>COMx</b> -p ATMEGA2560 -c wiring -b 115200 -U flash:w:\${TargetDir}\${TargetName}.hex:i

**Tabelle 6.2:** AVRdude Commands

Der entsprechende **COMx**-Port des zu flashenden Gerätes (Atmega2560) muss mit dem Gerätemanager ermittelt werden.

Sämtliche Tabellen aus dem Datenblatt und Screenshots aus Programmierumgebung, welche mit dem Setzen der Fuse- und Lock-Bits oder Programmierung des µC zusammenhängen, sind im Anhang Kapitel D angefügt.

cite Herr  
Meier  
Skript mc1

#### 6.4.4 Vorgehen

1. Als Erstes wurden die Fuse-Bits gesetzt. Dies geschah über den Reiter: [AtmelStudio → Tools → Device programming → Fuses](#) (siehe Abbildung F.1)  
Es wurde darauf geachtet, dass der AVR mkII ausgewählt wurde und der Gerätecode des Atmega2560 ausgelesen werden konnte.
2. Als Zweites wurde der Bootloader installiert. Dies geschah unter:  
[AtmelStudio → Tools → Device programming → Memory](#) (siehe Abbildung F.4)  
Hier wird ein stk500v2-BL verwendet, kann aber auch abweichen. (Entsprechende Anpassungen nötig, nicht Teil dieses Projektes.)

<sup>13</sup><http://download.savannah.gnu.org/releases/avrduude/>

3. Als Drittes wurden die Lock-Bits gesetzt unter:  
[AtmelStudio → Tools → Device programming → Lock-Bits](#) (siehe Abbildung F.3)  
Diese sollten nicht mehr geändert werden. Bei jedem Brennen des BL wieder zu setzen.
4. Ggf. USB-Firmware installieren auf dem USB-UART-Converter. (Nicht Teil dieses Projektes.)
5. Mikrocontroller mit der kompilierten Software (Cocktailmixer.HEX) programmieren.  
[AtmelStudio → Tools → Cocktailmixer](#)  
Alternativ direkt mit ISP-Programmer wie in Schritt 2 (ohne Bootloader, Programmcode an start-Vektor des µCs.):  
[z.B C:\Users\DaU\Software\Cocktailmixer\Cocktailmixer\Debug\Cocktailmixer.HEX](#)

Das Setzen der Fuse- und Lock-Bits sowie das brennen des Bootloaders kann mit einem AVR MKII Programmer in Atmel Studio gemacht werden. Alternativ gibt es einen Weg, den USB-Treiber (Atmega16U2) eines Arduino Uno mit einer entsprechenden Firmware zu laden, sodass dieser als Programmer verwendet werden kann<sup>14</sup>.

Für die Inbetriebnahme des Mikrocontrollers wurde der Alternativweg gewählt. Die Ergebnisse können sich zeigen lassen. Der Mikrocontroller ist programmierbar und erste Tests mit der Software waren erfolgreich, das Programm wurde auch ordnungsgemäß gestartet. Dies wurde geprüft, indem der während dem Projekt 5 erarbeitete Code hochgeladen wurde.

Wird der Mikrocontroller per Reset-Button neu gestartet, dauert es aufgrund des Bootloaders 2s, bis der Programmcode gestartet wird. Während dieser Zeit wartet der Bootloader auf einkommende Daten, welche auf den Flash-Speicher geschrieben werden sollen. Danach startet das Programm, sollten keine Daten kommen.

#### 6.4.5 Messungen

Die Messung des Oszillator-Eingangs ergab eine saubere harmonische Schwingung mit 16MHz, abgebildet in Abbildung 6.9.

---

<sup>14</sup><https://www.instructables.com/id/Turn-Ardudos-Serial-Converter-Into-AVRISP-MkII-CI/>

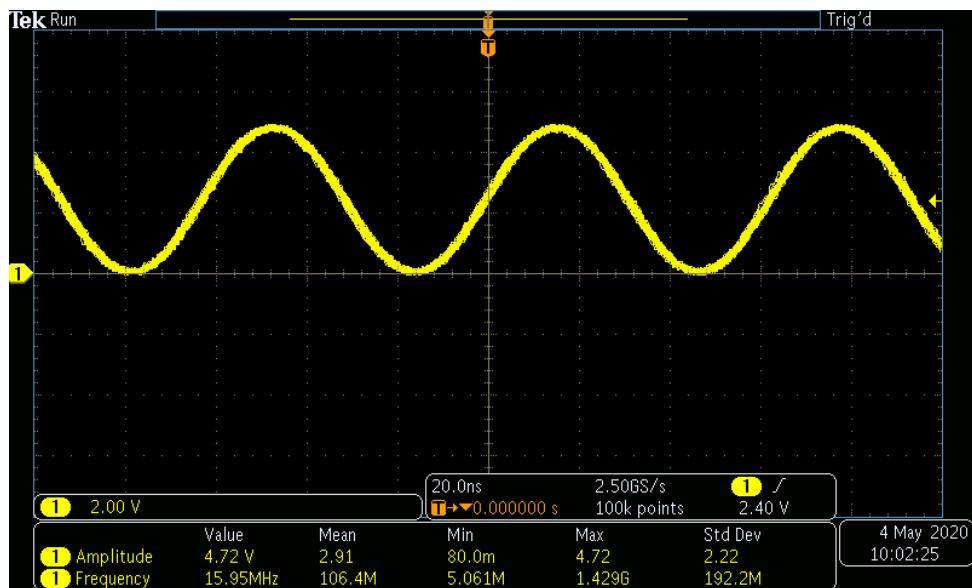


Abbildung 6.8: Schwingung des Oszillators

## 6.5 Datenspeicherung

Im Folgenden wird die Datenspeicherung in Betrieb genommen. Dies geschieht mit dem Programm, welches in der Library vorhanden ist. Damit kann auf das Directory Table zugegriffen werden und die darauf vorhandenen Files gelesen und bearbeitet werden. Es können auch neue Files generiert werden.

Die benutzte Library ist eine ältere Version. Die neueren Versionen sind auf der Homepage gezeigt und bieten einen noch breiteren Verwendungszweck. Ein wirklich gelungenes Projekt.

DharmaniTech - Blog with circuit designs and ideas around microcontrollers!!

### 6.5.1 mikroSD-Karte

Vorgehen:

1. Benötigte Applikation (Version 2.1) von Downloadseite<sup>15</sup> runterladen und auf das System anpassen oder direkt das File aus Abgabe-USB in Atmel Studio öffnen.  
**Software→Atmega→2\_Testapplikation\_SD\_Karte →SD\_Karte\_Testapplikation**
2. Software hochladen:  
**AtmelStudio →Tools →Cocktailmixer**
3. Programm für Kommunikation über die serielle Schnittstelle downloaden (z.B HTerm 0.8.1beta)<sup>16</sup>
4. Verbindung mit Mikrocontroller herstellen und Neustart über Button DTR auslösen.

<sup>15</sup><https://www.dharmanitech.com/2009/01/sd-card-interfacing-with-atmega8-fat32.html>

<sup>16</sup>z.B unter <https://www.heise.de/download/product/hterm-53283>

Port	=	<b>COMx</b>
Baudrate	=	9600
Data	=	8
Stop	=	1

Der Port **COMx** ist aus dem Geräte-Manager zu entnehmen. Es ist derselbe Port, welcher verwendet wird um die Software hochzuladen.

5. Ob eine SD-Karte gefunden wurde, ist erkennbar, wenn die Version der SD-Karte angezeigt wird. Jetzt sollte sich die SD-Karte lesen und beschreiben lassen.

## 6.6 Flüssigkeitsbeförderung

### 6.6.1 Pumpen

### 6.6.2 Durchflussmessgeräte

## 6.7 Beleuchtung

## 6.8 Motor

Der Motor wird in verschiedenen Etappen in Betrieb genommen. Beginnend mit dem FOC-Treiber, über den Gate Treiber wird zum Schluss die Inbetriebnahme der H-Brücke und des Motors beschrieben.

### 6.8.1 BLDC und H-Brücke

### 6.8.2 ABN-Encoder

### 6.8.3 FOC-Treiber

Der FOC-Treiber wird über die SPI-Schnittstelle in Betrieb genommen. Dazu werden die Parameter verwendet, welche aus der TMCL-IDE verwendet werden. Die Standardparameter beinhalten Informationen zum Motor, zwei Sekunden Linksdrehung im Open-Loop, 2 Sekunden Rechtsdrehung im Open-Loop und dann Stop.

Die Initialisierung sowie das Auslesen gewisser Register ist mit der Testapplikation "Motor" möglich. Auch die Initialisierung und Kommunikation mit den Gate-Treiber TMC6200 kann mit diesem getestet werden.

