

ESP32-Modul:

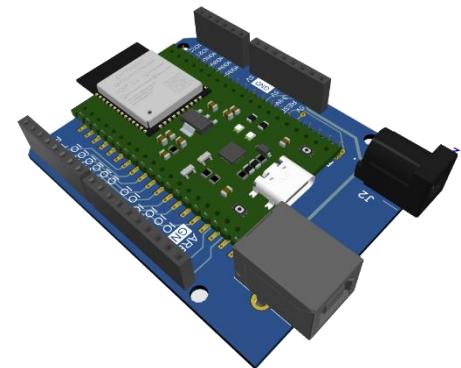
- ESP32 Mikrocontroller:
 - 32-Bit dual-Core Mikroprozessor (bis zu 240MHz)
 - 448kB ROM
 - 520kB SRAM
 - 16kB SRAM in RTC
 - Wi-Fi Modul (2.4 GHz) mit onboard PCB-Antenne
 - Bluetooth Modul
 - integrierter 40 MHz Quarzoszillator
 - 26 GPIOs
- USB-C to UART converter
- Enable- & Boot-Button für einfaches Debugging
- Enable- & Boot-Pin über RTS & CTS steuerbar
- 5V zu 3.3V DCDC-Wandler
- Standard ESP-Pinout mit Ausführung aller Pins
 - + castellated Holes für Entwicklungsumgebungen mit Flexi-Pins

ESP32-Arduino-Uno-Expander:

- Erweiterungsmodul für ESP32-Modul
- ermöglicht die Verwendung eines ESP32-Moduls für Arduino-Uno Schnittstellen
- zusätzliche USB-B und DC-Buchse zur Stromversorgung

ESP32-Devboard:

- Entwicklungsmodul für ESP32-Modul
- ermöglicht schnelle Realisierung von Hardware- & Software Konzepten mit dem ESP32
- Ausführung aller Pins für den Anschluss mit Jumper-Wires
- Standard Schnittstellen-Pins umschaltbar für externen Anschluss mit Jumper-Wires
- vorimplementierte Module:
 - I²C-Module:
 - EEPROM (24LC02BT)
 - 12-Bit-ADC (MCP3231)
 - 10-Bit-DAC (TC1321) + ext. Vref (MCP1525T)
 - 8-Bit-I/O-Expander (MCP23008)
 - Display (NHD-CO216CIZ-FSW-FBW-3V3)
 - SPI-Module:
 - 12-Bit-DAC (MCP4921T)
 - 12-Bit-ADC (MCP3201T)
 - Motortreiber (L298N Doppel-H-Brücke)
 - 4x invertierter FET-Ausgangstreiber



**ESP32-Devboard
mit
eigenem ESP32-Modul**

Dokumentation zum HWE-Projekt „ESP32-Devboard“ und allen dazugehörigen Modulen:

ESP32-Modul

+

**ESP32 Arduino-Uno
Expander**

+

ESP32-Devboard

Inhalt

ESP32-Modul:	5
Komponenten:	5
USB-C:	5
USB-UART-Converter:	6
Taster:	6
Spannungsversorgung:	7
Header:	7
ESP-32:	7
Schaltplan:	9
Boardplan:	10
Top:	10
Bottom:	11
Holes:	12
Bestückungsplan:	13
Bill of Materials:	14
ESP32 Arduino Uno Expander:	15
Arduino Uno Pinout:	15
Schaltplan:	16
Boardplan:	17
Top:	17
Bottom:	18
Holes:	19
Bestückungsplan:	20
Bill of Materials:	21
ESP32-Devboard:	22
Hardware:	22
Pin-Header:	22
Leistungselektronik:	22
I ² C-Module:	23
SPI-Module:	25
Schaltplan:	27
Connections:	27
I2C:	28
SPI:	29
Power:	30
Boardplan:	31
Top:	31

Bottom:	32
Holes:	33
Bestückungsplan:	34
Bill of Materials:	35
Beispielprogramme:	37
Display:	37
I/O-Expander:	39
I ² C-ADC:	40
Driver:	40
ESP32-Modul WLAN + Display:	41
Referenzen:	43
Projektplan:	44

Abbildungsverzeichnis

Abbildung 1 - ESP32-Modul: USB-C.....	5
Abbildung 2 - ESP32-Modul: USB Type-C	5
Abbildung 3 - ESP32-Modul: USB-UART Converter	6
Abbildung 4 - ESP32-Modul: Taster.....	6
Abbildung 5 - ESP32-Modul: Spannungsregler 5V-3V3	7
Abbildung 6 - ESP32-Modul: ESP-WROOM-32	7
Abbildung 7 - ESP32-Modul: ESP32-Chip + Peripherie	8
Abbildung 8 - ESP32-Devboard: Pinheader	22
Abbildung 9 - ESP32-Devboard: invertierter FET-Schalter	22
Abbildung 10 - ESP32-Devboard: L298N Vollbrücken-Treiber	23
Abbildung 11 - ESP32-Devboard: L298N Pinout.....	23
Abbildung 12 - ESP32-Devboard: IIC 24LC02-EEPROM	23
Abbildung 13 - ESP32-Devboard: IIC MCP3221-ADC.....	24
Abbildung 14 - ESP32-Devboard: IIC TC1321-DAC	24
Abbildung 15 - ESP32-Devboard: IIC MCP23008-I/O-Expander	24
Abbildung 16 - ESP32-Devboard: IIC NHD-Display	25
Abbildung 17 - ESP32-Devboard: SPI-Switch.....	25
Abbildung 18 - ESP32-Devboard: SPI MCP4921-DAC.....	25
Abbildung 18 - ESP32-Devboard: SPI MCP4921-DAC.....	26

ESP32-Modul:

Komponenten:

USB-C:

USB-C

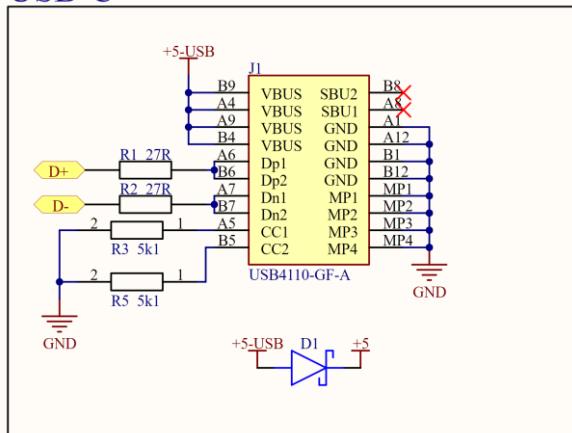


Abbildung 1 - ESP32-Modul: USB-C

Die Programmierung und Spannungsversorgung des Boards werden mit einem USB-C Stecker realisiert. Dabei müssen zwei Sachen unbedingt beachtet werden, um Kompatibilitätsprobleme zu vermeiden:

- Der UFP (Upstream Facing Port, Empfänger der Leistung) muss CC1 (Control Channel) und CC2 über jeweils einen 5k1 Widerstand auf GND ziehen. Damit wird dem DFP (Downstream Facing Port, Absender der Leistung) signalisiert, dass Leistung bzw Spannung benötigt wird.

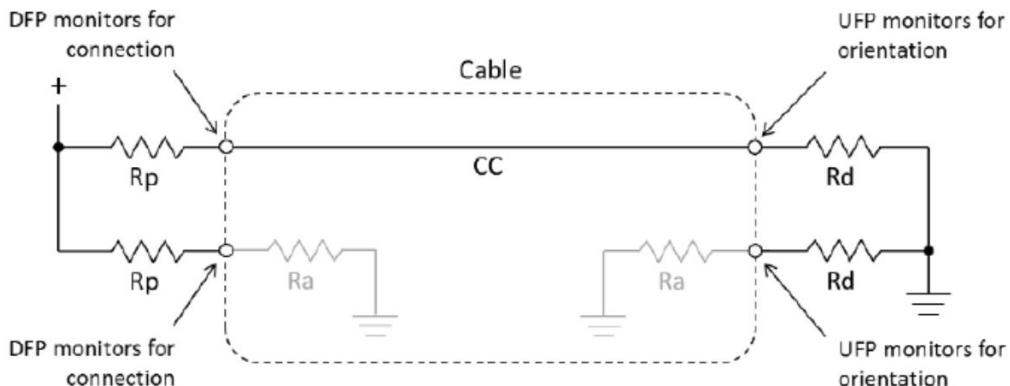


Abbildung 2 - ESP32-Modul: USB Type-C

Diese Schaltung kann bis zu 3A vom DFP ziehen. Wenn mehr Strom als maximal 3A@5V benötigt wird, muss ein USB-PD (Power Delivery) verwendet werden. Damit können bis zu 5A@48V (240W) empfangen werden. Dafür muss dann aber ein USB-PD Controller verwendet werden, denn DFP und UFP müssen zuerst miteinander verhandeln, wie viel Leistung vom DFP benötigt wird, und wie viel vom UFP maximal zur Verfügung gestellt werden kann. Außerdem werden die CC-Leitung zur Erkennung der Orientierung des Kabels benötigt, denn USB-C verfügt über eine zweifache Rotationssymmetrie.

- D+ und D- (Data Lines) müssen über einen 27R Widerstand laufen. Dies wird im Datenblatt des UART-Converters empfohlen, und auch in vielen anderen Datenblättern von ICs mit USB-Verbindung. FTDI empfiehlt 27R, Maxim Integrated (Analog Devices) 33R und Cypress (Infineon) 24R. Diese werden verwendet, um die Gesamtimpedanz der Leitung auf 90R zu bringen, wie vom USB-Standard gefordert. Idealerweise sollten also die Traces auch 90R Wellenwiderstand haben, dies wird aber erst bei höheren Geschwindigkeiten relevant.

Die Diode wird benötigt, um im Falle einer externen Spannungsversorgung des Boards zu verhindern, dass zwischen den von USB bereitgestellten 5V und den externen 5V Ausgleichsströme fließen. Diese fallen in der Regel extrem hoch aus (mehrere Ampere!) und führen zur Zerstörung. Darüber fällt zwar eine kleine Spannung ab, was aber kein Problem ist, da später sowieso mit einem LDO runtergeregt wird.

SBU1 und SBU2 (Secondary Bus) werden in unserem Fall nicht benötigt. Über diese kann z.B. Audio separat gesendet werden, wenn über die Datenleitungen Video läuft. Wir schließen sie also nicht an.

USB-UART-Converter:

USB-UART Converter

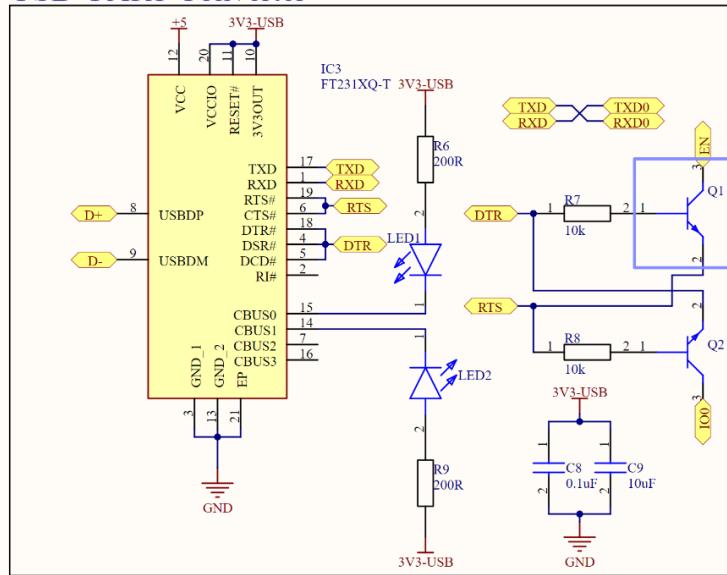


Abbildung 3 - ESP32-Modul: USB-UART Converter

Um den ESP32 flashen zu können, muss das USB-Signal in ein UART-Signal umgewandelt werden. Dafür haben wir einen FT231X von FTDI gewählt. Dieser verfügt über einen vollen Handshake-Prozess, welcher später noch wichtig wird. Außerdem gibt es 4 konfigurierbare CBUS-Pins. Darüber werden bei uns 2 LEDs angesteuert, welche für TX und RX stehen. Man könnte auch einen eigenen Treiber schreiben (theoretisch, wenn man das Geld für die Signatur übrig hat) und Daten bitbangen.

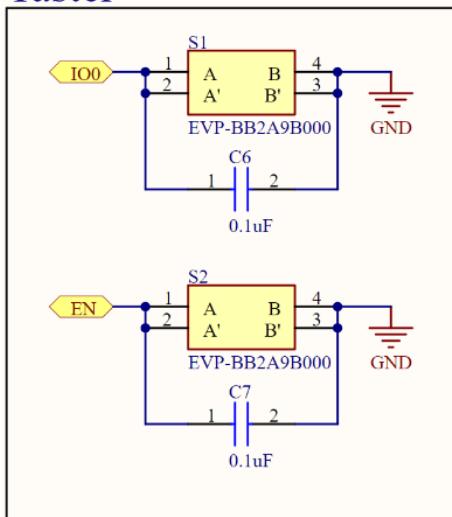
Der Converter hat einen eingebauten LDO auf 3.3V, und versorgt sich da-

mit bei uns selbst.

Die Transistorschaltung ist dafür da, um den ESP32 automatisch in den Bootloader zu setzen. Der ESP geht in den Bootloader, wenn GPIO0 beim Reset Low ist. ESPtool ist automatisch so eingerichtet, dass es versucht, den ESP32 über RTS und CTS in den Bootloader zu bringen.

Taster:

Taster



Falls der Boot in den Bootloader über RTS und CTS nicht funktioniert, kann er über die Taster erzwungen werden. Dabei müssen die Reset- und Boot-Taster gehalten werden. Dann kann der Reset losgelassen werden. Der ESP32 müsste dann in den Bootloader booten, und kann programmiert werden.

Abbildung 4 - ESP32-Modul: Taster

Spannungsversorgung: Spannungsregler 5V→3V3

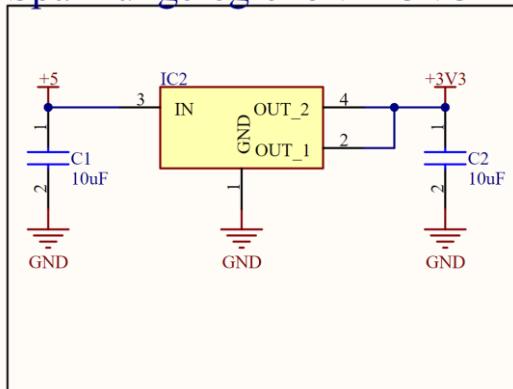


Abbildung 5 - ESP32-Modul: Spannungsregler 5V-3V3

Da der ESP32 nur mit 3.3V funktioniert, und USB im normalen Modus 5V liefert, muss die Spannung runtergeregt werden. Da der ESP32 nur relativ wenig Strom zieht, kann dies über einen LDO (Low Dropout Regler) gemacht werden. Dies ist auch „sicherer“ als ein Buck-Converter, denn dieser könnte eine stark verrauschte Spannung liefern, was für einen Mikroprozessor sehr ungünstig wäre

Header:

Die Pinheader wurden so angeordnet, dass die normalen ESP32-Devboards entsprechen. Damit ist unser Board 100% Pinkompatibel mit dem Original.

