Fachhochschule
Dortmund
University of Applied Sciences and Arts

FB Informations- und Elektrotechnik

Bachelor Thesis
zur Erlangung des Akademischen Grades
Bachelor of Engineering

Konfiguration eines STM32-Mikrocontrollers als einstellbare Referenzspannungsquelle mit SCPI-Schnittstelle

Omar Ben Hamouda
Matrikel Nr. 7099635

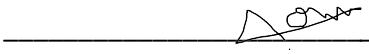
Erstprüfer: **Prof. Dr.-Ing Michael Karagounis**
Zweitprüfer:**M.A. Alexander Walsemann**

Abgabedatum: 13.08.2022

Erklärung

Hiermit versichere ich an Eides statt, dass die von mir vorgelegte Arbeit selbstständig und ohne unzulässige fremde Hilfe erstellt worden ist. Alle verwendeten Quellen sind in der Arbeit so aufgeführt, dass Art und Umfang der Verwendung nachvollziehbar sind.

Dortmund, 13.08.22022



Unterschrift

Kurzfassung

In dieser Bachelor-Arbeit wird ein Mikrocontroller mit integriertem DAC Baustein so konfiguriert, dass er als Referenzspannungsquelle dienen kann, welche mit SCPI Befehlen gesteuert werden kann. In diesem Projekt wird das STM32L476 Nucleo-Board mit zwei unabhängigen DAC Kanälen verwendet. Die beiden Kanäle des DAC werden so konfiguriert, dass sie einfache Gleichspannungen entsprechend der Benutzereingabe erzeugen.

Eine Qt-Anwendung wurde entwickelt, um mit dem Board zu kommunizieren und die Programmierung des Mikrcontrollers zu testen. Die Qt-Anwendung sendet einen Befehl an den Mikrocontroller. Der Mikrocontroller empfängt den Befehl, und auf der Grundlage dieses Befehls wird die entsprechende Anweisung ausgeführt.

Summary

In this bachelor thesis, a microcontroller with an integrated DAC device is configured to serve as a reference voltage source that can be controlled with SCPI commands.. In this project the STM32L476 Nucleo board with two indpendent DAC channels is used. The two channels of the DAC are configured to produce simple DC voltages according to the user input.

A Qt application has been developed to communicate with the board and test the microcontroller programming. The Qt application sends a command to the microcontroller. The microcontroller receives the command and based on the command the corresponding instruction is executed.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung

Im Rahmen dieses Projekts wird ein Mikrocontroller mit integriertem DAC Baustein so konfiguriert, dass er als Referenzspannungsquelle dienen kann, welche mit SCPI Befehlen gesteuert werden kann. Dabei steht SCPI für "Standard Commands for Programmable Instruments" und entspricht einem standardisiertem Befehlsformat, der zur Steuerung und Programmierung von Messgeräten verwendet wird. In diesem Projekt wird das STM32L476 Nucleo Board für die DAC-Konfiguration verwendet. Die beiden DAC-Kanäle sind so konfiguriert, dass sie einfache DC-Spannungen entsprechend der Benutzereingabe liefern. Als Entwicklungsumgebung werden STM32CubeIDE und STM32CubeMx werden für die Entwicklung verwendet.

Im ersten Schritt wird ein neues Projekt in der STM32CubeIDE erstellt. STM32CubeMx stellt einen Board-Konfigurator zur Verfügung, der das STM32L476 Nucleo Board unterstützt. Aus diesem Grund wird das neue Projekt mit STM32CubeIDE erstellt.

Im Anschluss kommt STM32CubeMX für die Projektkonfigurationen und die Codegenerierung zur Anwendung. Nach der Generierung des Codes mit STM32CubeMX wird der Firmware-Flow in STM32CubeIDE implementiert. Anschließend wird das Programm mit STM32CubeIDE kompliliertllen und auf das Board bzw. den Mikrocontroller geflasht..

Für die Kommunikation mit dem Board wird eine Qt-Anwendung entwickelt. Sie sendet den Befehl an den Mikrocontroller. Dieser empfängt den Befehl, und auf dessen Grundlage wird die entsprechende Anweisung ausgeführt.

2 Software-Details

Die Einzelheiten der in diesem Projekt verwendeten Software werden in diesem Abschnitt erörtert.

2.1 STM32CubeIDE

STM32CubeIDE wird für die Entwicklung der Firmware für die DAC-Kanal-Konfiguration verwendet. Zunächst ist das Projekt zu konfigurieren und der Code mit dem STM32CubeMX zu generieren. Dann wird das DAC-Kanal-Konfigurationsprojekt programmiert und in STM32CubeIDE kompiliert.