Vorgehen:

1. Benötigte Applikation aus dem Software-Ordner auf dem USB-Stick in Atmel Studio öffnen.

Software→Atmega→5\_Motor→Motor

2. Software anpassen:

**TMC4671\_init();**  
ggf. Pins, UART- und SPI-Schnittstelle

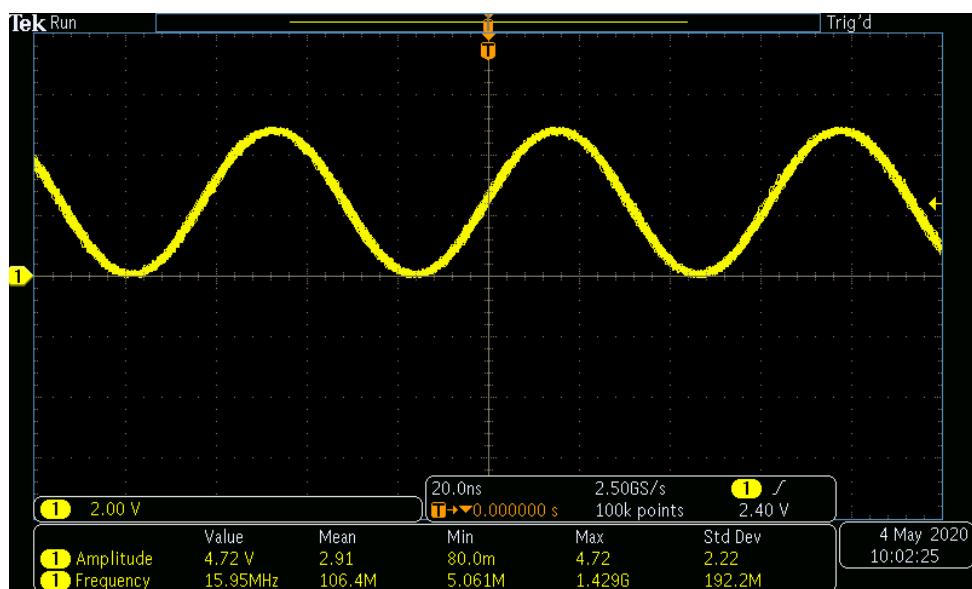
Kurz beschreiben,  
was die Register beinhalten

3. Software hochladen:  
[AtmelStudio→Tools→Cocktailmixer](#)
4. Ausgehende Gate-Signale zum Gate-Treiber mit Oszilloskop überprüfen. Da das Resultat mit dem aus dem P5 übereinstimmt, gilt der FOC-Teil als in Betrieb genommen. Im Anhang Kapitel A sind die Messbilder eingefügt (Abbildungen ??)

Bilder machen und einfügen mit Oszi.

#### 6.8.4 Gate-Treiber

Der Gate-Treiber wird ebenfalls über die SPI-Schnittstelle in Betrieb genommen. Dazu werden die Parameter verwendet, welche aus der TMCL-IDE verwendet werden. Die Standardparameter beinhalten Informationen zum Motor und Einschalten des Treibers.



Kurz beschreiben, was die Register beinhalten

**Abbildung 6.9:** Schwingung des Oszillators

Die Initialisierung sowie das Auslesen gewisser Register ist mit der Testapplikation "Motor" möglich. Auch die Initialisierung und Kommunikation mit den Gate-Treiber TMC4671 kann mit diesem getestet werden.

Vorgehen:

1. Benötigte Applikation aus dem Software-Ordner auf dem USB-Stick in Atmel Studio öffnen.  
[Software→Atmega→5\\_Motor→Motor](#)
2. Software anpassen:  
[TMC6200\\_init\(\);](#)  
ggf. Pins, UART- und SPI-Schnittstelle
3. Software hochladen:  
[AtmelStudio→Tools→Cocktailmixer](#)

4. Ausgehende Gate-Signale auf den Gatevorwiderstand des MOSFETS mit Oszilloskop überprüfen.

#### Fehler

Hier ergibt sich ein Problem mit dem Ansteuern der H-Brücke. Der Gate-Treiber schaltet die Gatesignale des FOC-Treibers nicht wie gewünscht durch, sondern bleibt auf einem Schaltzustand stehen. Dies führt dazu, dass die MOSFETS die H-Brücke nicht schalten und der Motor nicht läuft. Die Belegung der Pins und Spannungslevel an den Konfigurationspins ist jeweils die Selbe wie beim EVAL-Kit.

Bilder machen und einfügen mit Oszi.

Eine Möglichkeit besteht in der Beschaltung des externen Gate-Spannungs-Regulator. Im Datenblatt des TMC6200-TA steht auf Seite 11 geschrieben, dass die Verluste der internen Spannungsregler bei höheren Spannungen ab 48V mit einem externen Spannungsregler verringert werden können. Die sonstige Beschaltung ist gleich der von Trinamic.

Aus Zeitgründen wurde entschieden, dass zur Lösung das Universal Power Supply UPS-70V10A zum Einsatz kommt. Dieses wurde schon im P5 verwendet und ermöglicht ein Umgehen des Gate-Treibers, indem einfache Bauteile ohne komplizierte Logik verwendet werden. Dies hat den Vorteil, dass für die Ansteuerung des Motors nur die schon in Betrieb genommenen Gate-Signale des TMC4671 von Wichtigkeit sind. Für einen vorläufigen Betrieb läuft der Print also mit externer H-Brücke und Gate-Treiber.

## 7 Software

Im Folgenden Kapitel werden die beiden Softwareteile beschrieben. Die Software für den Mikrocontroller beinhaltet die komplette Ansteuerung der Teilsysteme, die Führung durch das Menu auf dem Display sowie die Ausführung der Funktionen bei Auswahl auf dem Display. Ausserdem wird hier der gesamte Stand der Maschine gespeichert.

Die Software für das ESP32 erweitert die Funktionalität der Maschine, indem einige Funktionen auch über eine Android-Applikation auf dem Handy aufgerufen werden können. Die Software auf dem ESP32 übernimmt dabei eine Zwischenfunktion, indem es die eingehende Bluetooth-Kommunikation der Android-Applikation übersetzt und die Daten an den Mikrocontroller sendet. Ausserdem werden Daten vom Mikrocontroller abgefragt und auch das RFID-System wird in das ESP32 implementiert.

### 7.1 Atmega2560

Die Software für den Atmega2560 ist wie folgt aufgebaut: Im **Init/Main** werden die Variablen deklariert, welche im Programm benutzt werden, um die aus den Kommunikationsbuffern geholten Daten zur Verarbeitung zu Speichern. Zudem werden hier die Adressen auf die Funktionen deklariert, welche die Module (z.B IO, Speicher, Devices, Interfaces) initialisieren und die Buffer in der main-Schleife prüfen. Durch Aufrufen des h-Files "Cocktail\_Statemachine.h" werden die Adressen der projektspezifischen Funktionen deklariert und somit die **User-Applikation** eingebunden. Darunter befinden sich die Libraries, welche die Registernummern oder Befehlssätze beinhalten, um die Devices anzusteuern. Sie bilden die Schnittstelle zwischen User-Applikation und Kommunikationsinterface. Dies ist vergleichbar mit einem **Application-Programming-Interface**<sup>17</sup>, einem Programmteil, welcher eine Verbindung eines Programms zu einem anderen Programm ermöglicht. Die Informationen werden dabei standardisiert zwischen den Anwendungen ausgetauscht. Die Daten oder Befehle werden strukturiert nach einem definierten Syntax übergeben. Der Zugriff auf die Hardware des Mikrocontrollers (SPI- und UART-Interface) erfolgt mittels den AVR-Registern. Die Libraries, die dafür verwendet werden befinden sich folglich zwischen dem "API"-Layer und der Hardware und können mit der **Hardware Abstraction Layer**<sup>18</sup> verglichen werden. Es wird nur über diese Funktionen auf die entsprechende Hardware zugegriffen.

cite:  
<https://www.insider.de/was-ist-ein-application-programming-interface-api-a-735797/>

---

<sup>17</sup> API, Schnittstelle zur Anwendungsprogrammierung

<sup>18</sup> HAL, Hardwareabstraktionsschicht

### 7.1.1 Strukturplan

In Abbildung 7.1 wird aufgezeichnet, wie die Software aufgebaut ist und welcher Teil der Software von welchen Libraries abhängig ist. Wie in Kapitel ?? beschrieben kann die Software in folgende Teile gegliedert werden:

- Init/Main
- User Application
- API - Application-Programming-Interface
- HAL - Hardware-Abstraction-Layer
- Interfaces / Pins

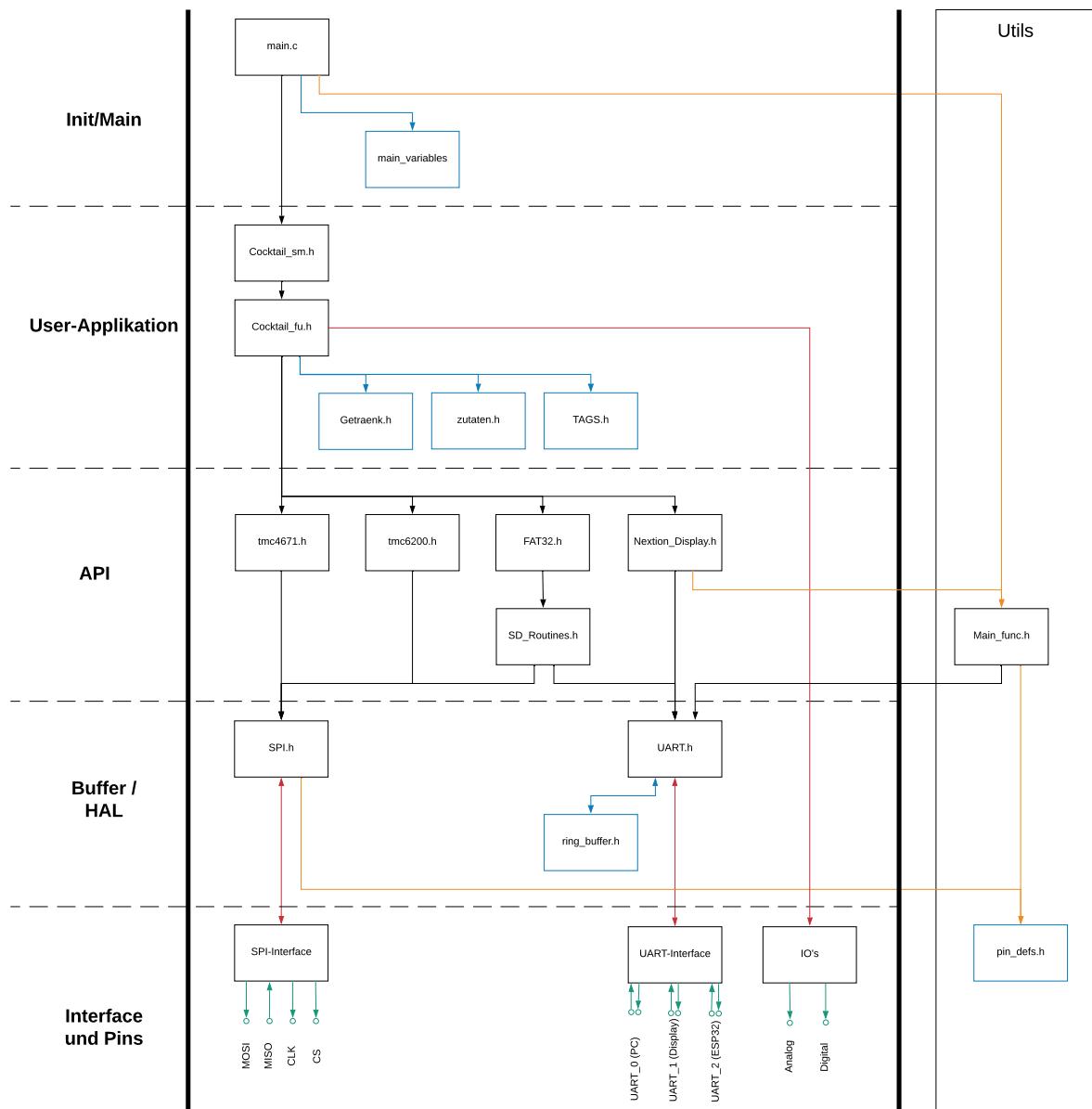


Abbildung 7.1: Softwareübersicht Atmega2560.