ESP-32:

Das Herzstück des Boards bildet der ESP32. Der Chip alleine braucht noch einige Peripherie, vor allem einen EEPROM, um den Programmcode zu speichern, um zu funktionieren. Es gibt aber auch fertige Module zu kaufen, welche diese Peripherie bereits beeinhalten. Diese sehen dann so aus:



Abbildung 6 - ESP32-Modul: ESP-WROOM-32

Das RF-Shield hat den Zweck, die darunter liegende, teils empfindliche Elektronik vor Störeinflüssen zu schützen. Wenn man dieses ablötet, sieht man was auf dem Modul alles verbaut wurde:

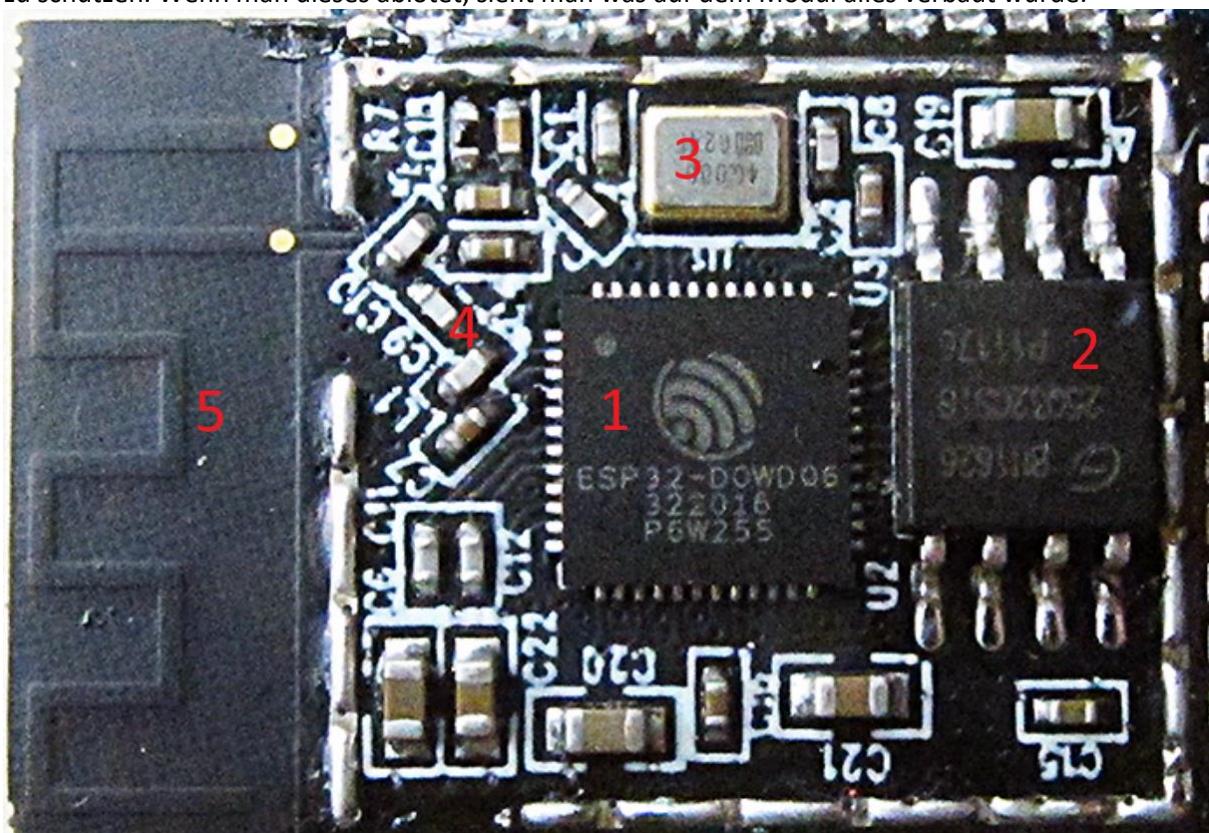
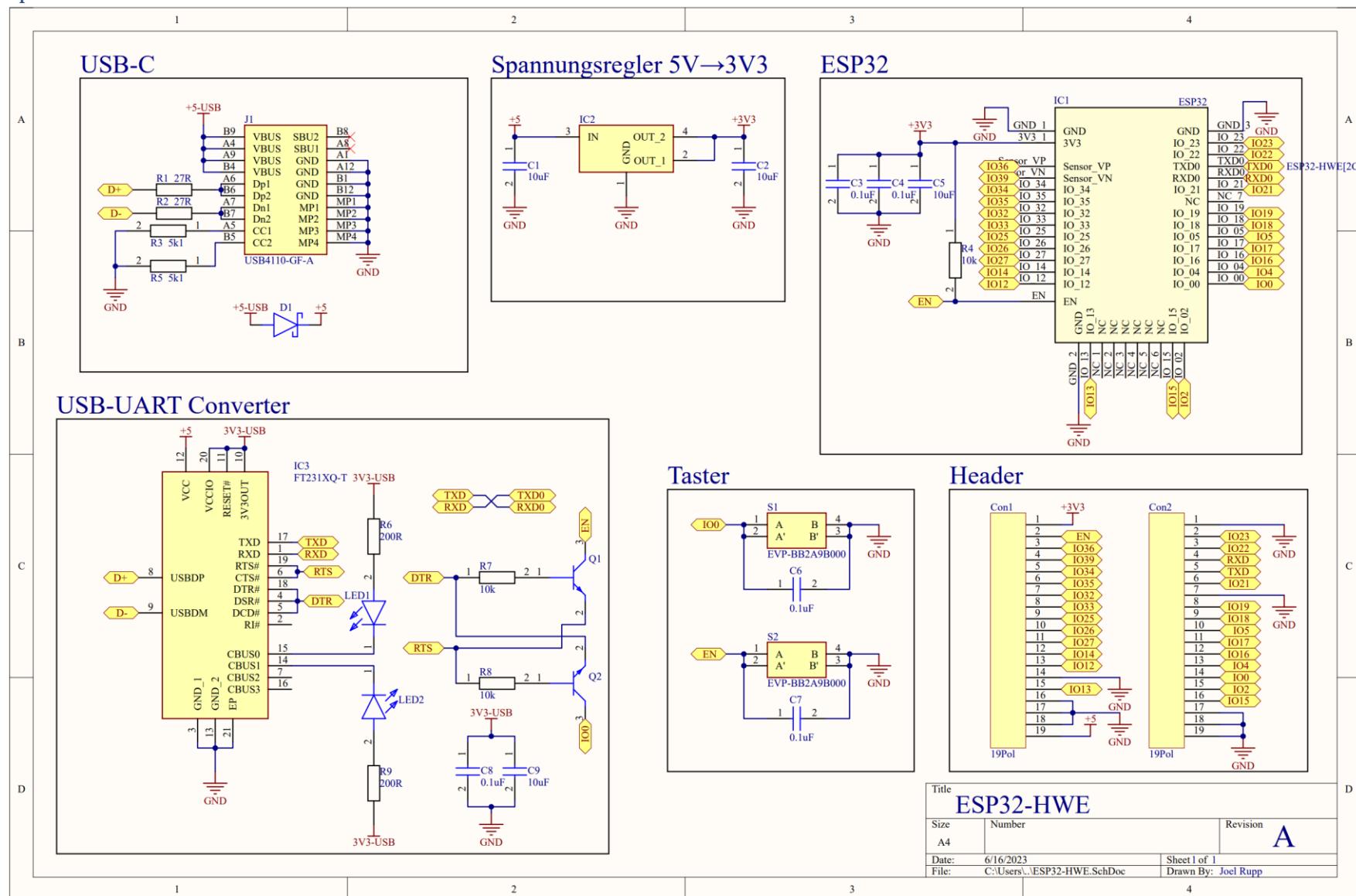


Abbildung 7 - ESP32-Modul: ESP32-Chip + Peripherie

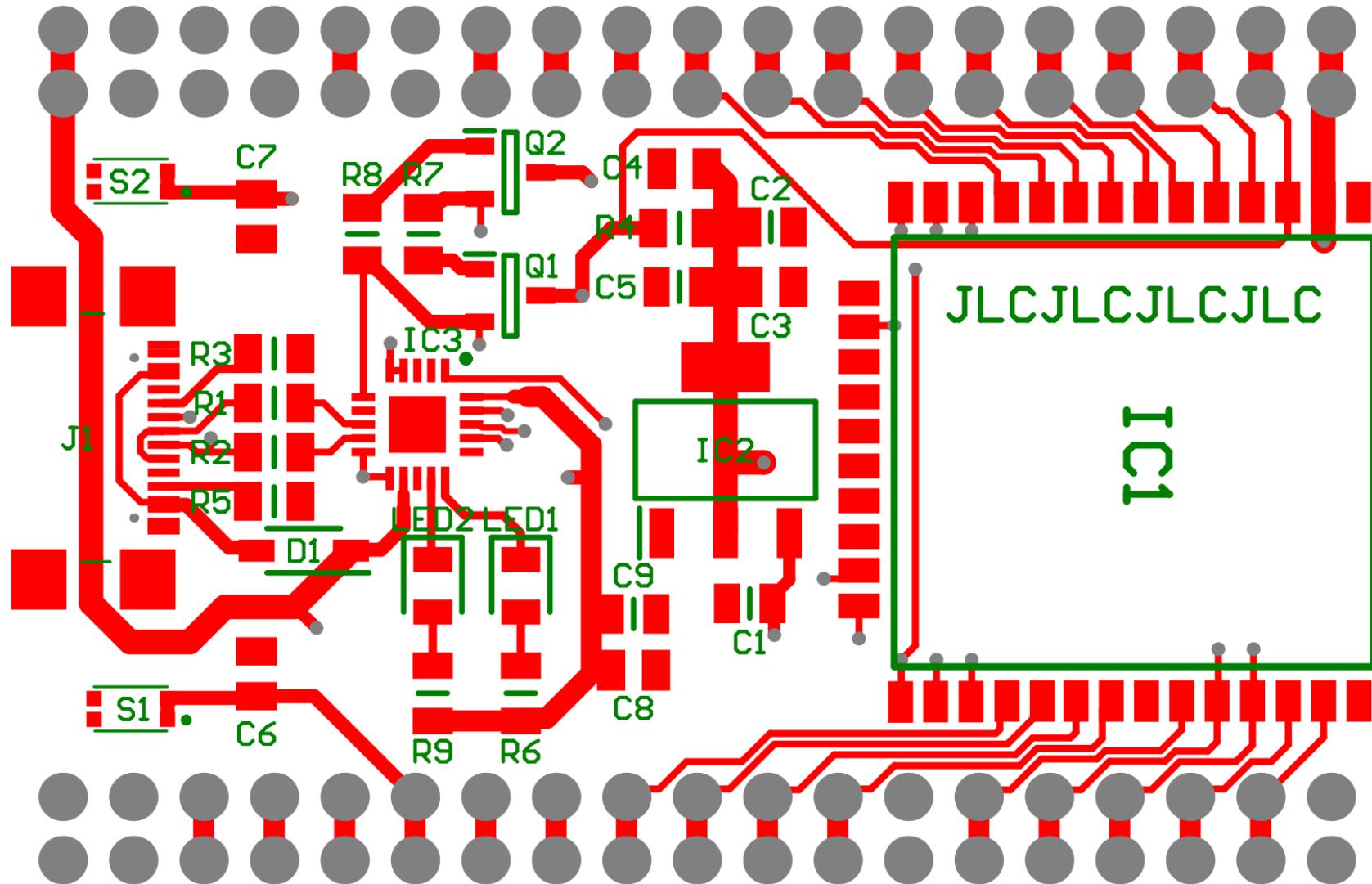
1. ESP32-Chip
2. EEPROM
3. 40 MHz Quarz
4. Anpassungsnetzwerk für Antenne
5. Antenne für WLAN/BLE

Schaltplan:

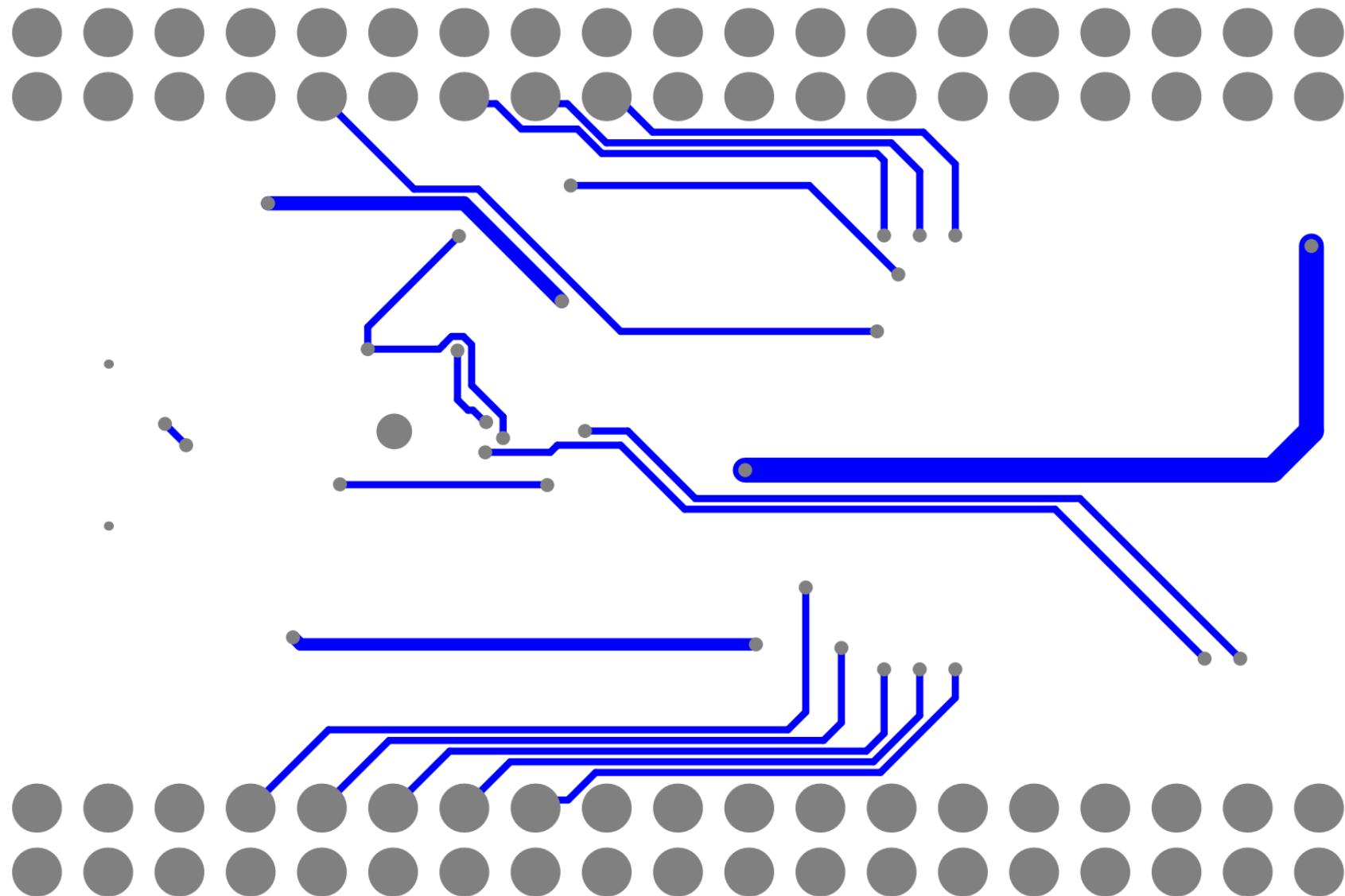


Boardplan:

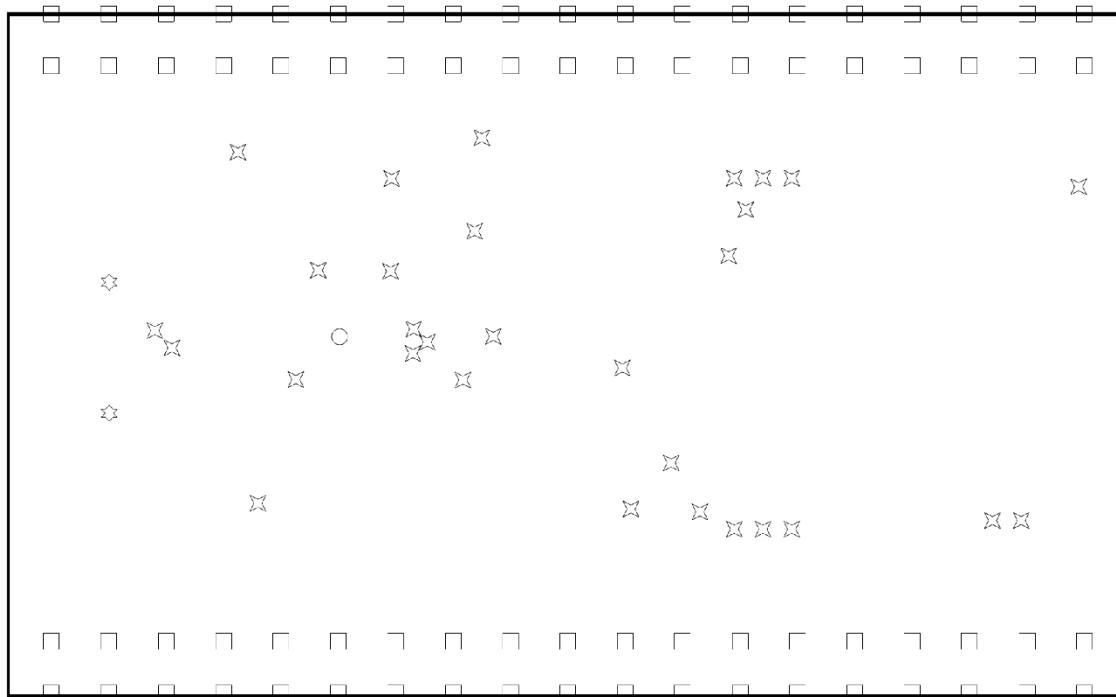
Top:



Bottom:



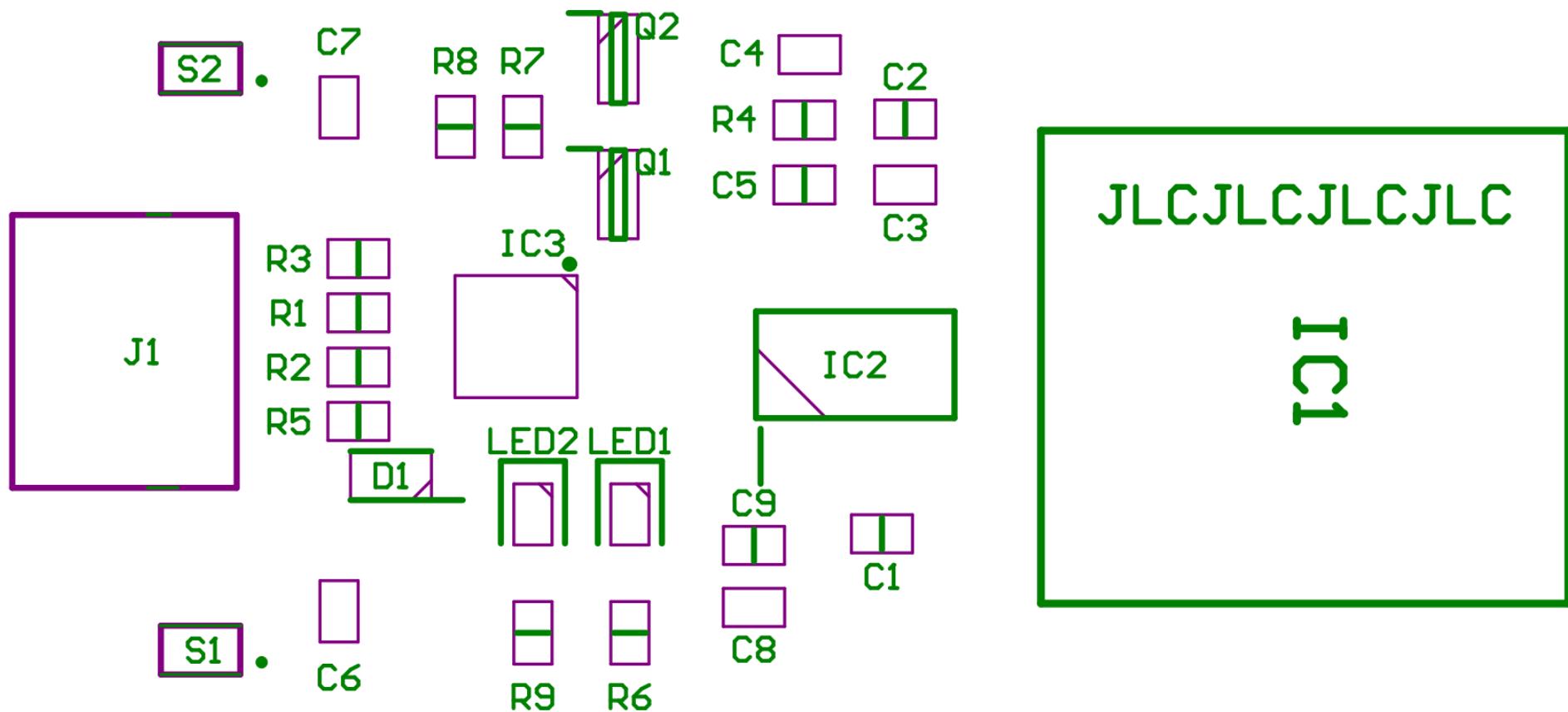
Holes:



Drill Table

Symbol	Count	Hole Size	Plated	Hole Tolerance
X	30	0.20mm	Plated	
◎	2	0.65mm	Non-Plated	
○	1	0.71mm	Plated	
□	76	1.00mm	Plated	
109 Total				

Bestückungsplan:



Bill of Materials

Description	Designator	Name	Quantity	Manufacturer 1	Manufacturer Part Number 1
Capacitor	C1, C2, C5, C9	10uF	4	Samsung	CL21A106KOQNNNG
Capacitor	C3, C4, C6, C7, C8	0.1uF	5	Kyocera AVX	08055C104KAT2A
Schottky Diode	D1	SS0540_R1_00001	1	Pan Jit	SS0540_R1_00001
	IC1	ESP32	1	Espressif Systems	ESP32-WROOM-32E-N16
Integrated Circuit	IC2	AP7361C-33ER-13	1	Diodes	AP7361C-33ER-13
Integrated Circuit	IC3	FT231XQ-T	1		
Connector	J1	USB4110-GF-A	1		
LED	LED1, LED2	150080YS75000	2	Wurth Electronics	150080YS75000
Transistor BJT NPN	Q1, Q2	PMBT3904,235	2	Nexperia	PMBT3904,235
Resistor	R1, R2	27R	2		
Resistor	R3, R5	5k1	2	Bourns	CR0805-JW-512ELF
Resistor	R4, R7, R8	10k	3	Yageo	RC0805FR-1010KL
Resistor	R6, R9	200R	2		
Switch	S1, S2	EVP-BB2A9B000	2		

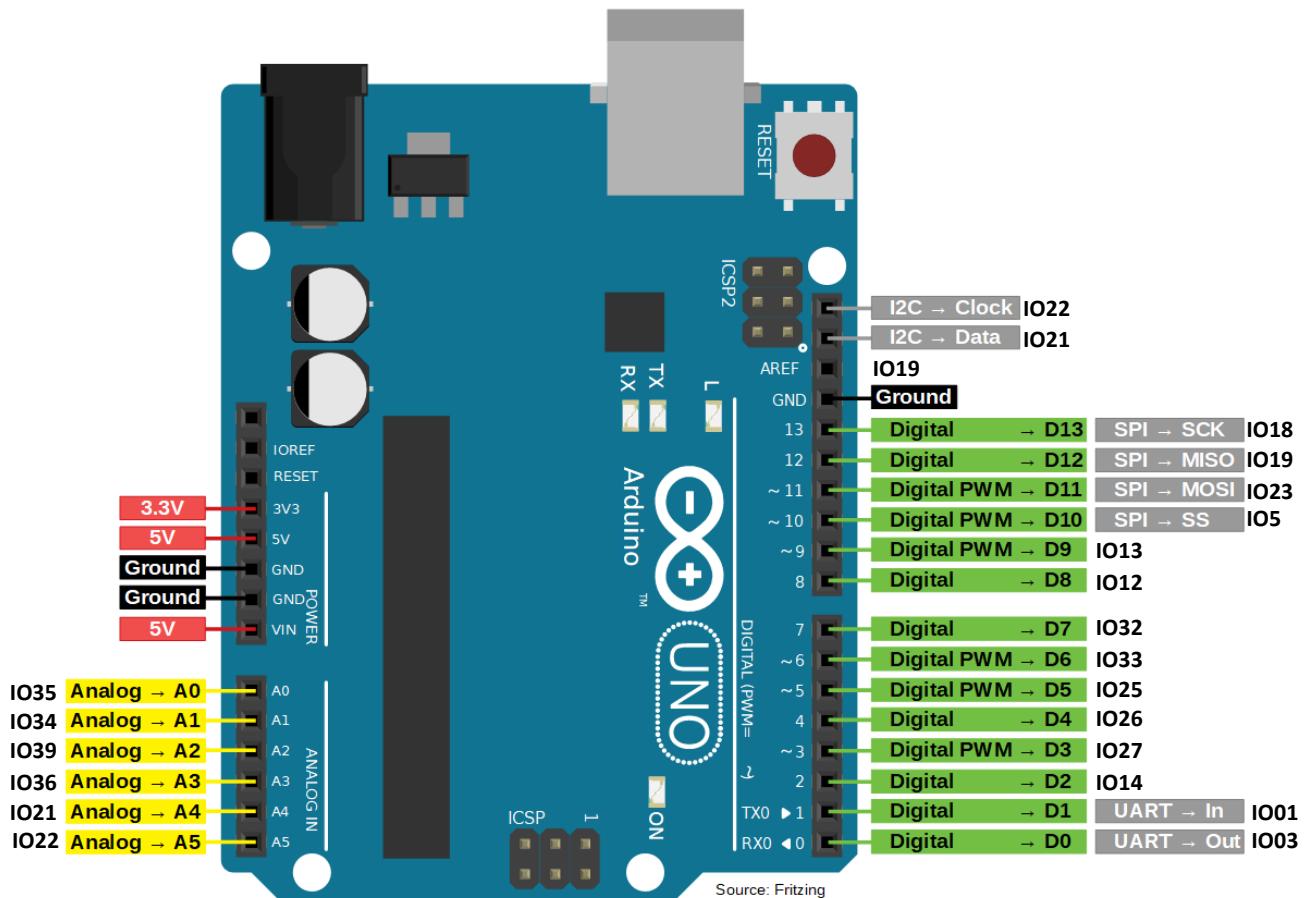
ESP32 Arduino Uno Expander:

Mit dem Arduino Uno Expander kann ein ESP32 im Formfaktor eines Arduino Uno verwendet werden. Damit können Roboter, welche für den Uno gebaut wurden, mit Wireless-Fähigkeiten ausgestattet werden.

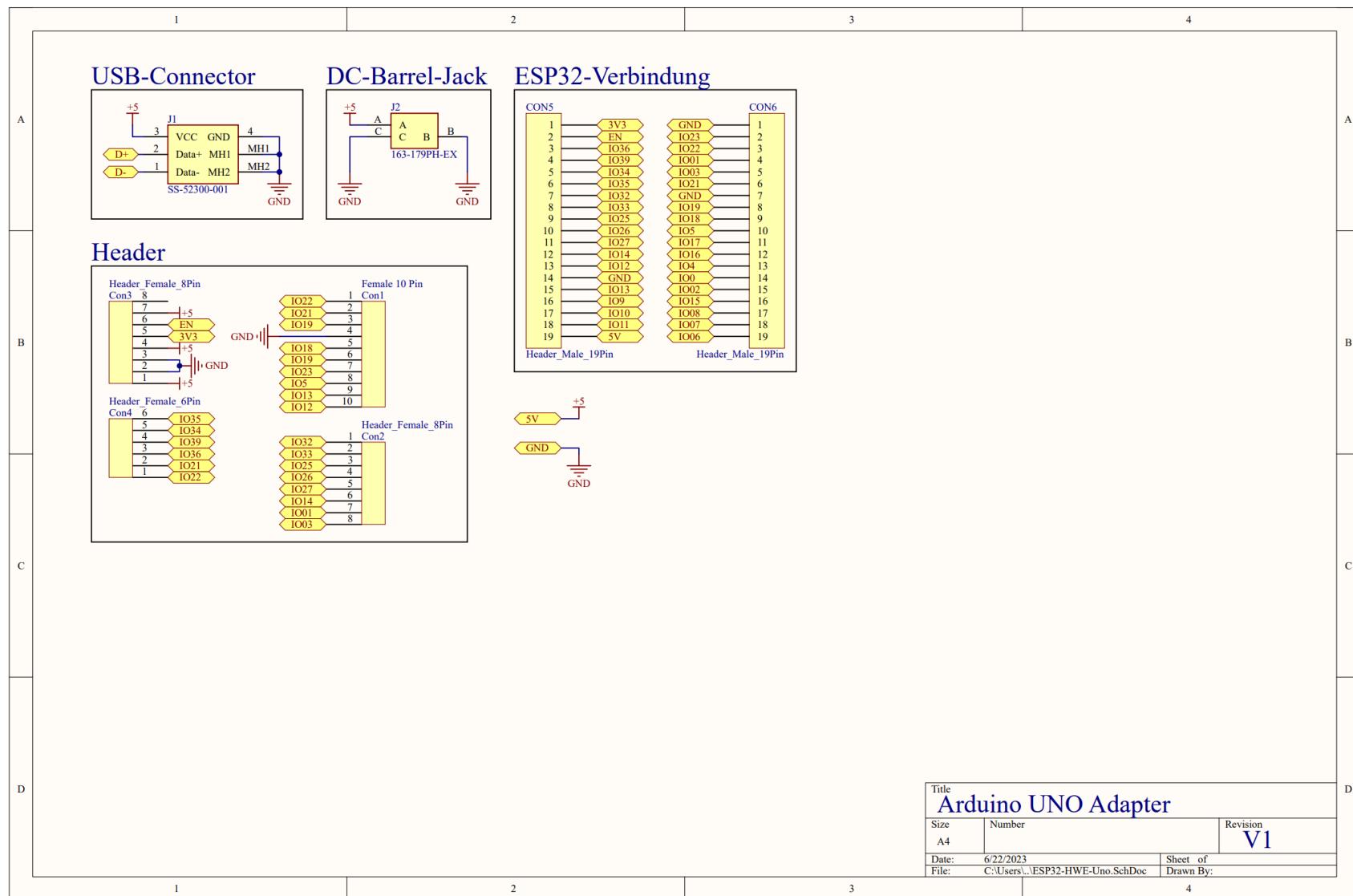
Der Expander bietet zusätzlich die Möglichkeit den ESP32 über den, beim Arduino Uno üblichen, DC-Jack und USB-B Schnittstellen mit Strom zu versorgen.

Hinweis: Die USB-B Schnittstelle dient lediglich der Stromversorgung, die Datenpins sind nicht mit dem ESP32 verbunden.