2.1.1 Installationsschritte

Für die Entwicklung wird STM32CubeIDE von der STMicroelectronics-Website heruntergeladen.

Der Installationsvorgang wird im folgenden Abschnitt beschrieben. Die Installation erfolgt mit dem Produktinstallationsprogramm. Zunächst ist das Installationsprogramm von der Website von STMicroelectronics herunterzuladen. Daraufhin sind die unten beschriebenen Schritte zu befolgen:

1) Produktinstallationsprogramm (.exe-Datei) starten
2) warten, bis der Willkommensdialog des Installationsprogramms angezeigt wird, und auf [Weiter>] klicken

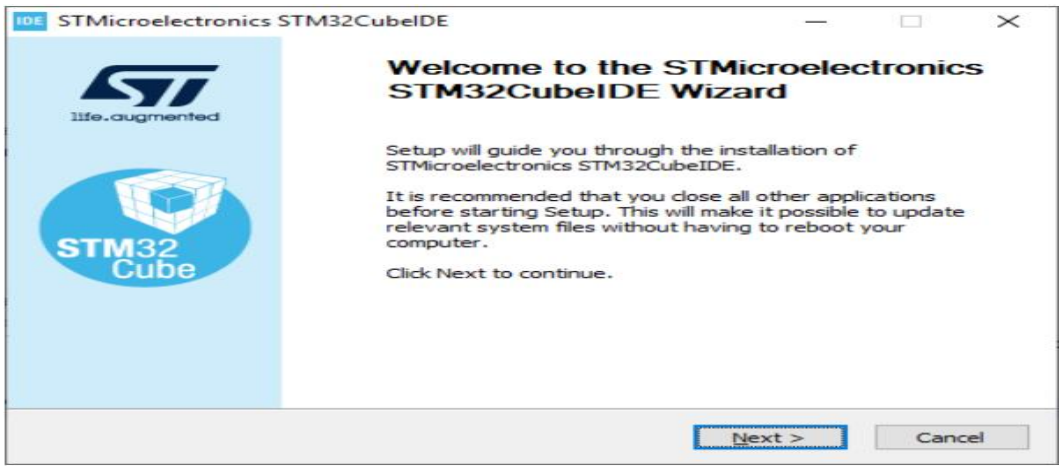


Abbildung 1: Willkommensseite für Installateure

3) Lizenzvereinbarung lesen und auf [Ich stimme zu] klicken, um die Bedingungen der Vereinbarung zu akzeptieren, oder auf [Abbrechen], um die Installation abzubrechen. Wenn die Vereinbarung akzeptiert wurde, wird der Installationsassistent fortgesetzt.

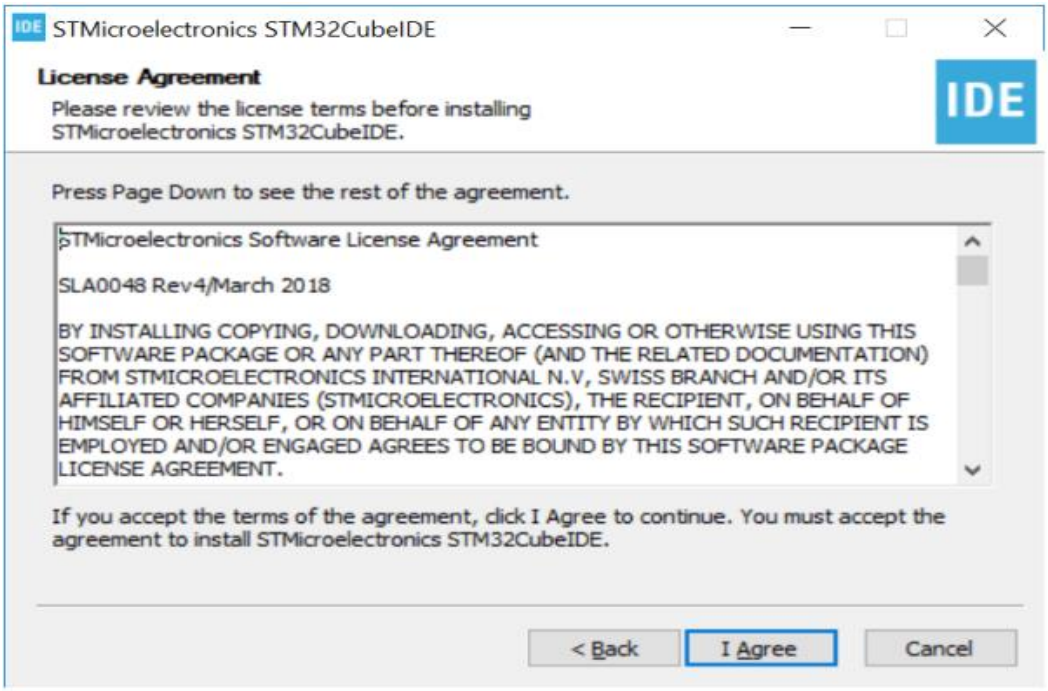


Abbildung 2: Dialogfeld Lizenzvereinbarung

4) In diesem Dialogfeld wählt der Benutzer den Speicherort für die Installation aus. Es wird empfohlen, einen kurzen Pfad zu wählen, um Windows®-Beschränkungen mit zu langen Pfaden für den Arbeitsbereich zu vermeiden.