### 7.1.2 Programmflussdiagramm

**Bootloader-Section** In Abbildung 7.2 wird der Programmfluss der Atmega2560-Software aufgezeigt. Bei einschalten des Mikrocontrollers wird in der Bootloader-Section zuerst der Bootloader gestartet (Boot Loader Flash Section). Sofern keine neue Software über die UART0-Schnittstelle kommt, so beginnt das cocktailspezifische Programm (Application Flash Section). Sofern eine neue Software über die UART0-Schnittstelle gesendet wird, wird die kommende Software in diese Application Flash Section geschrieben und das Programm nach der Übertragung gestartet.

**Init-Section** Wird das Programm gestartet, beginnt die Init-Section. Hier werden die zuerst die Main-Variablen initialisiert. Danach werden die Adressen der Funktionen ins Programm geladen mittels den h-Files. Die Interfaces (SPI, Uart, I/O's) müssen als erstes initialisiert werden, damit die Initialisierung der SPI-Devices stattfinden kann und Boot-Informationen über eine gewünschte Schnittstelle angezeigt werden können (z.B Uart0 ==> Computer). Sobald die Schnittstellen initialisiert wurden, werden die Devices initialisiert. Dazu gehören die SD-Karte, der FOC-Treiber und der Gate-Treiber. Daraufhin werden die Speicher-Strukturen initialisiert, welche für die User-Applikation (Tabelle 7.1 in Kapitel ??) gebraucht werden. Die Struktur und Anwendung der Speicherplätze wird in Kapitel ?? erklärt. Zu guter Letzt wird die Startanzeige des Displays geladen.

**Main-Loop** Da die Maschine nur auf Inputs reagieren muss, werden im Main-Loop nur die Buffer der Devices abgefragt. Sobald ein Terminator-Zeichen ankommt (z.B ein carriage return r der UART0-Schnittstelle oder 0xFF 0xFF 0xFF der UART1-Schnittstelle), werden die zuvor empfangenen Daten interpretiert und verarbeitet.

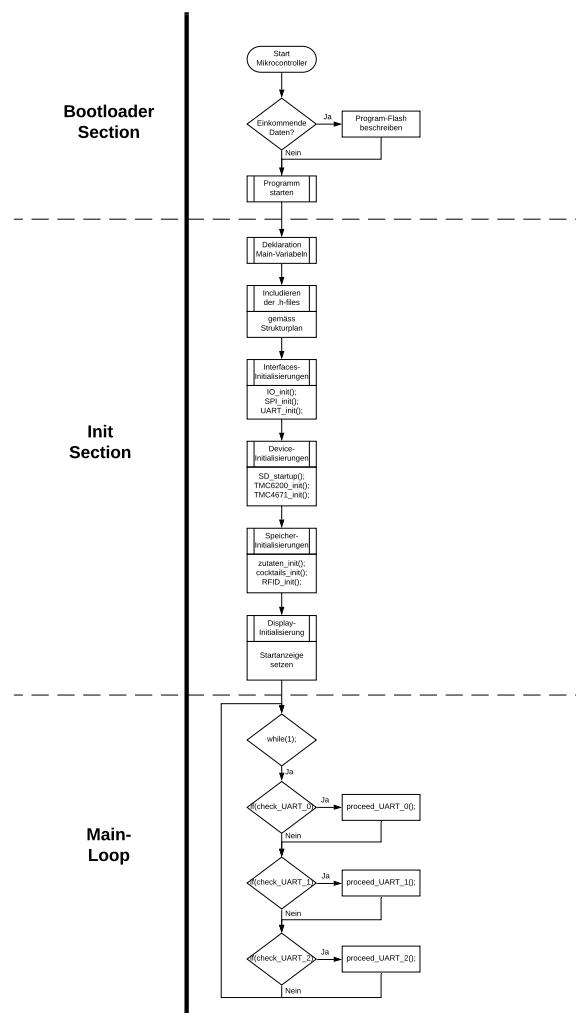


Abbildung 7.2: Programmflussdiagramm Atmega2560.

### 7.1.3 Doubly Dynamic Linked Lists Allgemein

Eine Doppelt verkettete Liste ist eine gängige Datenstruktur, welche verwendet wird, sobald es in der Anwendung wichtig ist, in einer Liste hoch und runter zu bewegen und effizient und ohne grossen Aufwand sogenannte Nodes<sup>19</sup> hinzugefügt werden müssen. Im Folgenden wird von Elementen statt Nodes gesprochen. Die Idee dabei ist, dass für jedes Element zwei Pointer zu initialisieren, welche auf das nächste oder vorhergehende Element zeigen. Nebst den Zeigern auf die umliegenden Elemente enthält das Element die gewünschten Daten. Um zu wissen, ob das Ende oder der Beginn erreicht wurde, werden zwei zusätzliche Zeiger initialisiert, welche auf das erste und/oder letzte Element zeigen (Head und Tail). Um zu wissen, auf welches Element zurzeit gezeigt wird, gibt es einen Zeiger auf dieses Element (actual).

Abbildung 7.3 zeigt eine Grundsätzliche Struktur der beschriebenen Liste mit head, tail und Elementen.

cite Text:  
<https://perlguru.de/tutorials/perl-linked-lists/>

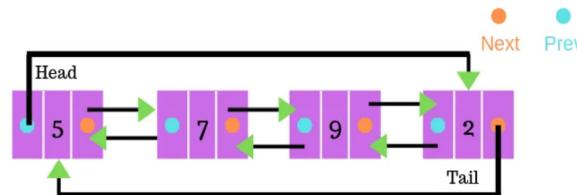


Abbildung 7.3: Doppelt verkettete Liste mit zwei Elementen.

In Abbildung 7.4 wird nur der start-Pointer (head) initialisiert, ohne auf ein Element zu zeigen (NULL). Sobald der benötigte Speicherplatz des Elementes alloziert wurde, wird der Zeiger auf die angelegte Struktur (Element) gelegt. Dem Head-Zeiger wurde nun ein Element zugewiesen und der Beginn der Liste wurde definiert.

cite Bild:  
<https://learncode.academy/structures/circular-doubly-linked-list-in-javascript/>

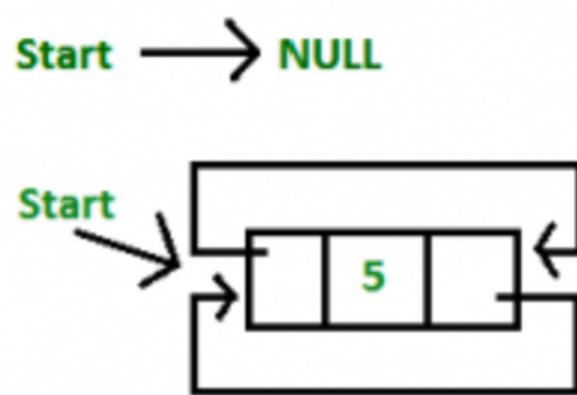


Abbildung 7.4: Doppelt verkettete Liste mit zwei Elementen.

Abbildung 7.5 zeigt eine bestehende Liste mit zwei Elementen. Ein drittes Element soll am Schluss eingefügt werden. Dazu muss zuerst der Speicherplatz für das neue Element alloziert werden (Data, next, prev). Danach können die Zeiger der bestehenden Elemente (in diesem Falle

<sup>19</sup>Nodes = Knoten

cite  
Bild:<https://circular-linked-list-set-1-introduction-and-insertion/>

noch head und tail) umgelegt werden und die Zeiger des neuen Elementes auf das vorhergehende und nächste Element gelegt werden. Auch der tail-Zeiger muss nun auf das neue Element gelegt werden.

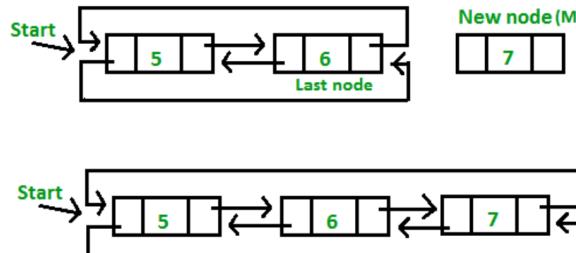


Abbildung 7.5: Doppelt verkettete Liste mit zwei Elementen.