Arduino Uno Pinout:

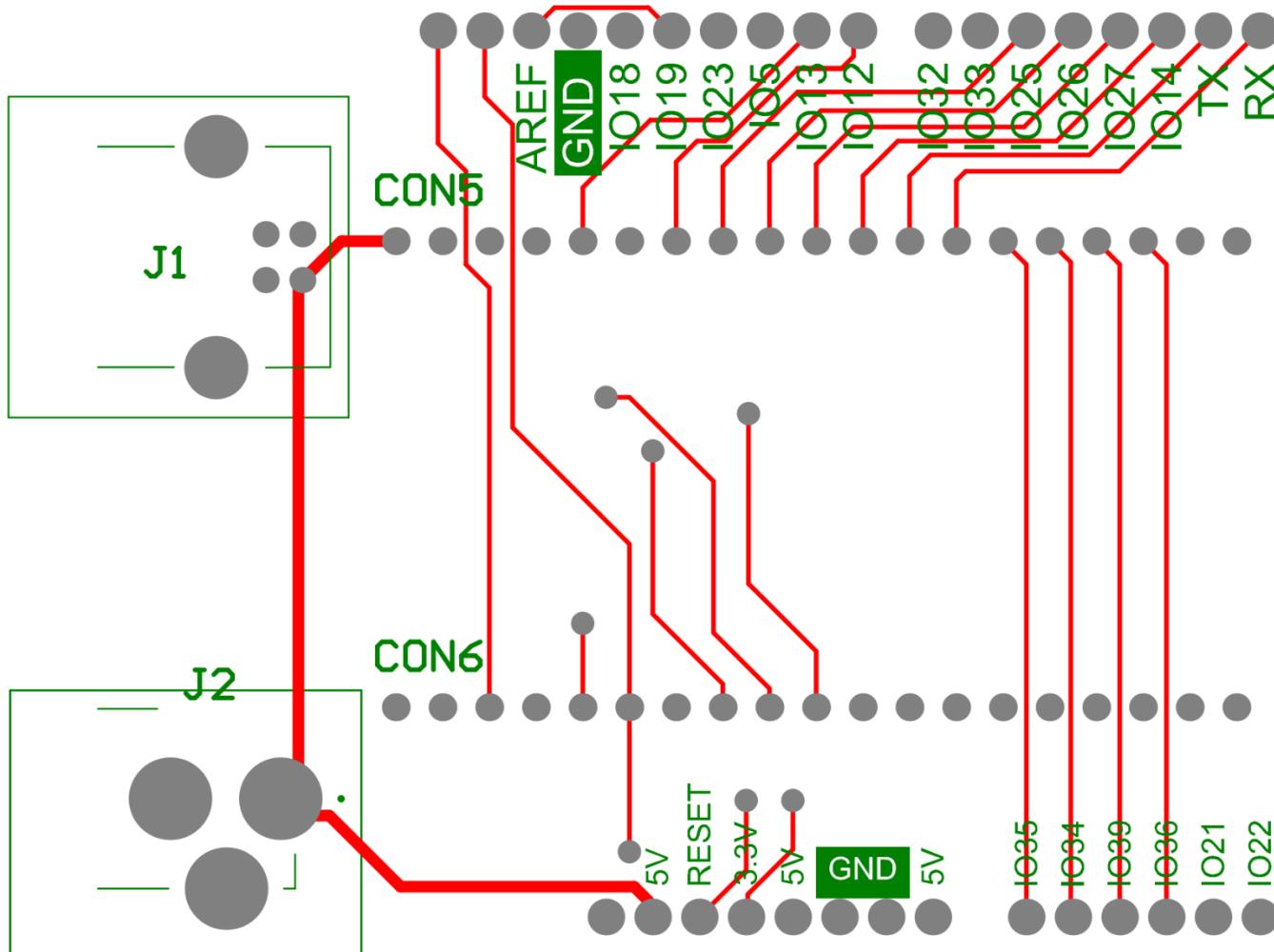


Schaltplan:

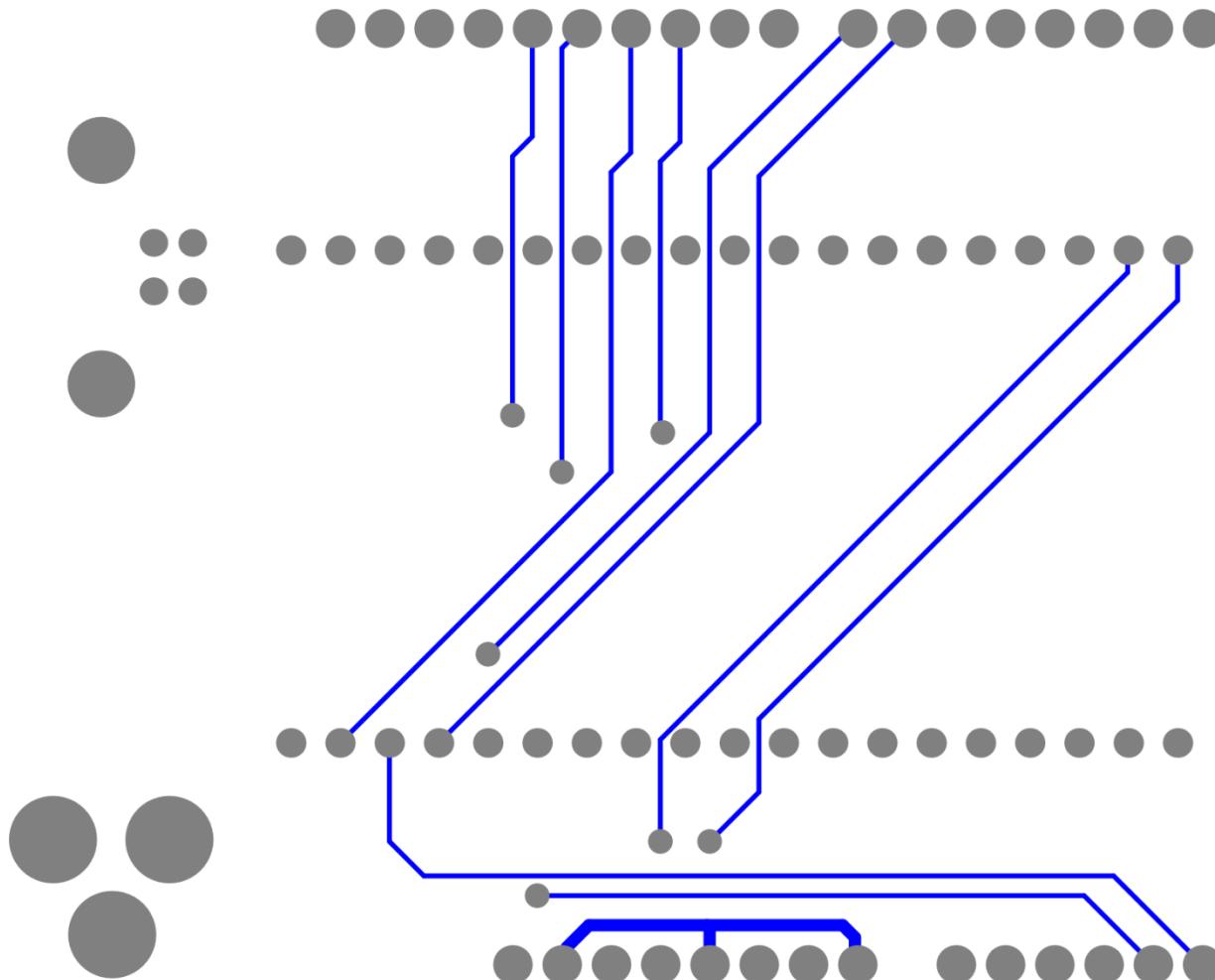


Boardplan:

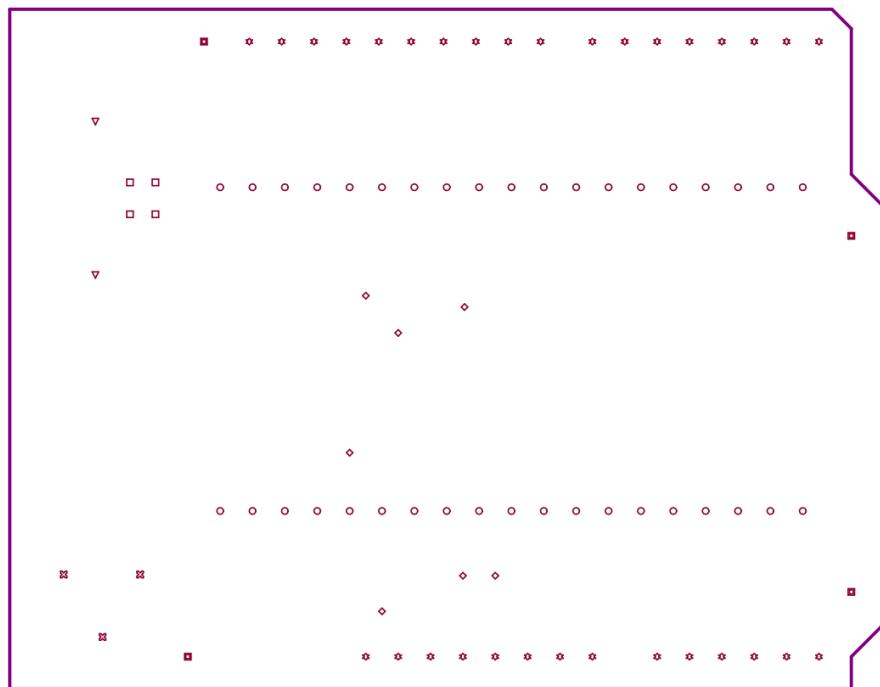
Top:



Bottom:

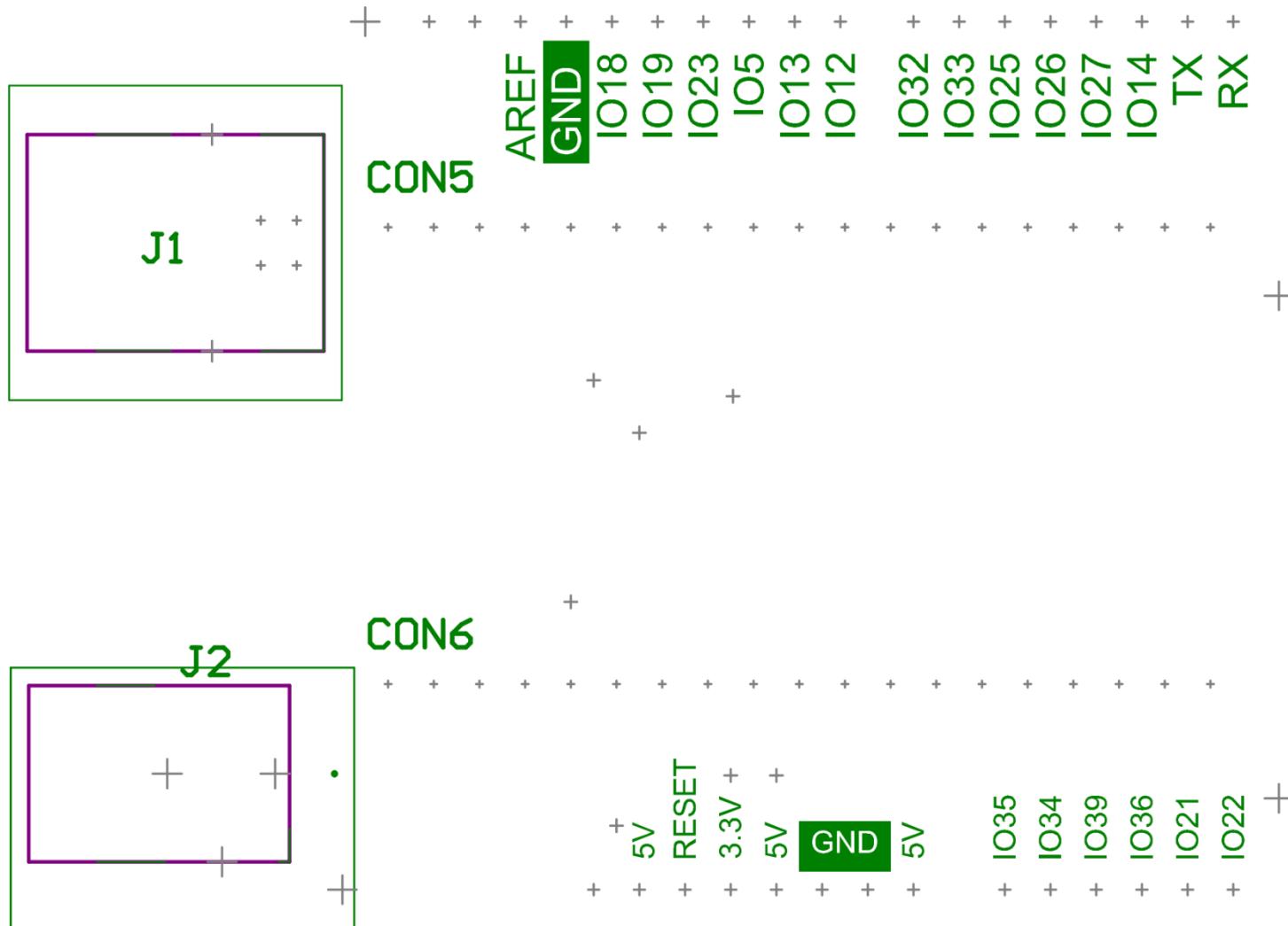


Holes:



Symbol	Count	Hole Size	Plated
▽	2	90.55mil <2.300mm>	PTH
☒	3	127.95mil <3.250mm>	PTH
□	4	36.22mil <0.920mm>	PTH
▣	4	125.00mil <3.175mm>	NPTH
◊	7	28.00mil <0.711mm>	PTH
✿	32	49.21mil <1.250mm>	PTH
○	38	30.00mil <0.762mm>	PTH
90 Total			

Bestückungsplan:



Bill of Materials:

	Description	Name	Quantity
1		Female 10 Pin	1
2		Header_Female_8Pin	2
3		Header_Female_6Pin	1
4		Header_Male_19Pin	2
5	Connector	SS-52300-001	1
6	Connector	163-179PH-EX	1

ESP32-Devboard:

Das ESP-Devboard soll eine einfache Realisierung von Software- als auch Hardware-Projekten ermöglichen. Verschiedene SPI- & I²C-Bausteine, sowie Leistungselektronik können modular über ein, am Devboard angestecktes, ESP32-Modul angesteuert werden.

Hardware:

Pin-Header:

Die Pin-Header können verwendet werden, um benutzerdefinierte Pins mit externer oder bereits auf dem Dev-Board befindlichen Hardware zu verbinden.

Pinout:

	UCC	VCC
	I016	I026
	I017	I027
	I018	I032
	I019	I033
	I021	I034
	I022	I035
	I023	I036
	I025	I039
	GND	GND

	UCC	VCC
	I000	I008
	I001	I009
	I002	I010
	I003	I011
	I004	I012
	I005	I013
	I006	I014
	I007	I015
	GND	GND

Abbildung 8 - ESP32-Devboard: Pinheader

Leistungselektronik:

Auf dem Devboard sind zwei Arten von Modulen für höhere Ausgangsleistungen untergebracht. Ein L298N-Doppel-Vollbrücken-Treiber welcher zum Beispiel für die Ansteuerung von BLDC-Motoren verwendet werden kann und 4 invertierte FET-Schalter, welche jeweils für bis zu 8A kontinuierlichem Drain-Strom ausgelegt sind. Um diese beiden Module verwenden zu können muss das Board zusätzlich mit 12V über die DC-Buchse versorgt werden.

IRF840 – invertierte FET-Schalter:

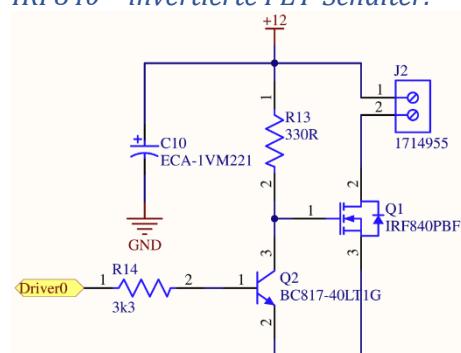
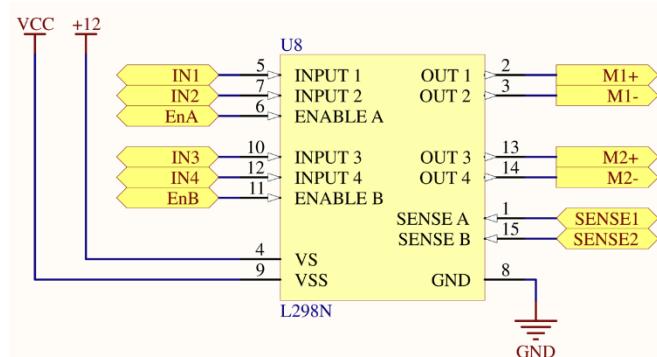


Abbildung 9 - ESP32-Devboard: invertierter FET-Schalter

Die Low-Side Treiberschaltung mit dem IRF840 (Datenblatt siehe [Referenzen](#)) und dem BC817 NPN Transistor eignet sich hauptsächlich für langsamere Schaltvorgänge. Höhere Leistungen sind möglich, doch die, durch die langsamten Schaltvorgänge resultierenden, Verluste sollten berücksichtigt werden. Das Schaltverhalten ist sehr unsymmetrisch (langsames Einschalten, sehr schnelles Ausschalten).