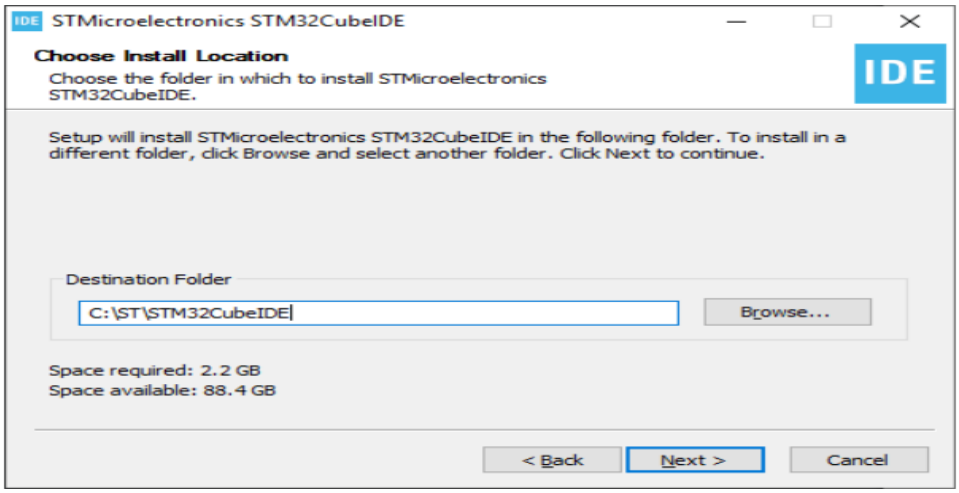


Abbildung 3: Dialogfeld für den Installationsort

5) warten, bis der Dialog Komponenten auswählen angezeigt wird, und die GDB-Server-Komponenten auswählen, die zusammen mit STM32CubeIDE installiert werden sollen. Für jede Art von JTAG-Tastkopf, der zum Debuggen mit STM32CubeIDE verwendet wird, ist ein Server erforderlich.

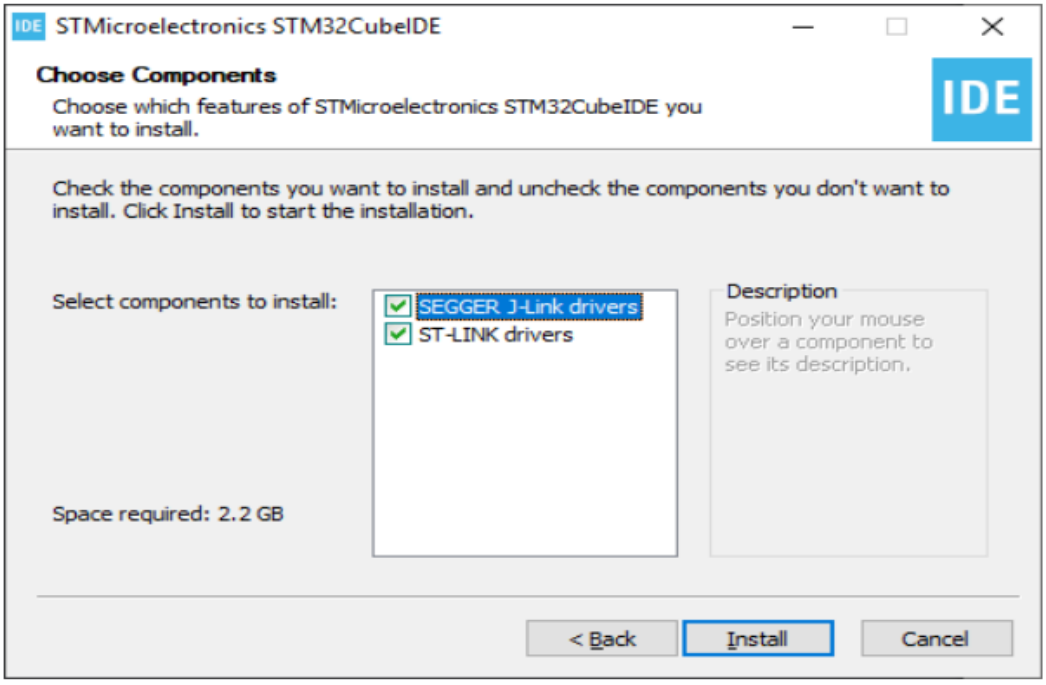


Abbildung 4: Dialog zur Auswahl von Komponenten

6) Auf [Installieren] klicken, um die Installation zu starten. Die ausgewählten Treiber werden parallel zur Installation von STM32CubeIDE installiert.