Abbildung 7.6 zeigt ebenfalls eine bestehende Liste mit zwei Elementen. Ein drittes Element soll am Beginn eingefügt werden. Dazu muss zuerst der Speicherplatz für das neue Element alloziert werden (Data, next, prev). Danach können die Zeiger der bestehenden Elemente (in diesem Falle noch head und tail) umgelegt werden und die Zeiger des neuen Elementes auf das vorhergehende und nächste Element gelegt werden. Auch der head-Zeiger muss nun auf das neue Element gelegt werden.

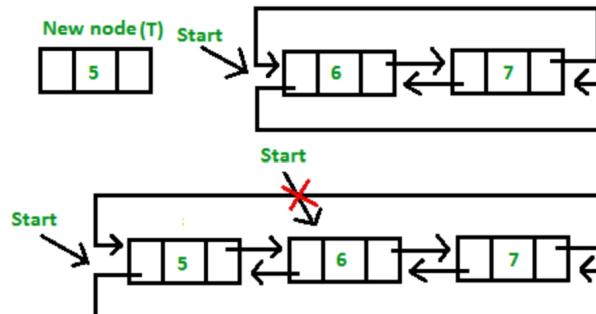


Abbildung 7.6: Doppelt verkettete Liste mit zwei Elementen.

Abbildung 7.7 zeigt eine bestehende Liste mit vier Elementen. Ein fünftes Element soll in der Mitte eingefügt werden. Dazu muss zuerst der Speicherplatz für das neue Element alloziert werden (Data, next, prev). Danach können die Zeiger der bestehenden Elemente umgelegt werden und die Zeiger des neuen Elementes auf das vorhergehende und nächste Element gelegt werden.

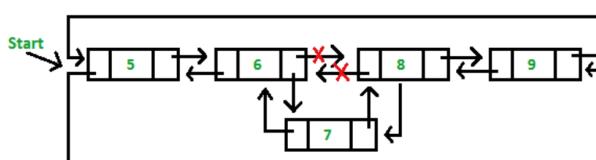


Abbildung 7.7: Doppelt verkettete Liste mit zwei Elementen.

### 7.1.4 Doubly Dynamic Linked Lists Cocktailmixer

Im Falle des Cocktailmixers werden verschiedene Listen verwendet. Dazu gehören die folgenden Elemente:

- Cocktail-Liste (Files)
- Zutaten-Liste (Files)
- Tag-Liste (Maschine)
- Zutaten-in-Maschine-Liste (Maschine)

Die Cocktail- und die Zutaten-Liste beinhalten nur die Nummer des Files, welches sich auf der SD-Karte befindet. Wird ein Cocktail oder eine Zutat benötigt, so werden die Daten temporär von der SD-Karte in einen Struct im Programmspeicher geladen. Dies wird gemacht, da während der Entwicklung der Programmspeicher mit zu vielen Daten gefüllt wurde, was zur Folge hatte, dass der Mikrocontroller abstürzte. Im Vergleich zum kompletten Cocktail, welcher ca. 80 Bytes Speicher braucht, benötigt das File nur eines. Ein einziger Struct, welcher immer neu geladen wird, reicht aus, um die benötigten Informationen zu extrahieren und verwenden. Jeder Cocktail und jede Zutat wird separat auf der SD-Karte gespeichert.

Die Tag-Liste sowie die Liste der Zutaten in der Maschine werden komplett in den Programmspeicher geladen. Die beiden Listen werden bei einer Veränderung als Backup jeweils in einem File gespeichert, damit diese bei einem Neustart der Maschine wieder in den Programmspeicher geladen werden können.

## 7.2 ESP32

Bei der Inbetriebnahme in Kapitel 5.5.2 wurde das ESP32 mittels eines Web-Servers getestet und in Betrieb genommen. Da jedoch dies für einen Benutzer relativ umständlich ist, wurde eine Android-App erstellt, mit welcher Befehle an das ESP32 gesendet werden können. Diese kommuniziert über Bluetooth und hat den Vorteil, dass man sich nicht jedes Mal in einem neuen Netz anmelden muss und sich somit auch nicht vom Heimnetz trennen muss. Man ist also trotzdem immer mit dem Heiminternet verbunden. Ein weiterer grosser Vorteil ist, dass eine App einiges Benutzerfreundlicher ist und somit einem auch mehr Spass bereitet.

Die Software für das ESP32 wurde komplett in Arduino IDE geschrieben und beinhaltet folgende Bereiche:

- Daten von der App empfangen
- Daten an die App senden
- Daten vom uP abfragen
- Daten an den uP senden
- Daten vom RFID-Leser empfangen

Im Programm werden zuerst die notwendigen Bibliotheken eingebunden. Dabei werden folgende Bibliotheken verwendet:

- Arduino.h
- BluetoothSerial.h
- SPI.h
- MFRC522.h

Dabei beinhaltet die «Arduino.h» Bibliothek die Standardbefehle von Arduino, die «Bluetooth-Serial.h» Bibliothek beinhaltet alle Kommunikationsbefehle um über Bluetooth kommunizieren zu können, die «SPI.h» Bibliothek beinhaltet alle Kommunikationsbefehle um über SPI kommunizieren zu können und die «MFRC522.h» Bibliothek beinhaltet alle Befehle um mit dem RFID-Modul arbeiten zu können.

Es wurde bei der Programmierung darauf geachtet, dass mit möglichst einfachen Mitteln gearbeitet wird. Um zu kommunizieren wurden die Datentypen String und char verwendet. Für Zählvariablen wurde der Datentyp int verwendet.

### **7.2.1 Programmflussdiagramm**

## **7.3 Displaysoftware**

## **7.4 Android App**

# **8 Evaluation**

# **9 Fazit**

## **9.1 Zielerreichung**

## **9.2 Kosten**

# **10 Schlusswort**

# **11 Ehrlichkeitserklärung**

Mit der Unterschrift bestätigt der Unterzeichnende Projektleiter, dass die vorliegende Projektdokumentation selbstständig im Team und ohne Verwendung anderer, als der angegebenen Hilfsmittel verfasst wurde, sämtliche verwendeten Quellen erwähnt und die gängigen Zitierregeln eingehalten wurden. Eine Überprüfung der Arbeit auf Plagiate mithilfe elektronischer Hilfsmittel darf vorgenommen werden.

Unterschrift:

Ort, Datum:

---

---

## Literatur

- [1] N. B. 2010, Antenna design guide for MFRC52x, PN51x and PN53x, Okt. 2010. Adresse: <https://my.eng.utah.edu/~mlewis/ref/NFC/AN1445.pdf>.

## A TMC4671

### A.1 Standard-Schaltkreis TMC4671

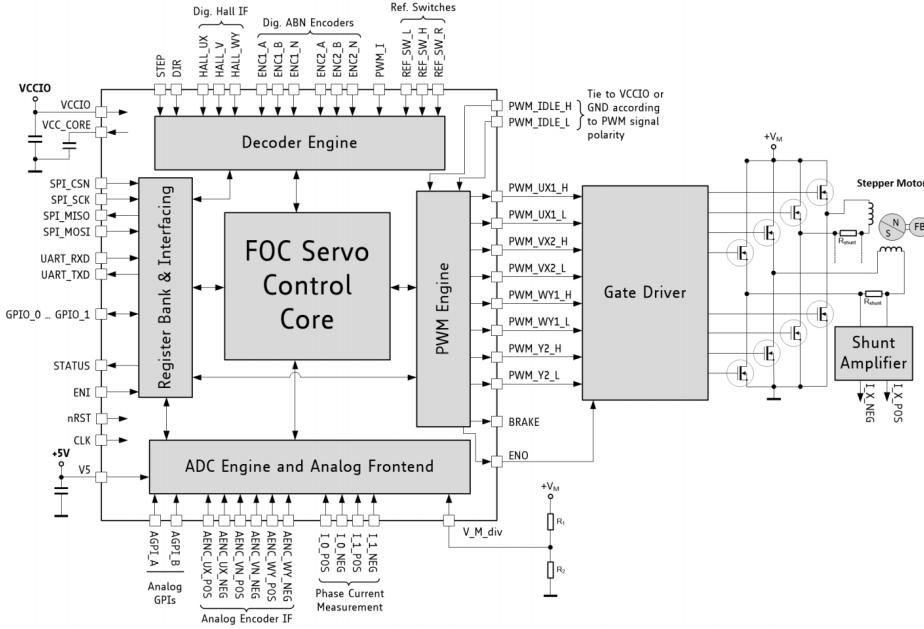


Abbildung A.1: Standard-Anwendungs-Schaltung.

citeTMC467  
Datenblatt

### A.2 Blockdiagramm TMC4671

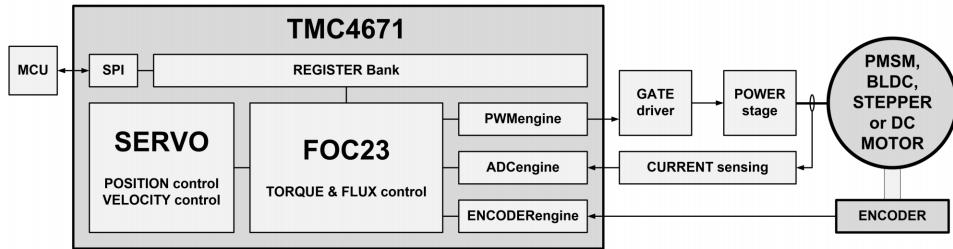


Abbildung A.2: Blockdiagramm TMC4671.