Driver0...3 sind mit gesetztem Jumper standardmäßig auf Pin IO16...19 verbunden, können aber jederzeit mit Jumperdrähten auf andere Pins gesetzt werden.

L298N – Doppel-Vollbrücken-Treiber:



Der L298N ist ein weiterverbreiteter IC, mit zwei integrierten Vollbrücken.

Aufgrund der großen Anzahl an Pins wurden standardmäßig keine Pins des ESP32 dem Treiber zugewiesen und lediglich für die Spannungsversorgung gesorgt.

Abbildung 10 - ESP32-Devboard: L298N Vollbrücken-Treiber

Pinout:

VCC	M1+	M1-	SN1	SN2	M2+	M2-	GND
VCC	IN1	IN2	EnA	In3	In4	EnB	GND

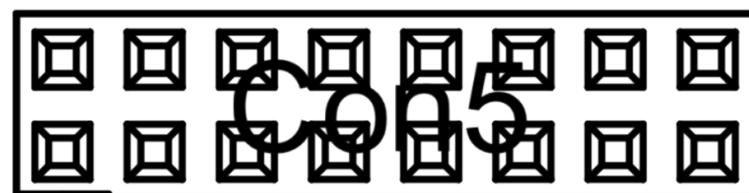
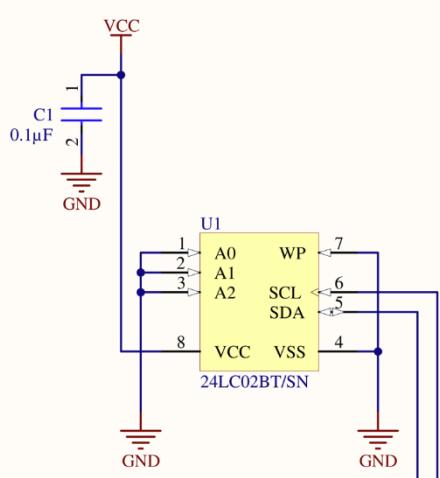


Abbildung 11 - ESP32-Devboard: L298N Pinout

I²C-Module:

Die I²C-Module können durch Umlegen des dazugehörigen Schalters entweder über die Pins IO21 (SDA) und IO22 (SCL) oder über extern am SCA/SCL-Header angeschlossenen Signale angesprochen werden. Die SCA- und SCL-Leitung sind auf dem Board, sowohl für externe als auch interne Quelle, bereits an 4.7kΩ Pull-Up Widerstände angeschlossen.

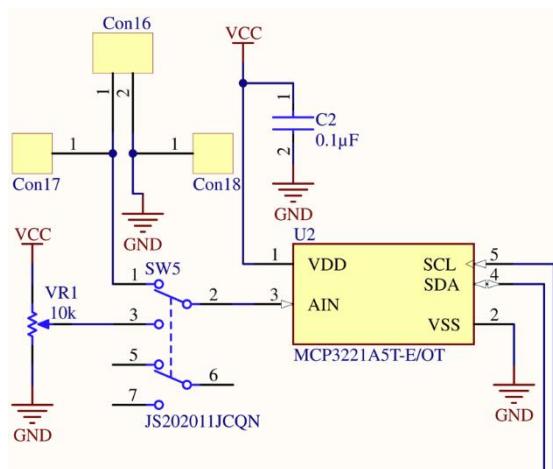
24LC02 – EEPROM



Adresse: 8-Bit 0xA...X → “don’t Care”
bzw. 7-Bit 0x5X

Da die Hauptaufgabe eines EEPROMs darin besteht empfangene Bytes zu speichern und wieder auszugeben, ist keine zusätzliche Hardware, abgesehen vom Kompensationskondensator, verbaut. (Datenblatt siehe [Referenzen](#))

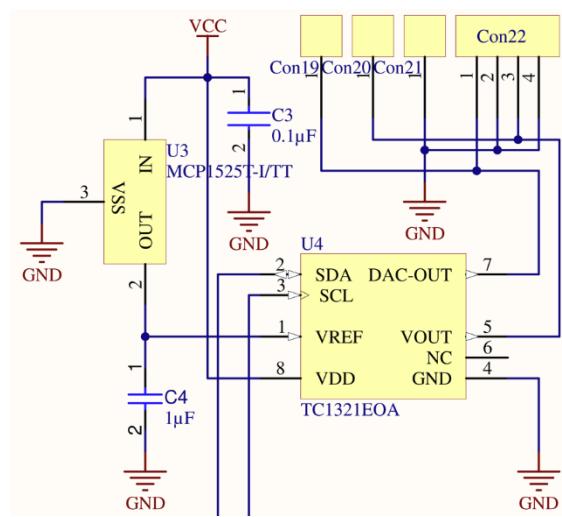
Abbildung 12 - ESP32-Devboard: I²C 24LC02-EEPROM

MCP3221 – ADC:

Adresse: 8-Bit 0x9A
bzw. 7-Bit 0x4D

Der MCP3221-ADC besitzt eine Auflösung von 12-Bit und kann durch Umschalten des Schalters entweder die über den Pinheader oder die Lötstifte extern angeschlossene Spannung oder die Spannung des Schleifers an einem Potentiometer, welches von Vcc zu GND verbunden ist, auslesen. (Datenblatt siehe [Referenzen](#))

Abbildung 13 - ESP32-Devboard: IIC MCP3221-ADC

TC1321 – DAC:

Adresse: 8-Bit 0x90
bzw. 7-Bit 0x48

Der TC1321 10-Bit DAC ist an eine externe Spannungsreferenz von 2.5V angeschlossen. Die Ausgänge DAC-OUT (unbuffered) und Vout (buffered) werden an die Lötstifte und Pinheader geführt. (Datenblatt siehe [Referenzen](#))

Aufgrund von Platzmangel ist lediglich ein Ground-Lötstift angebracht und die Silkscreen Beschriftung zeigt das Pinout der Lötstifte, welches aus diesem Grund von dem des Female-Headers abweicht. Das Pinouts des Header von Links nach rechts ist:
DAC-OUT – GND – Vout – GND

Abbildung 14 - ESP32-Devboard: IIC TC1321-DAC

MCP23008 – I/O-Expander:

Adresse: 8-Bit 0x40
bzw. 7-Bit 0x20

Die I/Os des MCP23008 I/O-Expander sind an 8 LEDs angeschlossen welche über den On/Off-Schalter entweder auf Ground oder Floating gezogen werden. Ebenfalls sind die acht I/Os des MCP23008 an den Female-Pinheader angeschlossen, um die Ausgänge mit Jumper-Drähten flexibel weiterzuverwenden. (Datenblatt siehe [Referenzen](#))

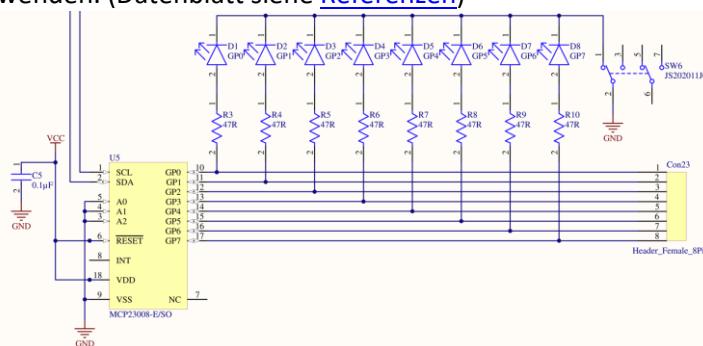
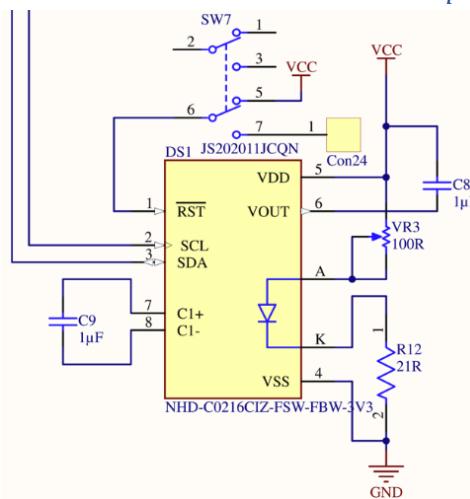


Abbildung 15 - ESP32-Devboard: IIC MCP23008-I/O-Expander

NHD-C0216CIZ-FSW-FBW-3V3 – Display:

Adresse:
bzw.

8-Bit **0x7C**
7-Bit **0x3E**

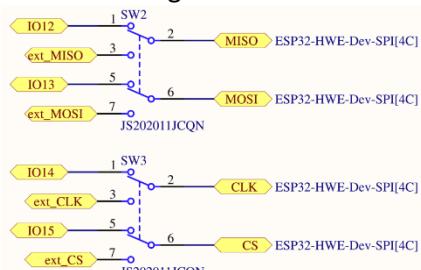
Das NHD-Display ist ein I²C 2x16 Zeichen LCD für Spannungen im Bereich von ca. 3.3V. Die Hintergrundbeleuchtung des Displays ist fix an einen 21 Ohm Vorwiderstand angeschlossen und die Helligkeit/Kontrast kann über das 100 Ohm Potentiometer variiert werden.

Der Reset-Pin kann mithilfe des Schiebschalters auf Vcc oder den Pinheader für eine externe Reset-Quelle geschalten werden. (Datenblatt siehe [Referenzen](#))

Abbildung 16 - ESP32-Devboard: I²C NHD-Display

SPI-Module:

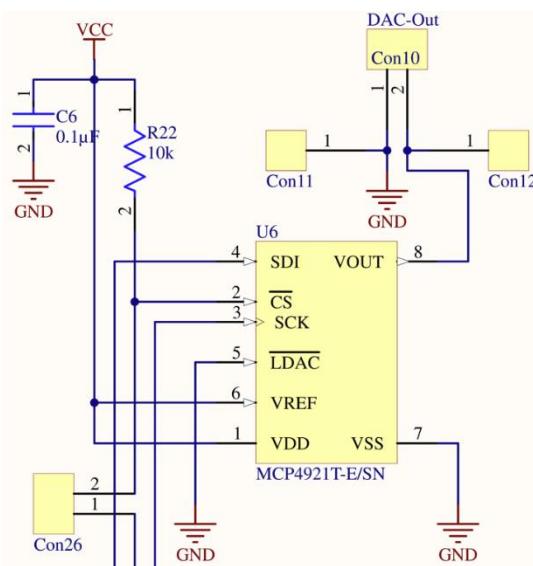
Die SPI-Module können, wie die I²C-Module, über standardmäßige intern verbundene und fixe IO-Pins angesteuert werden, oder aber auch über externe am Pinheader angeschlossene Signale. Die standardmäßigen Pins sind:



MISO → IO12
MOSI → IO13
CLK → IO14
CS → IO15

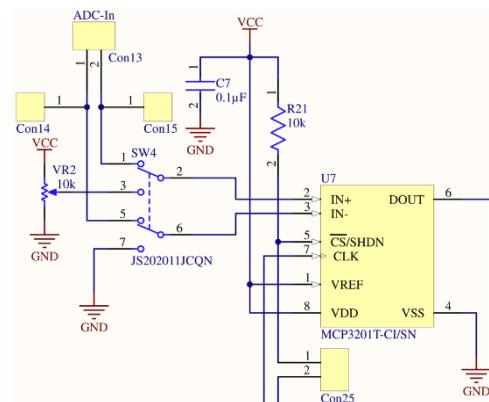
Ein Modul kann via Jumper mit dem CS-Pin verbunden werden. Werden allerdings mehrere Geräte verwendet, so empfiehlt sich zumindest eines mit Jumper-Draht extern zu verbinden.

Abbildung 17 - ESP32-Devboard: SPI-Switch

MCP4921 – DAC:

Der MCP4921 ist ein 12-Bit DAC mit einer maximalen SPI-Taktfrequenz von 20MHz. Als Referenzspannungsquelle ist der MCP4921 mit Vcc verbunden. (Datenblatt siehe [Referenzen](#))

Abbildung 18 - ESP32-Devboard: SPI MCP4921-DAC

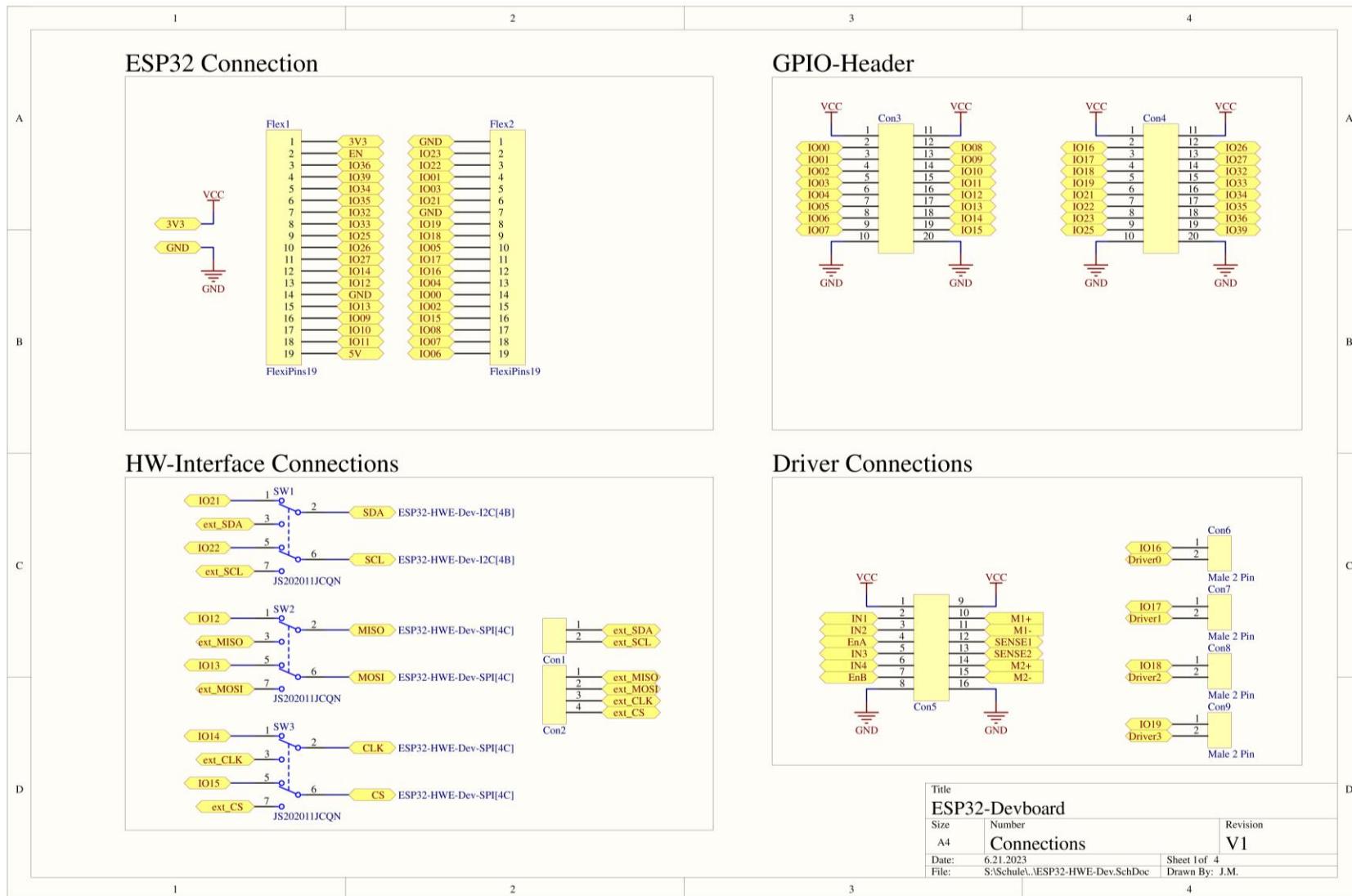
MCP3201 – ADC:

Der MCP3201 ist ein 12-Bit ADC mit einer maximalen SPI-Taktfrequenz von 0.8MHz bei 2.7V Vdd. Als Referenzspannungsquelle ist der MCP3201 mit Vcc verbunden. (Datenblatt siehe [Referenzen](#))