citeTMC467  
Datenblatt

### A.3 Inbetriebnahme Gate-Control

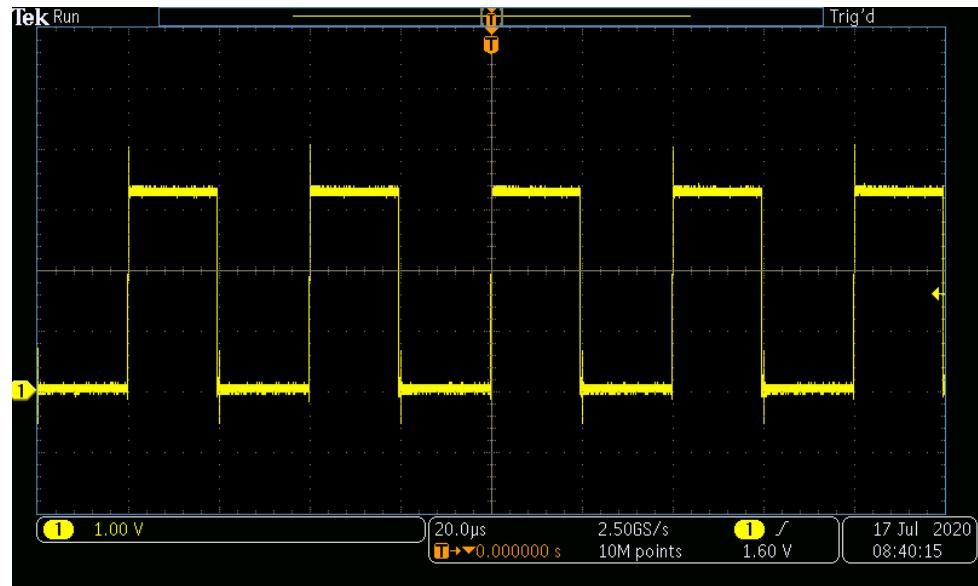


Abbildung A.3: Steuersignal PWM\_UX1\_H

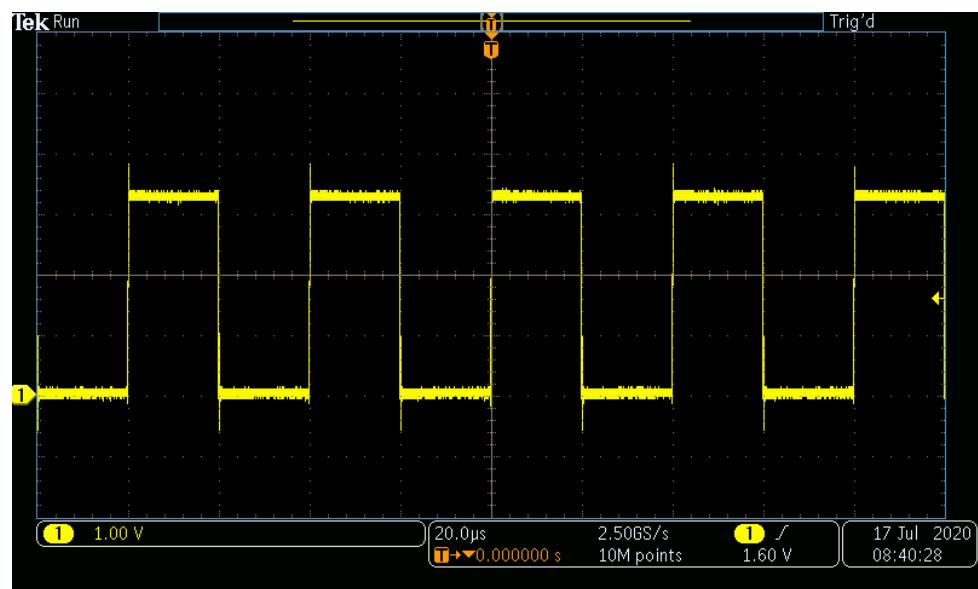


Abbildung A.4: Steuersignal PWM\_UX1\_L

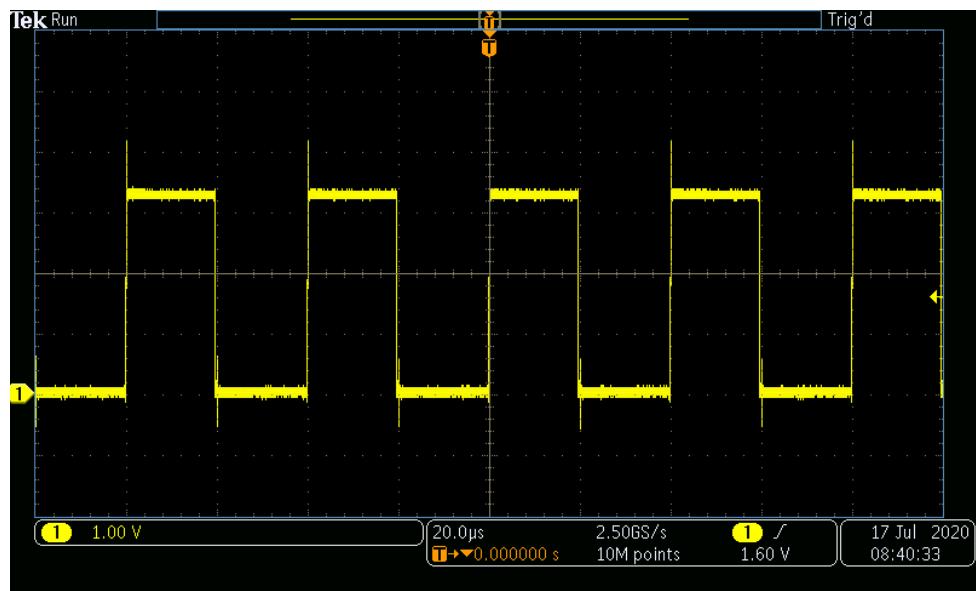


Abbildung A.5: Steuersignal PWM\_UX2\_H

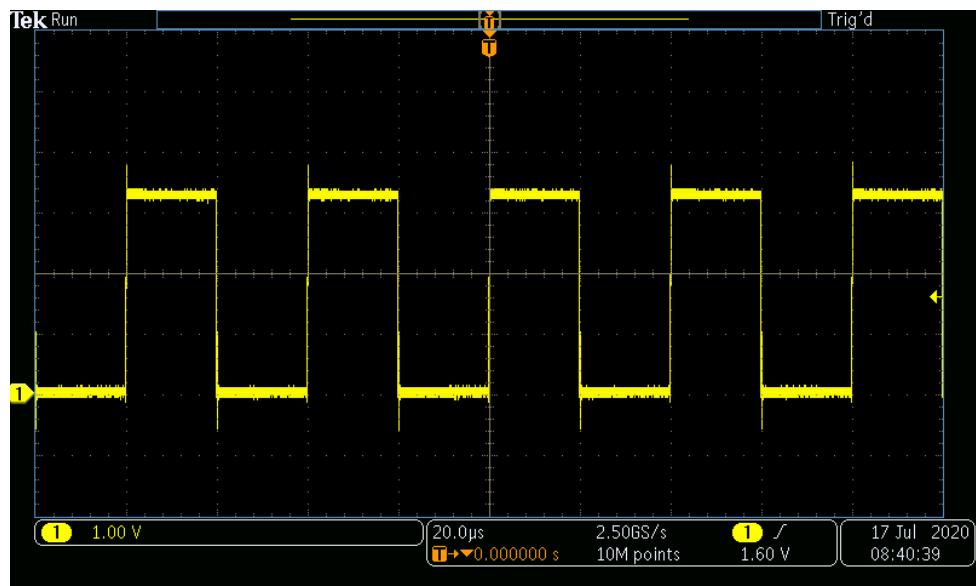


Abbildung A.6: Steuersignal PWM\_UX2\_L

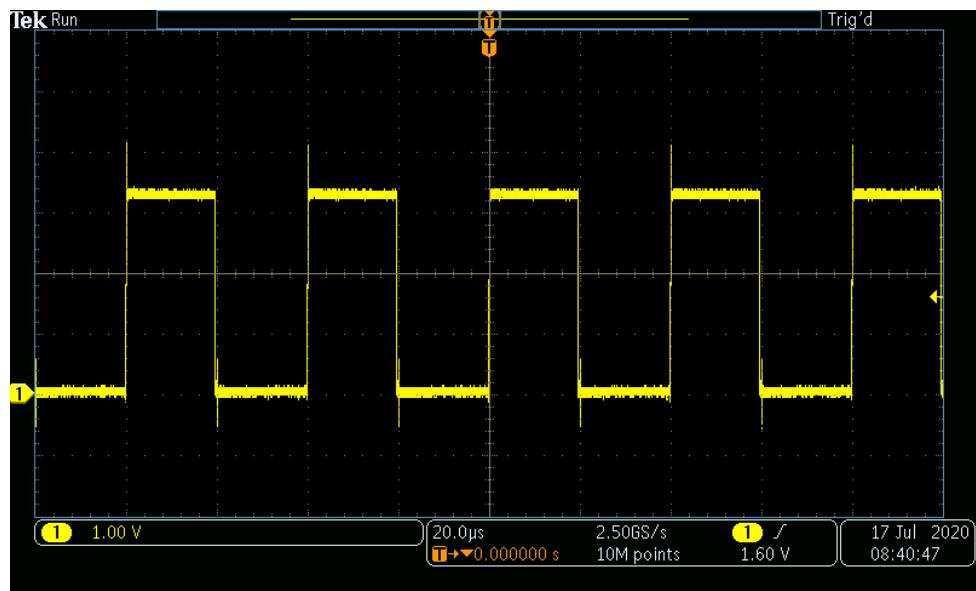


Abbildung A.7: Steuersignal PWM\_UX3\_H

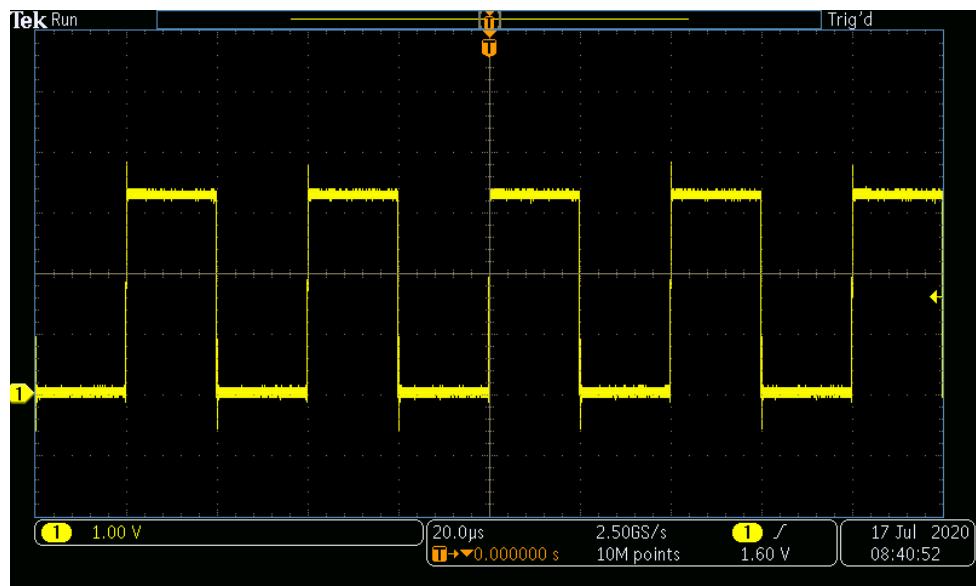


Abbildung A.8: Steuersignal PWM\_UX3\_L

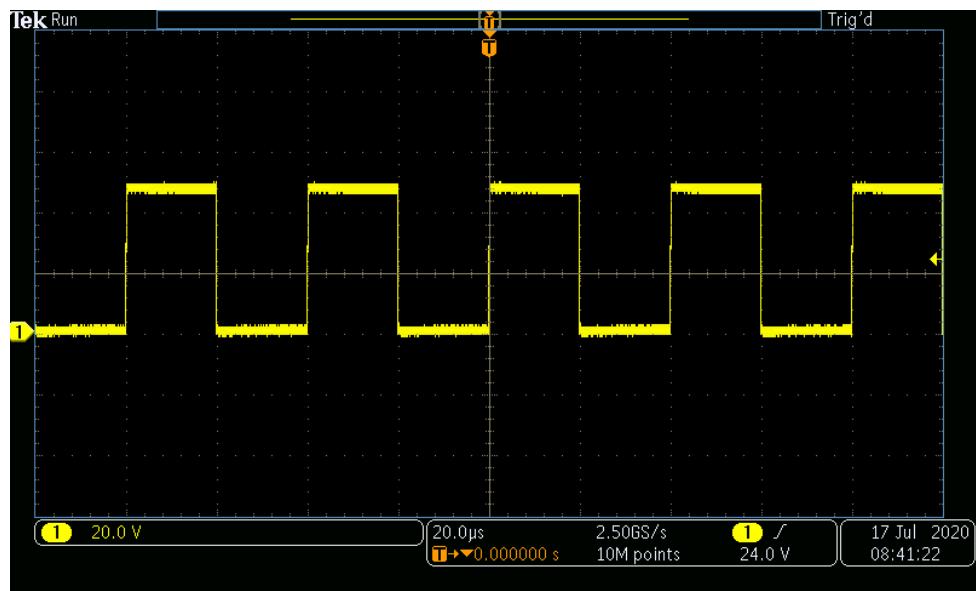


Abbildung A.9: Motorsignal U nach H-Brücke

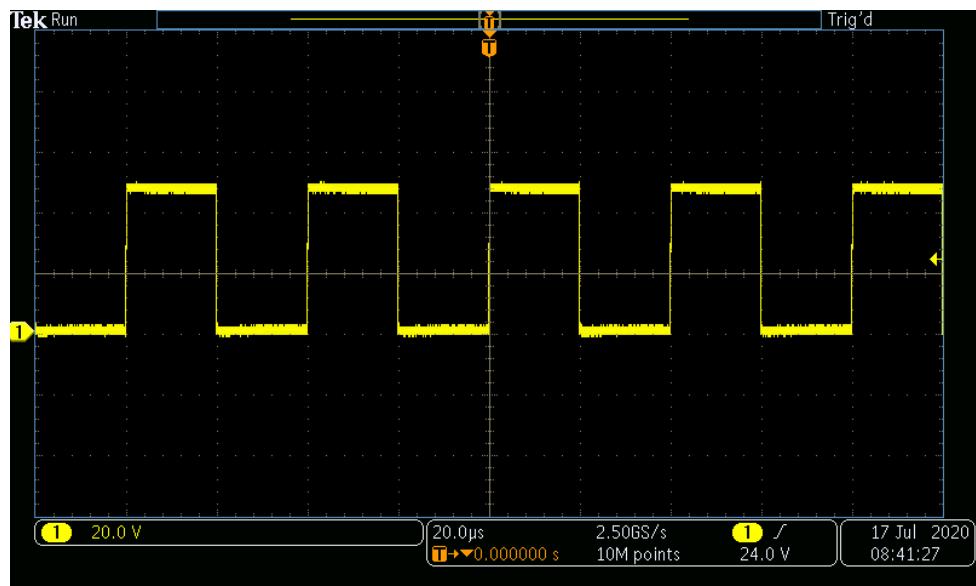


Abbildung A.10: Motorsignal U nach H-Brücke

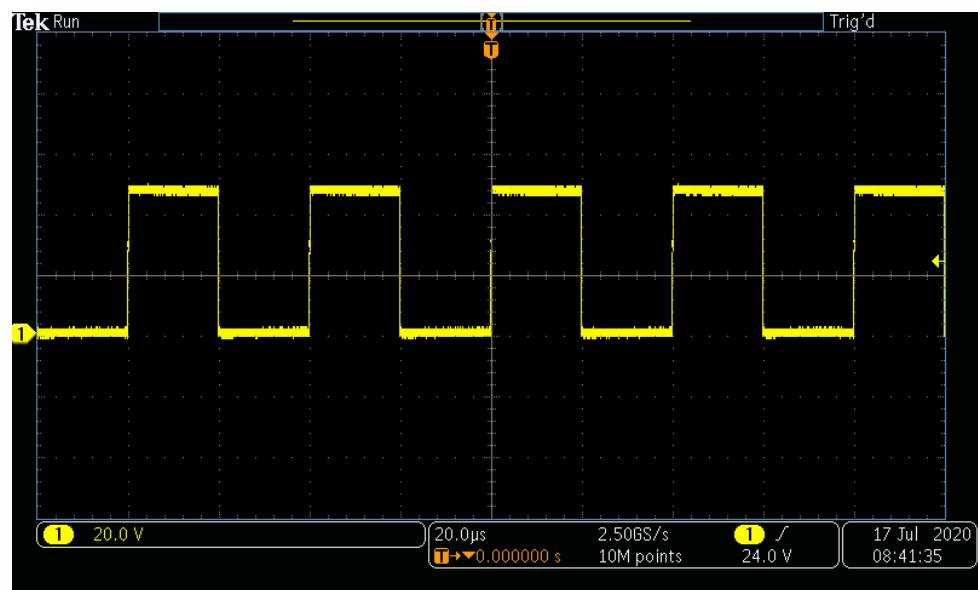


Abbildung A.11: Motorsignal U nach H-Brücke

## B TMC6200

### B.1 Standard-Schaltkreis TMC6200

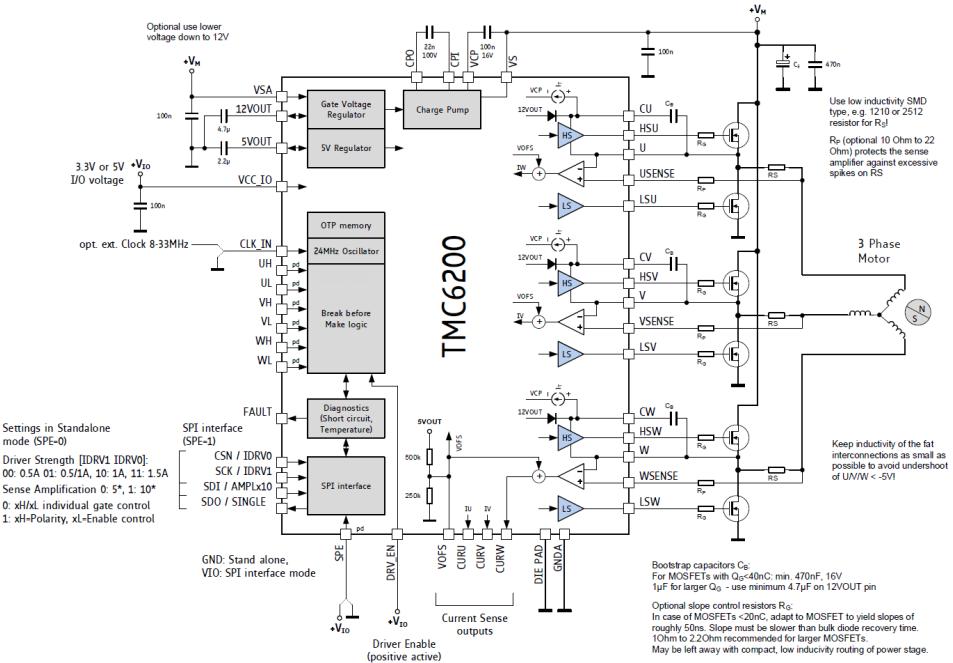


Abbildung B.1: Standard-Anwendungs-Schaltung TMC6200.