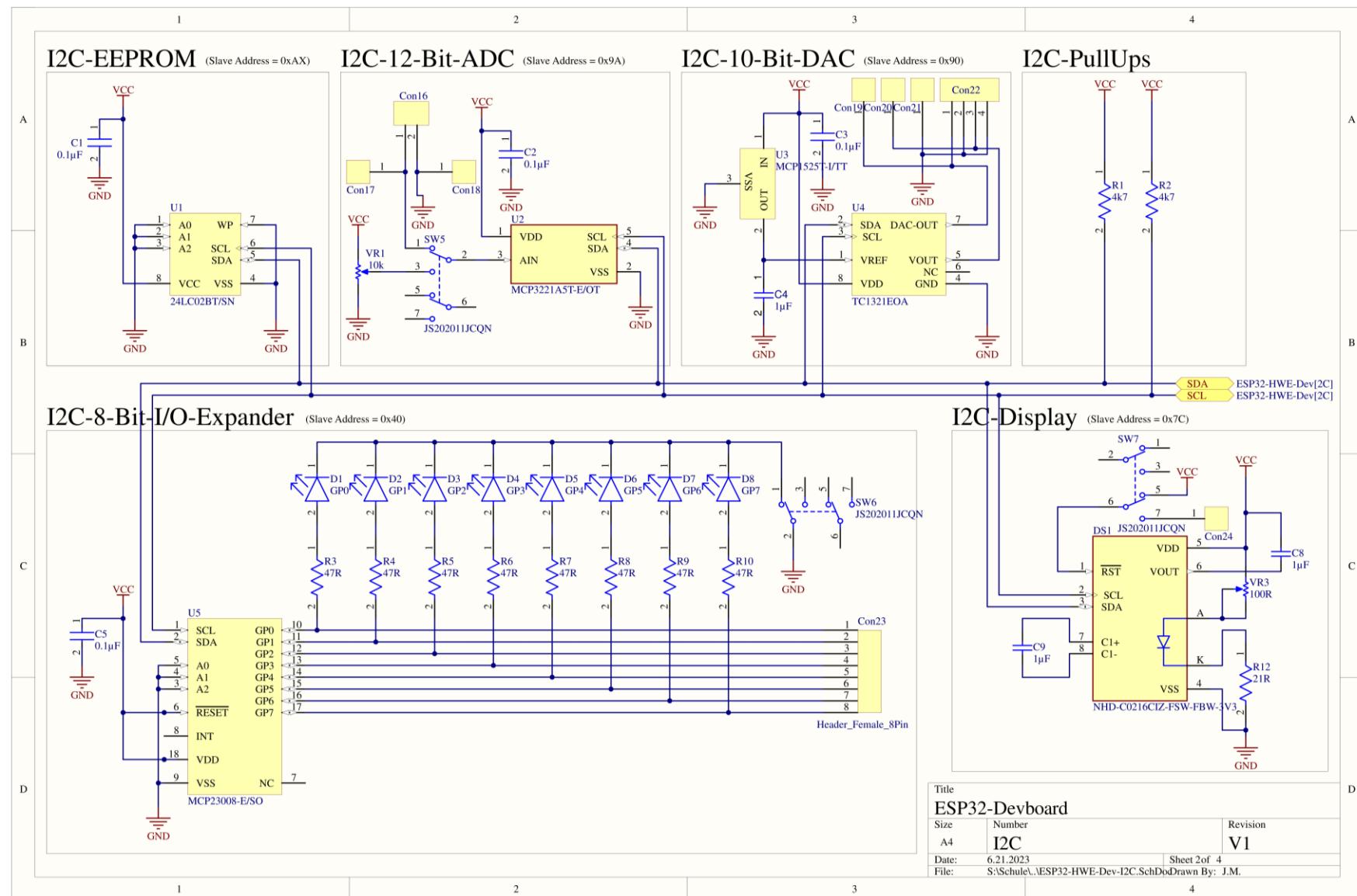
Die „Messeingänge“ des ADC können mit einem Schiebeschalter entweder auf den Schleifer eines Potentiometers, welches von Vcc zu GND verbunden ist, oder die externen Eingänge IN+ und IN- geschalten werden.

Abbildung 19 - ESP32-Devboard: SPI MCP4921-DAC

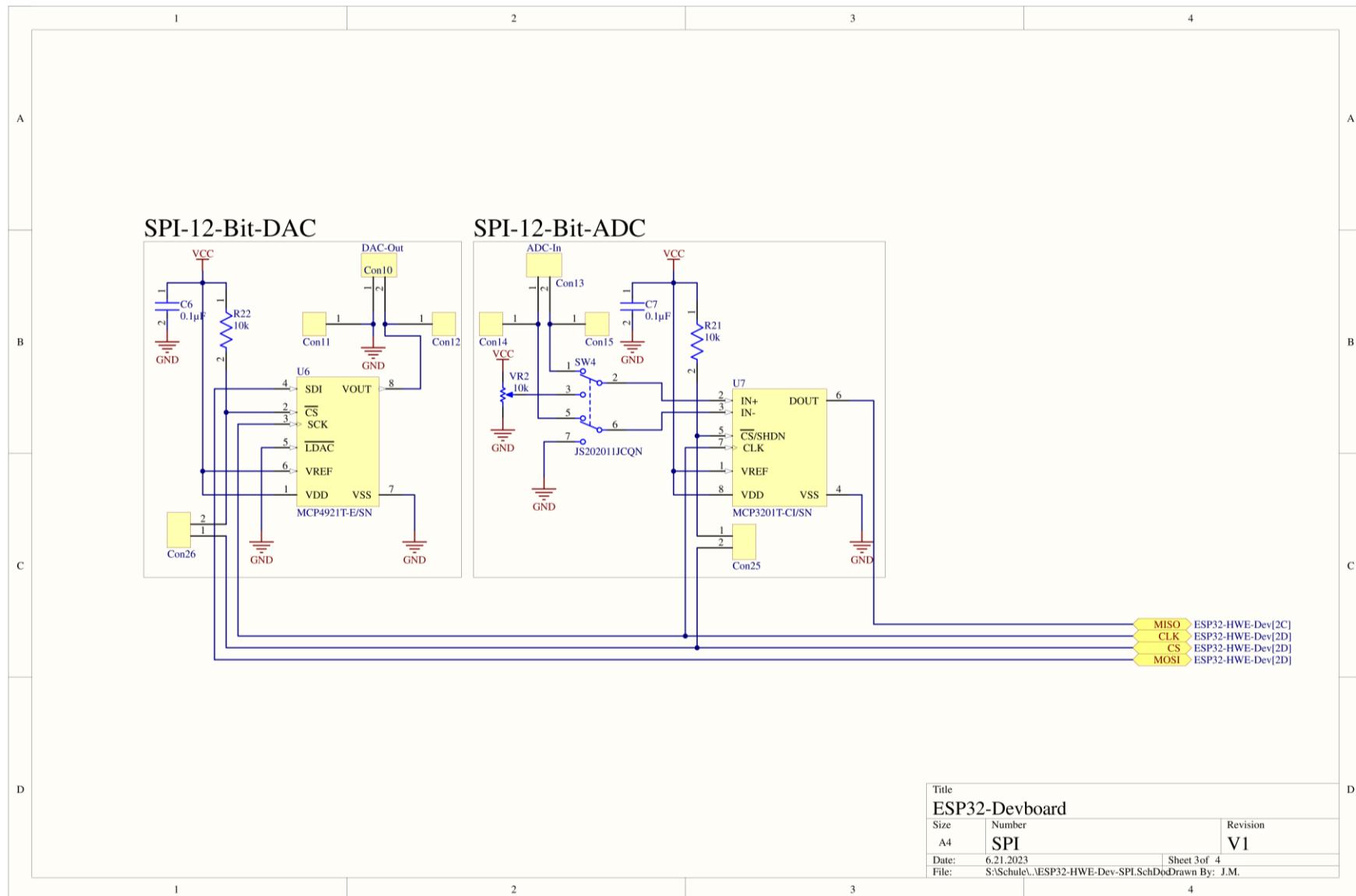
Schaltplan: Connections:



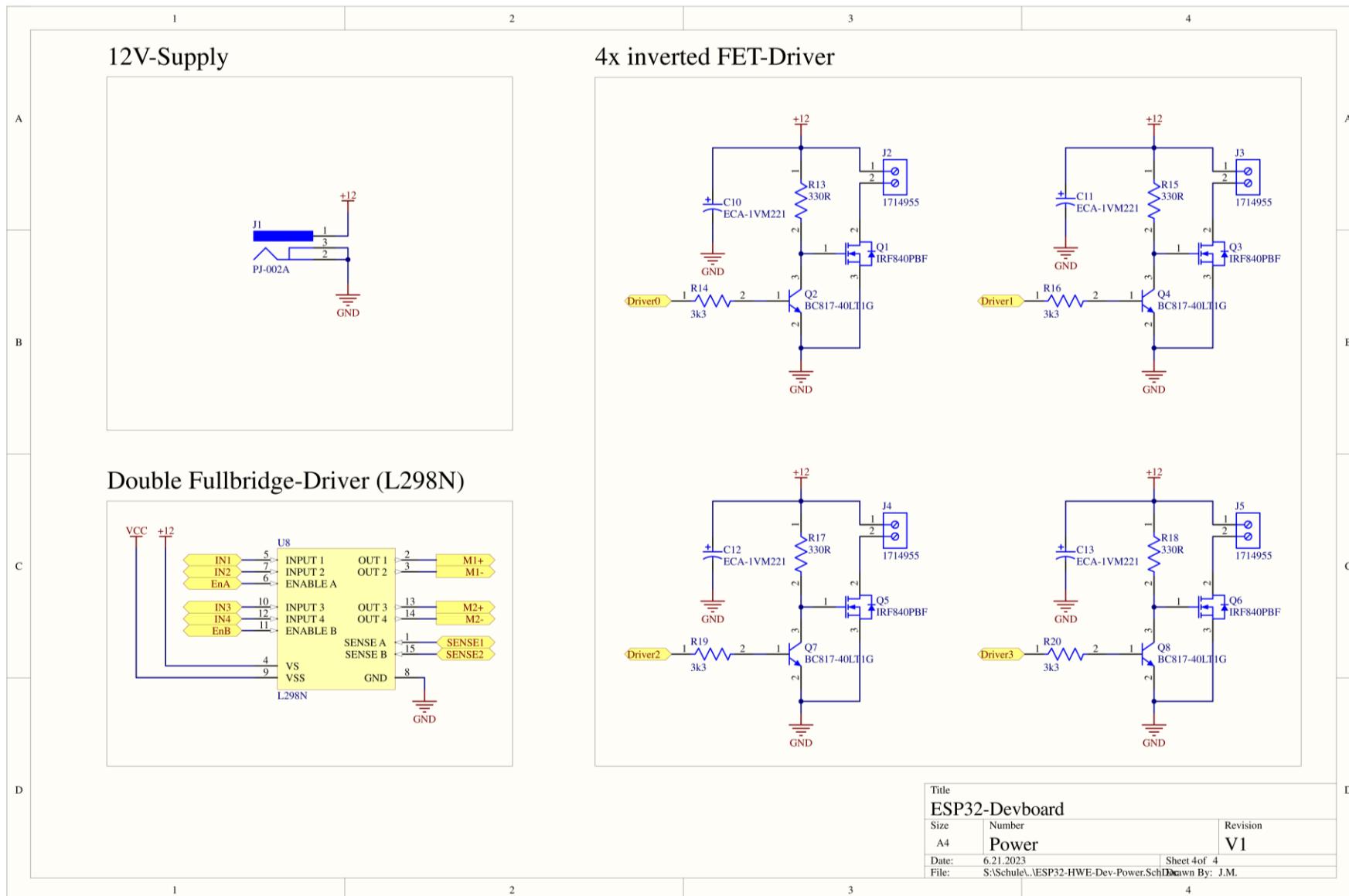
I2C:



SPI:

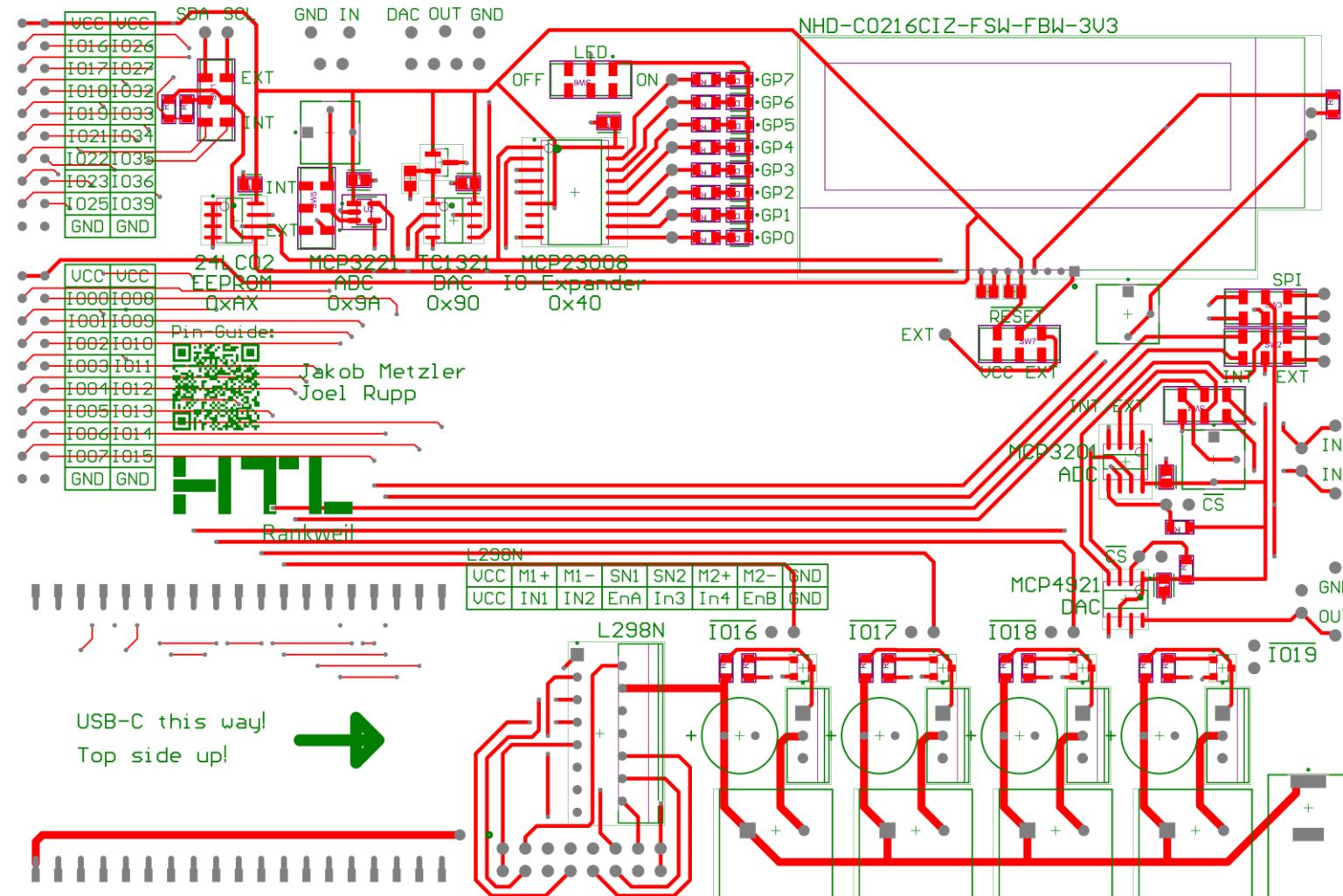


Power:

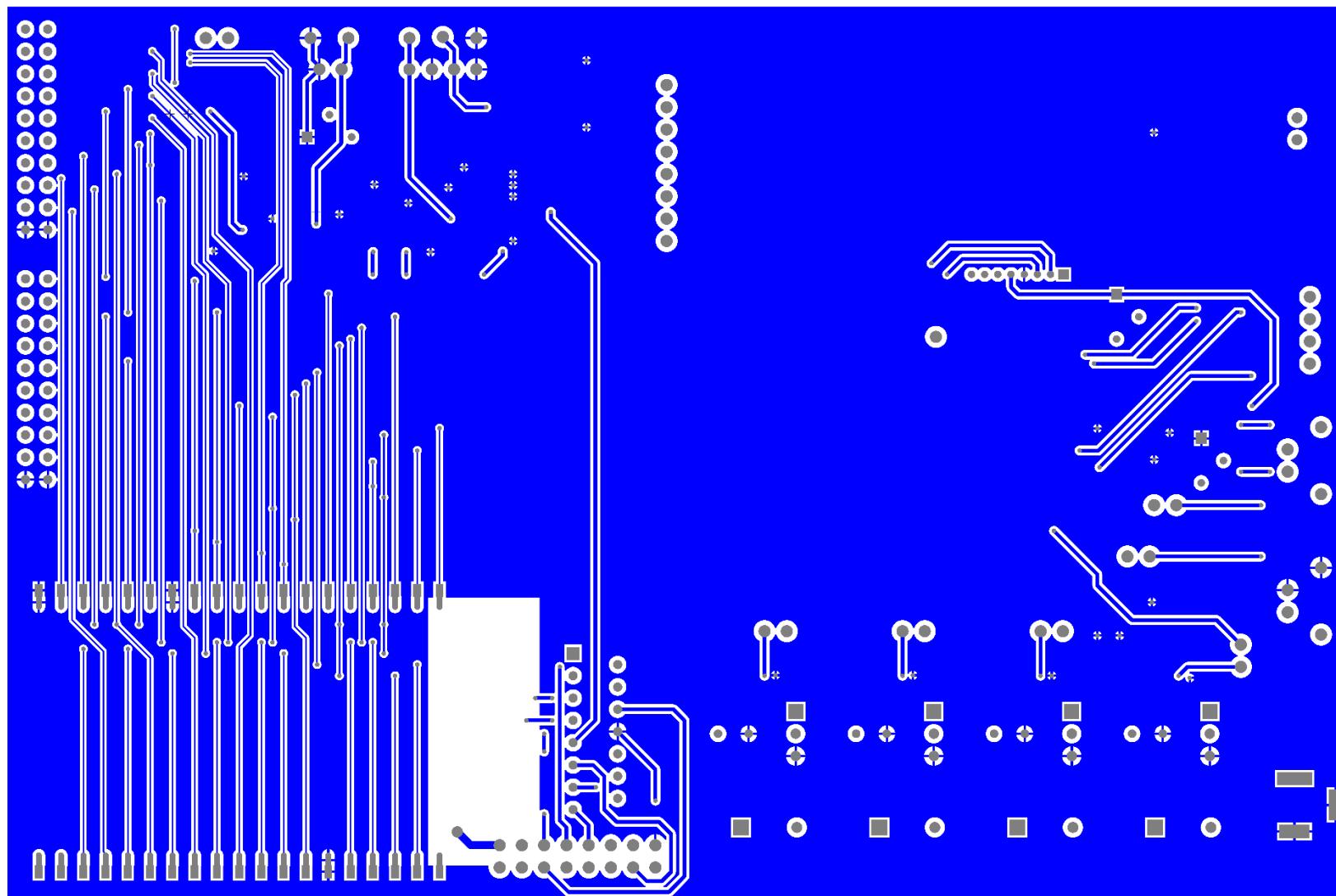


Boardplan:

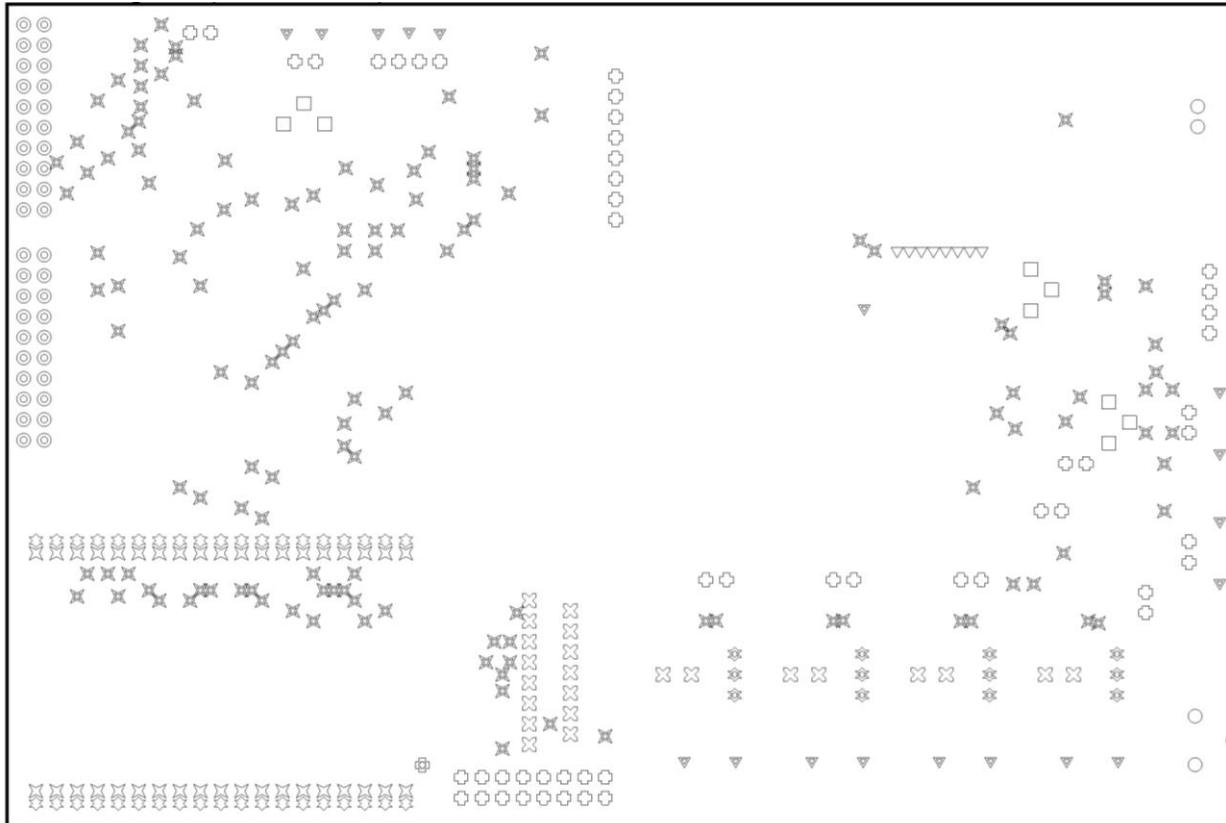
Top:



Bottom:



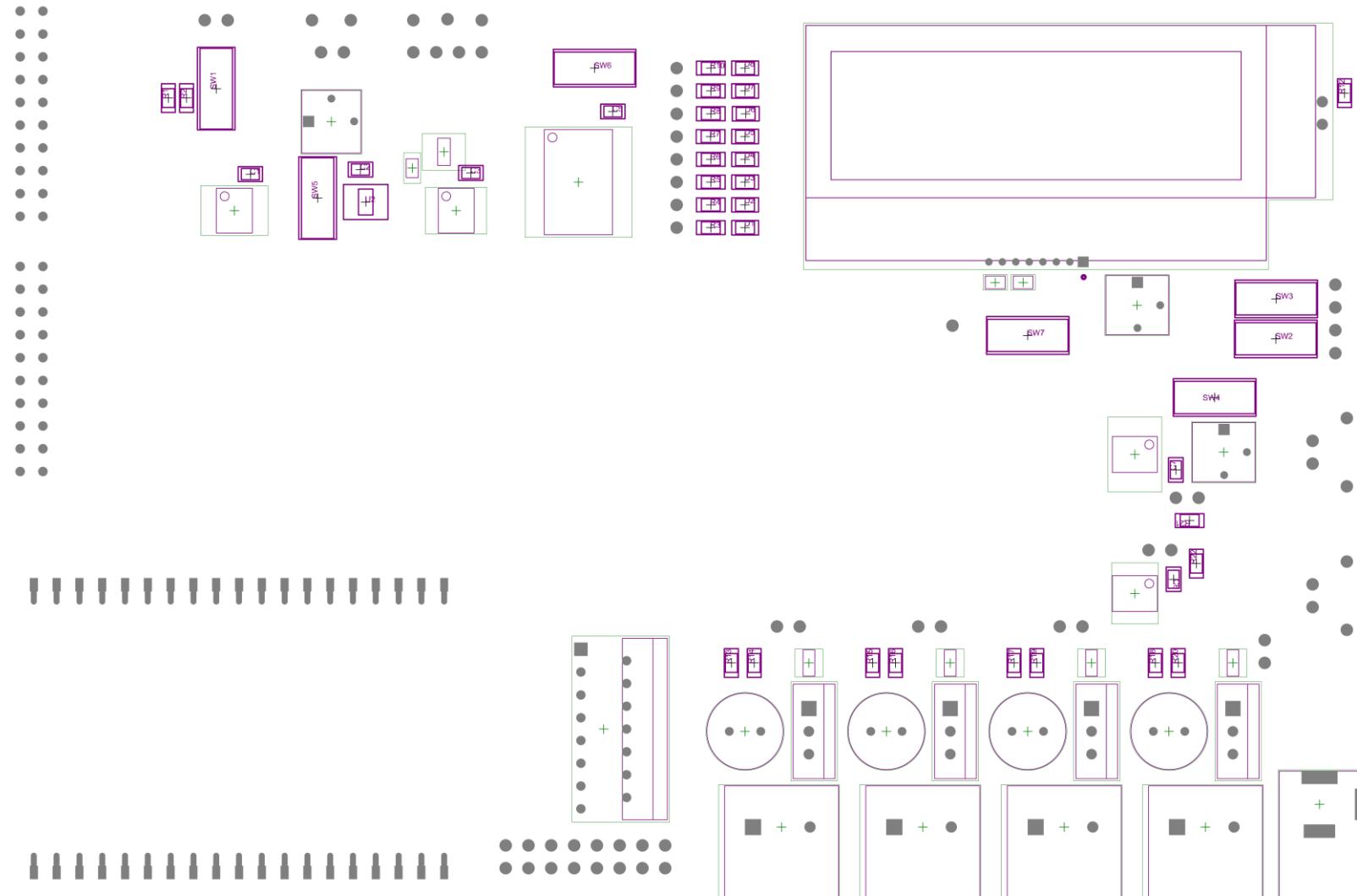
Holes:



Drill Table

Symbol	Count	Hole Size	Plated	Hole Tolerance
☒	140	0.25mm	Plated	
⊕	1	0.51mm	Plated	
⊗	38	0.60mm	Plated	
✳	38	0.65mm	Plated	
□	9	0.75mm	Plated	
◎	40	0.76mm	Plated	
▽	8	0.80mm	Plated	
✗	23	0.90mm	Plated	
○	5	1.00mm	Plated	
⊛	12	1.10mm	Plated	
⊕	52	1.25mm	Plated	
▽	18	1.30mm	Plated	
384 Total				

Bestückungsplan:



Bill of Materials:

Name	Description	Designator	Quantity	Manufacturer 1	Manufacturer Part Number 1
08055C104KAT2A	General Purpose Ceramic Capacitor, 0805, 100nF, 10%, X7R, 0.15, 50V	C1, C2, C3, C5, C6, C7	6	Kyocera AVX	08055C104KAT2A
Capacitor 1 uF +/- 10% 50 V 0805	Chip Capacitor, 1 uF, +/- 10%, 50 V, 0805 (2012 Metric)	C4	1		
C0805C105K4PACTU		C8, C9	2	Kyocera AVX	0805YC105KAT2A
ECA-1VM221	Aluminum Electrolytic Capacitor, 220 uF, +/- 20%, 35 V	C10, C11, C12, C13	4	Panasonic	ECA1VM221
Female 2 Pin		Con1, Con16	2		
Female 4 Pin		Con2, Con22	2		
Female 2x10 Pin		Con3, Con4	2		
Female 2x8 Pin		Con5	1		
Male 2 Pin		Con6, Con7, Con8, Con9, Con25, Con26	6		
DAC-Out	2 Pin Female	Con10	1		
Solder Nail RTM 1.3		Con11, Con12, Con14, Con15, Con17, Con18, Con24	10		
ADC-In	2 Pin Female	Con13	1		
I2C-DAC-DAC-Out	Solder Nail RTM 1.3	Con19	0		
I2C-DAC-Vout	Solder Nail RTM 1.3	Con20	0		
DAC-GND	Solder Nail RTM 1.3	Con21	0		
Header_Female_8Pin		Con23	1		
SML-LXT0805IW-TR	LED RED DIFFUSED SMD	D1, D2, D3, D4, D5, D6, D7, D8	8	Lumex	SML-LXT0805IW-TR
NHD-C0216CIZ-FSW-FBW-3V3	Character Display 16Char x 2Line 8-Pin	DS1	1		
FlexiPins19		Flex1, Flex2	2		
PJ-002A	Through Hole Right Angle DC Power Jack, 2.5 A	J1	1	CUI	PJ-002A

1714955	PC Terminal Block, Pitch 6.35 mm, 1 x 2 Position	J2, J3, J4, J5	4	Phoenix Contact	1714955
IRF840PBF	Power MOSFET, 500 V, -55 to 150 degC, 3-Pin FM	Q1, Q3, Q5, Q6	4	Vishay	IRF840PBF
BC817-40LT1G	General Purpose Transistor, NPN Silicon, 3-Pin SOT-23	Q2, Q4, Q7, Q8	4	ON Semiconductor	BC817-40LT1G
Resistor	4k7 1/8W 0805	R1, R2	2		
Resistor	47R 1/8W 0805	R3, R4, R5, R6, R7, R8, R9, R10	8		
Resistor	21R 1/8W 0805	R12	1		
Resistor	330R 1/8W 0805	R13, R15, R17, R18	4		
Resistor	3k3 1/8W 0805	R14, R16, R18, R20	4		
Resistor	10k 1/8W 0805	R21, R22	2		
JS202011JCQN	SWITCH SLIDE DPDT 300MA 6V	SW1, SW2, SW3, SW4, SW5, SW6, SW7	7	ITT C&K	JS202011JCQN
24LC02BT/SN	2Kbit, 400kHz, 2.5V, I2C Serial EEPROM	U1	1	Microchip	24LC02BT/SN
MCP3221A5T-E/OT	IC ADC 12BIT SAR SOT23-5	U2	1	Microchip	MCP3221A5T-E/OT
MCP1525T-I/TT	2.5V Voltage Reference, 3-Pin SOT-23	U3	1	Microchip	MCP1525T-I/TT
TC1321EOA	10-Bit Digital-to-Analog Converter with Two-Wire Interface	U4	1	Microchip	TC1321EOA
MCP23008-E/SO	8-Bit I/O Expander with Serial Interface, 18-Pin SOIC	U5	1	Microchip	MCP23008-E/SO
MCP4921T-E/SN	12-Bit DAC with SPI Interface, 8-Pin SOIC 150mil	U6	1	Microchip	MCP4921T-E/SN
MCP3201T-CI/SN	2.7V 12-Bit A/D Converter with SPI Serial Interface	U7	1	Microchip	MCP3201T-CI/SN
L298N	Dual Full-Bridge Driver, 4.8 to 46 V	U8	1	STMicroelectronics	L298N
3362P-1-103LF	Square Trimpot(R) Trimming Potentiometer, 10 Kohm	VR1, VR2, VR3	3	Bourns	3362P-1-103LF

Beispielprogramme:

Um die Funktion des Devboards und des ESP32-Moduls zu überprüfen, finden Sie anschließend einige Beispielprogramme.

Die Programme sind in Micropython geschrieben und werden live auf dem ESP32-Modul interpretiert. Programme für das ESP32-Modul können aber auch zum Beispiel über C++ beschrieben werden.

Die Programme befinden sich ebenfalls im GitHub-Repository unter [Referenzen](#).