### B.2 Blockdiagramm TMC6200

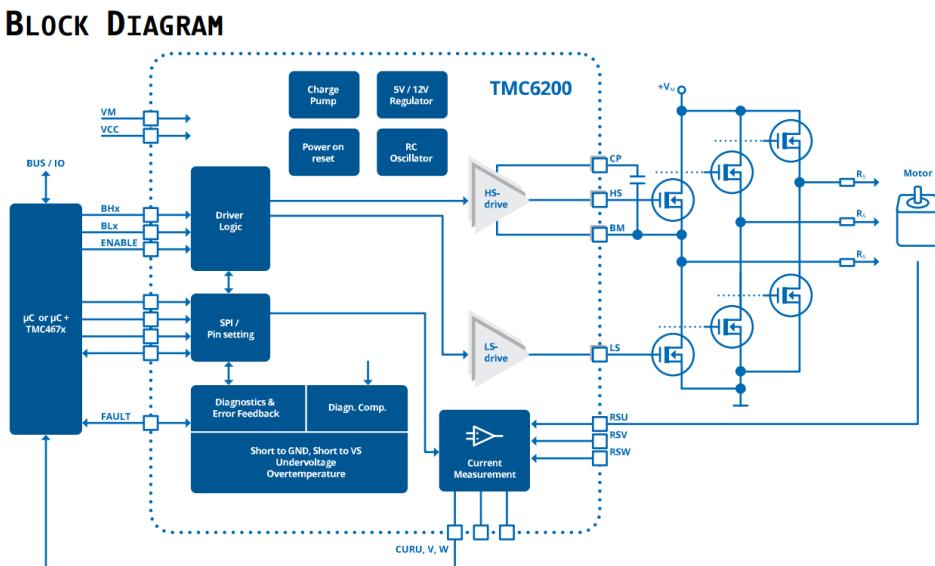


Abbildung B.2: Blockdiagramm TMC6200.

### B.3 Verstärkungsfaktor, Strommessung, Strommesswiderstand

CHOICE OF $R_{SENSE}$ AND AMPLIFICATION DEPENDING ON MAX. COIL CURRENT				
$R_{SENSE}$ [ $m\Omega$ ]	Amplification factor	Current range [A]	RMS motor current limit [A]	Max. power dissipation of $R_{SENSE}$ [W]
150	10	0.7	0.5	0.05
150	5	1.3	1	0.15
100	5	2	1.5	0.23
75	5	2.6	2	0.3
33	10	3	2.2	0.16
25	10	4	3	0.23
50	5	4	3	0.45
33	5	6	4.5	0.67
15	10	6.5	5	0.38
25	5	8	6	0.9
10	10	10	7.5	0.56
5	10	20	15	1.1
2.5	20	20	15	0.56
2.5	10	40	30	2.3
1	20	50 (40@1.65V ofs.)	37	1.4

**Abbildung B.3:** Tabelle zur Bestimmung des Strommesswiderstandes aus dem Datenblatt von Trinamic.

### B.4 Gate-Vorwiderstand

MOSFET MILLER CHARGE VS. DRVSTRENGTH AND $R_G$		
Miller Charge [ $nC$ ] (typ.)	DRVSTRENGTH setting	Value of $R_G$ [ $\Omega$ ]
<10	0 or 1	$\leq 10$ (recommended)
10...20	0 to 2	$\leq 5$ (optional)
20...80	1 to 3	$\leq 2.5$ (optional)
>80	3	$\leq 1$ (optional)

**Abbildung B.4:** Tabelle zur Bestimmung der Gatewiderstände aus dem Datenblatt von Trinamic.

### B.5 Externe Gate-Spannungsversorgung

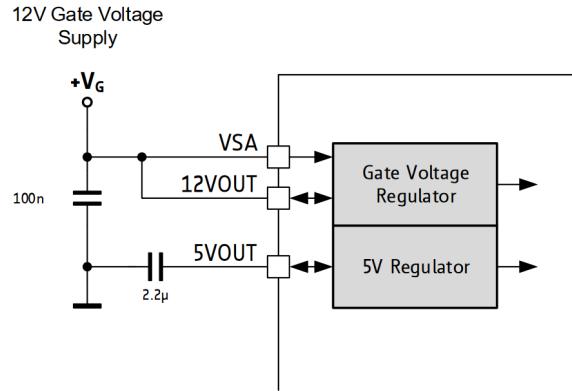


Abbildung B.5: Schema externe Gate-Spannungsversorgung.

## C H-Brücke

### C.1 Referenzschema

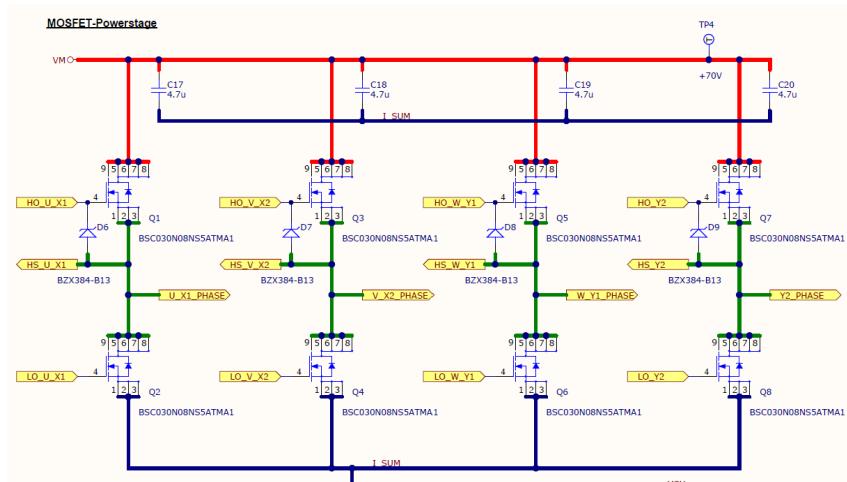


Abbildung C.1: H-Brücke.

## D Mikrocontroller

### D.1 Brown-out-Detection

cit: Datenblatt UPS 10A70V Schema

### D.2 Full Swing Crystal Oscillator

cit: Datenblatt Atmega 2560, Seite 361

cit: Datenblatt Atmega 2560, Seite 43

BODLEVEL 2:0 Fuses	Min. V <sub>BOT</sub>	Typ. V <sub>BOT</sub>	Max. V <sub>BOT</sub>	Units	
111	BOD Disabled				
110	1.7	1.8	2.0	V	
101	2.5	2.7	2.9		
100	4.1	4.3	4.5		
011	Reserved				
010					
001					
000					

Abbildung D.1: Tabelle Brown-out-Detection.

Frequency Range [MHz]	CKSEL3:1	Recommended Range for Capacitors C1 and C2 [pF]
0.4 - 16	011	12 - 22

Abbildung D.2: Tabelle Frequenzbereich Crystal Oszillatior.

Oscillator Source / Power Conditions	Start-up Time from Power-down and Power-save	Additional Delay from Reset (V <sub>CC</sub> = 5.0V)	CKSEL0	SUT1:0
Ceramic resonator, fast rising power	258 CK	14CK + 4.1ms <sup>(1)</sup>	0	00
Ceramic resonator, slowly rising power	258 CK	14CK + 65ms <sup>(1)</sup>	0	01
Ceramic resonator, BOD enabled	1K CK	14CK <sup>(2)</sup>	0	10
Ceramic resonator, fast rising power	1K CK	14CK + 4.1ms <sup>(2)</sup>	0	11
Ceramic resonator, slowly rising power	1K CK	14CK + 65ms <sup>(2)</sup>	1	00
Crystal Oscillator, BOD enabled	16K CK	14CK	1	01
Crystal Oscillator, fast rising power	16K CK	14CK + 4.1ms	1	10
Crystal Oscillator, slowly rising power	16K CK	14CK + 65ms	1	11

Abbildung D.3: Tabelle Aufstartzeit.

### D.3 Bootloader-Speicherplatz

cite: Datenblatt Atmega 2560, Seite 43

BOOTSZ1	BOOTSZ0	Boot Size	Pages	Application Flash	Boot Loader Flash Section	End Application Section	Boot Reset Address (Start Boot Loader Section)
1	1	512 words	4	0x0000 - 0x7FFF	0x7E00 - 0x7FFF	0x7FFF	0x7E00
1	0	1024 words	8	0x0000 - 0x7BFF	0x7C00 - 0x7FFF	0x7FFF	0x7C00
0	1	2048 words	16	0x0000 - 0x77FF	0x7800 - 0x7FFF	0x7FFF	0x7800
0	0	4096 words	32	0x0000 - 0x6FFF	0x7000 - 0x7FFF	0x6FFF	0x7000

Abbildung D.4: Tabelle Bootloader Speicherplatz.

### D.4 Memory-Lock Bootloader

cite: Datenblatt Atmega 2560, Seite 320

cite: Datenblatt Atmega 2560, Seite 326

Memory Lock Bits			Protection Type
BLB1 Mode	BLB12	BLB11	
1	1	1	No restrictions for SPM or (E)LPM accessing the Boot Loader section.
2	1	0	SPM is not allowed to write to the Boot Loader section.
3	0	0	SPM is not allowed to write to the Boot Loader section, and (E)LPM executing from the Application section is not allowed to read from the Boot Loader section. If Interrupt Vectors are placed in the Application section, interrupts are disabled while executing from the Boot Loader section.
4	0	1	(E)LPM executing from the Application section is not allowed to read from the Boot Loader section. If Interrupt Vectors are placed in the Application section, interrupts are disabled while executing from the Boot Loader section.

Abbildung D.5: Tabelle Memory Lock.

## E USB-B

### E.1 Geräte-Manager

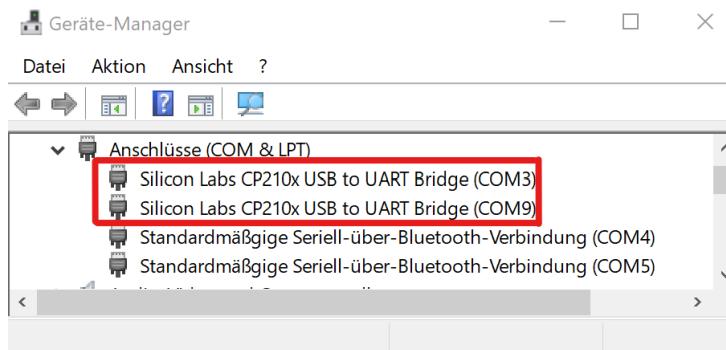


Abbildung E.1: Geräte-Manager mit den aufgelisteten USB-UART-Converter (Mikrocontroller und WiFi-Modul).