Display:

```
from machine import Pin, I2C, UART
import time

# I2C-Adresse des Displays
slave2w = 0x3e
# Befehls- und Datercommandcode für das Display
comsend = 0x00
datasend = 0x40
# Adresse der zweiten Zeile des Displays
line2 = 0xC0

# Initialisierung der I2C-Kommunikation
i2c = I2C(0, scl=Pin(22), sda=Pin(21), freq=100000)

# Scan nach I2C-Geräten
devices = i2c.scan()

# Funktion zum Schreiben von Text auf das Display
def DisplayWrite(text):
    i2c.writeto(slave2w, bytes([datasend]) + text)

# Funktion zum Wechseln zur nächsten Zeile des Displays
def DisplayNextLine():
    i2c.writeto(slave2w, bytes([comsend, line2]))
```

```
# Funktion zur Initialisierung des Displays
def DisplayInit():
    i2c.writeto(slave2w, bytes([comsend, 0x38])) # Function set: 8-bit data
    time.sleep_ms(10)
    i2c.writeto(slave2w, bytes([comsend, 0x39])) # Function set: 8-bit data +
instruction table 1
    time.sleep_ms(10)
    i2c.writeto(slave2w, bytes([comsend, 0x14])) # Setze OSC frequency
    i2c.writeto(slave2w, bytes([comsend, 0x78])) # Setze contrast
    i2c.writeto(slave2w, bytes([comsend, 0x5D])) # Setze ICON display ON |
Booster ON | contrast
    i2c.writeto(slave2w, bytes([comsend, 0x6D])) # Setze follower circuit ON
| Set follower ratio
    i2c.writeto(slave2w, bytes([comsend, 0x0C])) # Setze display ON
    i2c.writeto(slave2w, bytes([comsend, 0x01])) # Clear display
    i2c.writeto(slave2w, bytes([comsend, 0x06])) # Entry mode | Increment
    time.sleep_ms(10)

# Überprüfen, ob die Display-Adresse in den gefundenen Geräten vorhanden ist
if 62 in devices:
    print("Display-Address found!")
    DisplayInit()
    DisplayWrite("Ready!")
else:
    print("Display-Address NOT found!")

time.sleep(3)

# Schleife zur kontinuierlichen Aktualisierung des Displays
while True:
    DisplayInit()
    DisplayWrite(" Username: ")
    DisplayNextLine()
    DisplayWrite(" XXX ")
    time.sleep(1)
    DisplayInit()
    DisplayWrite(" Password: ")
    DisplayNextLine()
    DisplayWrite(" XXX ")
    time.sleep(1)
```

I/O-Expander:

```
import machine
import time

# Define I2C pins and create I2C object
i2c = machine.I2C(0, scl=machine.Pin(22), sda=machine.Pin(21), freq=100000)

addr = 0x20

# MCP23008 Register addresses
DDR = 0x00    # I/O direction register
PORT = 0x09   # GPIO port register

def setOutput(data):
    # Set all pins as outputs
    i2c.writeto(addr, bytes([DDR, 0x00]))
    time.sleep(0.001)

    # Turn on all LEDs
    i2c.writeto(addr, bytes([PORT, data]))
    time.sleep(0.001)

# Blink the LEDs
while True:
    setOutput(0xFF)
    print("On")
    time.sleep(1)
    setOutput(0x00)
    print("Off")
    time.sleep(1)
```

I²C-ADC:

```

import machine
import time

# Define I2C pins and create I2C object
i2c = machine.I2C(0, scl=machine.Pin(22), sda=machine.Pin(21), freq=100000)

addr = 0x4D # ADC I2C address

def get_adc_value():
    # Read ADC value
    i2c.writeto(addr, bytes([0x00])) # Start conversion command
    time.sleep_ms(100) # Wait for conversion to complete
    data = i2c.readfrom(addr, 2) # Read 2 bytes of ADC value
    adc_value = (data[0] << 8) | data[1] # Combine bytes to get ADC value
    return adc_value

def get_adc_voltage():
    adc_value = get_adc_value()
    # Convert ADC value to millivolts (assuming 3.3V reference voltage)
    voltage = (adc_value * 3300) / 4096
    return voltage

# Read and print ADC value in millivolts
while True:
    adc_voltage = get_adc_voltage()
    print("ADC Value: {} mV".format(adc_voltage))
    time.sleep(1)

```

Driver:

```

from machine import Pin
import time

Driver1 = Pin(16, Pin.OUT)
Signal  = Pin(13, Pin.IN, Pin.PULL_UP)

while True:
    Driver1.value(not Signal.value())

```

ESP32-Modul WLAN + Display:

```
import network
from machine import Pin, I2C, UART
import time
import urequests

slave2w = 0x3e
comsend = 0x00
datasend = 0x40
line2 = 0xC0

i2c = I2C(0, scl=Pin(22), sda=Pin(21), freq=100000)

# WLAN-Hotspot konfigurieren
ap = network.WLAN(network.AP_IF)
ap.config(essid="ESP-Test", password="Passwort123")
ap.active(True)
print(ap.ifconfig())

count = 0

def DisplayWrite(text):
    i2c.writeto(slave2w, bytes([datasend]) + text)

def DisplayNextLine():
    i2c.writeto(slave2w, bytes([comsend, line2]))

def DisplayInit():
    i2c.writeto(slave2w, bytes([comsend, 0x38])) # Function set: 8-bit data
    time.sleep_ms(10)
    i2c.writeto(slave2w, bytes([comsend, 0x39])) # Function set: 8-bit data +
instruction table 1
    time.sleep_ms(10)
    i2c.writeto(slave2w, bytes([comsend, 0x14])) # Set OSC frequency
    i2c.writeto(slave2w, bytes([comsend, 0x78])) # Set contrast
    i2c.writeto(slave2w, bytes([comsend, 0x5D])) # Set ICON display ON |
Booster ON | contrast
    i2c.writeto(slave2w, bytes([comsend, 0x6D])) # Set follower circuit ON |
Set follower ratio
    i2c.writeto(slave2w, bytes([comsend, 0x0C])) # Set display ON
    i2c.writeto(slave2w, bytes([comsend, 0x01])) # Clear display
    i2c.writeto(slave2w, bytes([comsend, 0x06])) # Entry mode | Increment
    time.sleep_ms(10)
```

```
# Webserver-Handler für LED-Steuerung
def handle_led(request):
    global count
    count += 1
    print(request)
    DisplayInit()
    DisplayWrite("Zaehlerstand:")
    DisplayNextLine()
    DisplayWrite(str(count))
    return "Erfolg!"

# Webserver starten
def start_webserver():
    import usocket as socket

    server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    server.bind(("0.0.0.0", 80))
    server.listen(1)

    while True:
        client, addr = server.accept()
        request = client.recv(1024)
        request = str(request)
        if "GET /led" in request:
            response = handle_led(request)
        else:
            response = "Unbekannte Anfrage"
        response_headers = "HTTP/1.0 200 OK\r\nContent-Type:
text/html\r\n\r\n"
        client.send(response_headers + response)
        client.close()

# Hauptprogramm
def main():
    DisplayInit()
    DisplayWrite("Ready!")
    start_webserver()

# Programm ausführen
main()
```

Referenzen:

- GitHub-Repository
(<https://github.com/Skh4rf/HWE-ESP32>)
- ESP32-Modul Gerber-Files
(<https://github.com/Skh4rf/HWE-ESP32/tree/main/production/ESP32-HWE>)
- ESP32-Uno-Expander Gerber-Files
(<https://github.com/Skh4rf/HWE-ESP32/tree/main/production/ESP32-HWE-Uno>)
- ESP32-Devboard Gerber-Files
(<https://github.com/Skh4rf/HWE-ESP32/tree/main/production/ESP32-HWE-Dev>)
- Beispielprojekte
(<https://github.com/Skh4rf/HWE-ESP32/tree/main/examples>)
- ESP32-WROOM-E Datenblatt
(<https://github.com/Skh4rf/HWE-ESP32/blob/main/doc/datasheets/ESP32-WROOM-E.pdf>)
- 24LC02 Datenblatt
(<https://github.com/Skh4rf/HWE-ESP32/blob/main/doc/datasheets/24LC02.pdf>)
- IRF840 Datenblatt
(<https://github.com/Skh4rf/HWE-ESP32/blob/main/doc/datasheets/IRF840.pdf>)
- L298N Datenblatt
(<https://github.com/Skh4rf/HWE-ESP32/blob/main/doc/datasheets/L298N.pdf>)
- MCP23008 Datenblatt
(<https://github.com/Skh4rf/HWE-ESP32/blob/main/doc/datasheets/MCP23008.pdf>)
- MCP3201 Datenblatt
(<https://github.com/Skh4rf/HWE-ESP32/blob/main/doc/datasheets/MCP3201.pdf>)
- MCP3221 Datenblatt
(<https://github.com/Skh4rf/HWE-ESP32/blob/main/doc/datasheets/MCP3221.pdf>)
- MCP4921 Datenblatt
(<https://github.com/Skh4rf/HWE-ESP32/blob/main/doc/datasheets/MCP4921.pdf>)
- NHD-C0216CiZ-FSW-FBW-3V3 Datenblatt
(<https://github.com/Skh4rf/HWE-ESP32/blob/main/doc/datasheets/NHD-C0216CiZ-FSW-FBW-3V3.pdf>)
- TC1321 Datenblatt
(<https://github.com/Skh4rf/HWE-ESP32/blob/main/doc/datasheets/TC1321.pdf>)

Projektplan:

ESP32-Devboard

22.06.2023

<https://github.com/Skh4rf/HWE-ESP32>

Projekt Manager
Projektbeginn/endé

Joel Rupp, Jakob Metzler
28.02.2023 - 23.06.2023

Fortschritt
Vorgänge
Ressourcen

100%
41
2

ESP32-Devboard

22.06.2023

Vorgänge

Kosten	Vorgang	Anfang	Ende
0	ESP32-Modul + Devboard + Bot-Exapnder	28.02.23	22.06.23
0	späteste Projektabgabe <i>Abgabe Hardware+Software</i>	23.06.23	23.06.23
0	ESP32-Modul definieren <i>Das ESP32-Modul muss vollständig in seiner Funktion, mechanik und Pinout beschrieben sein, um mit weiteren Arbeiten fortzufahren.</i>	28.02.23	06.03.23
0	ESP32-Modul mechanisch definieren <i>Maße des PCBs fixieren (Höhe, Breite, Länge)</i>	28.02.23	06.03.23
0	ESP32-Modul Pinout definieren <i>Pinout des PCBs definieren</i>	28.02.23	06.03.23
0	ESP32-Modul elektrisch definieren <i>Funktionen des Moduls bestimmen und elektronische Komponenten für das Modul festlegen.</i>	28.02.23	06.03.23
0	ESP32-Modul definiert	07.03.23	07.03.23
0	Bot-Expander Entwicklung	07.03.23	13.03.23
0	Bot-Expander Pinout festlegen <i>Fixieren des Pinouts an den Roboter. Ansprechperson: Franz Lauritsch</i>	07.03.23	07.03.23
0	Bot-Expander PCB entwerfen <i>Expander-PCB entwerfen</i>	08.03.23	13.03.23
0	ESP32-Modul Entwicklung	07.03.23	20.03.23
0	ESP32-Modul PCB entwerfen	07.03.23	20.03.23
0	ESP32-Modul BOM erstellen	20.03.23	20.03.23
0	Entwicklungsphase Modul + Expander abschließen <i>Entwicklung des ESP32-Modul und des Bot-Expanders abgeschlossen</i>	21.03.23	21.03.23
0	Bestellung (Modul + Expander)	21.03.23	03.04.23
0	ESP32-Modul Bauteile bestellen	21.03.23	03.04.23
0	ESP32-Modul PCB bestellen	21.03.23	03.04.23
0	Bot-Expander PCB bestellen	21.03.23	03.04.23
0	Bot-Expander hestellen	14.04.23	14.04.23
0	Bot-Expander bestücken	14.04.23	14.04.23
0	Bot-Expander funktionsfähig	14.04.23	14.04.23

ESP32-Devboard

22.06.2023

3

Vorgänge

Kosten	Vorgang	Anfang	Ende
0	Bot-Expander funktionsfähig	17.04.23	17.04.23
0	ESP32-Modul herstellen	04.04.23	11.04.23
0	ESP32-Modul bestücken	04.04.23	10.04.23
0	ESP32-Modul überprüfen	11.04.23	11.04.23
0	ESP32-Modul funktionsfähig	12.04.23	12.04.23
0	ESP32-Devboard Entwicklung	07.03.23	03.04.23
0	ESP32-Devboard Funktionen definieren	07.03.23	10.03.23
0	ESP32-Devboard Schaltung und PCB entwickeln	13.03.23	03.04.23
0	Bestellung (Devboard)	04.04.23	24.04.23
0	ESP32-Devboard Bauteile bestellen	04.04.23	17.04.23
0	ESP32-Devboard PCB bestellen	04.04.23	24.04.23
0	ESP32-Devboard herstellen	25.04.23	10.05.23
0	ESP32-Devboard bestücken	25.04.23	03.05.23
0	ESP32-Devboard überprüfen	04.05.23	10.05.23
0	ESP32-Devboard funktionsfähig	11.05.23	11.05.23
0	Dokumentation	11.05.23	21.06.23
0	Bot-Expander technisch Dokumentieren	11.05.23	17.05.23
0	ESP32-Devboard technisch Dokumentieren	11.05.23	21.06.23
0	ESP32-Modul technisch Dokumentieren	11.05.23	21.06.23
0	Projektabgabe	22.06.23	22.06.23

ESP32-Devboard

22.06.2023

Ressourcen

Ressource

Joel Rupp
Jakob Metzler

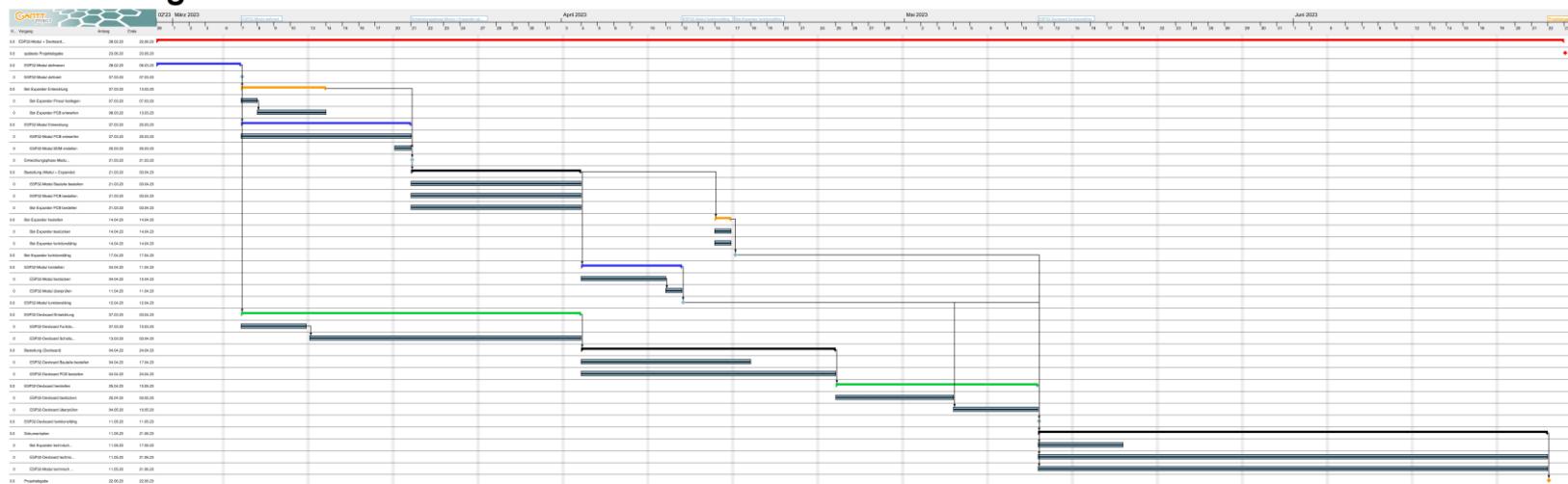
4

ESP32-Devboard

Gantt-Diagramm

22.06.2023

5



ESP32-Devboard

22.06.2023

6

Ressourcendiagramm