## F Atmel Studio

### F.1 Fuse Bits

### F.2 Lock Bits

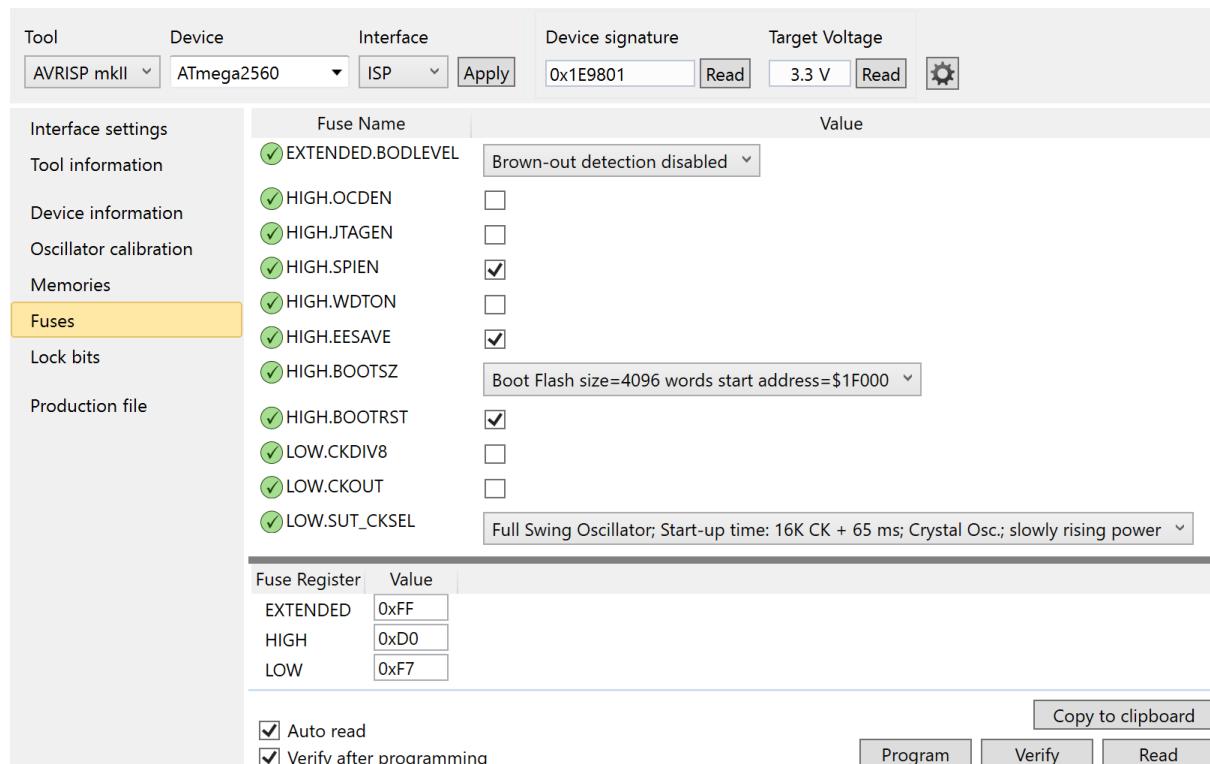


Abbildung F.1: Fuse-Bits Atmega2560.

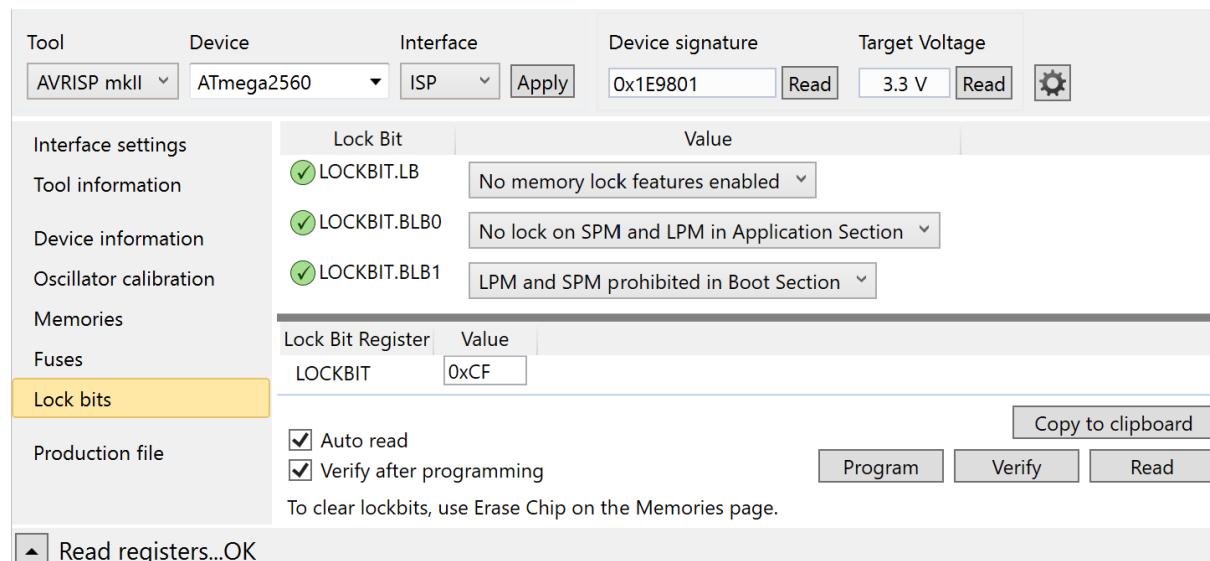


Abbildung F.2: Lock-Bits Atmega2560.

### F.3 Einbinden AVRdude und stk500v2 (wiring)

### F.4 Bootloader "Brennen"

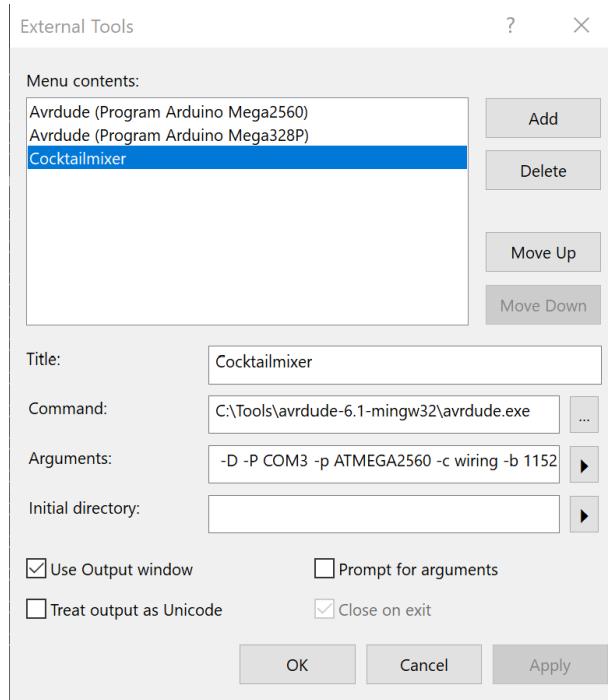


Abbildung F.3: External Tools Atmega2560.

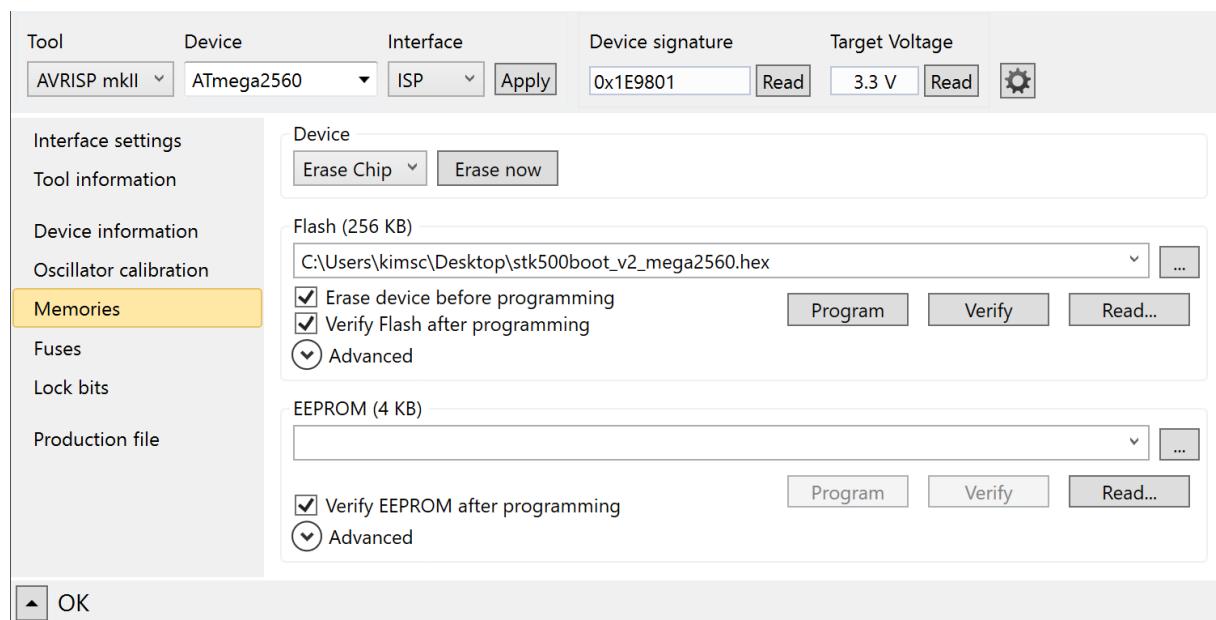


Abbildung F.4: Bootloader brennen.