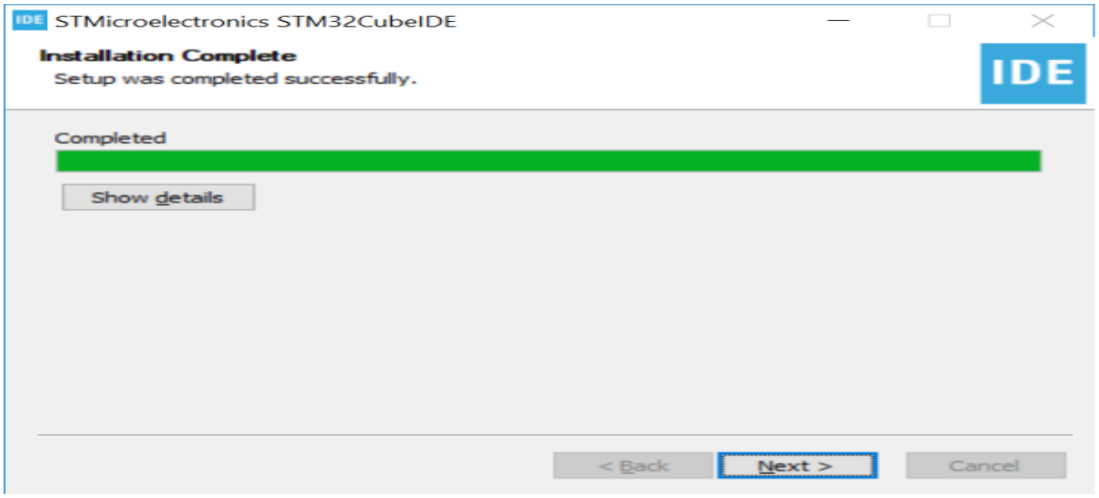


Abbildung 5: Erfolgreiche Installation

7) Auf [Weiter] klicken, um zum letzten Schritt des Installationsvorgangs zu gelangen. Dies ist ein Bestätigungsdialog, der den Benutzer darüber informiert, dass die Installation abgeschlossen ist. Sobald der Benutzer auf [Fertigstellen] geklickt hat, ist der Installationsvorgang abgeschlossen.

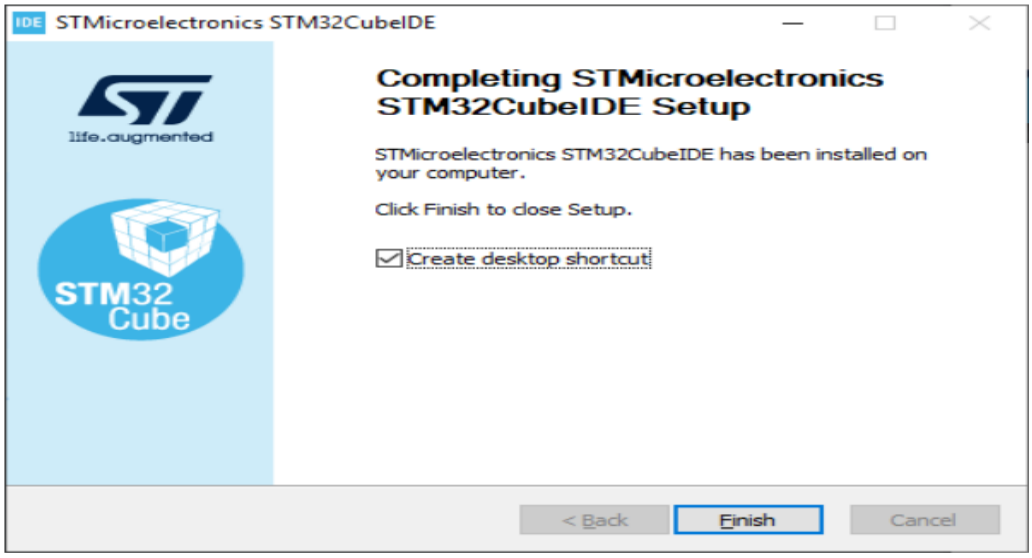


Abbildung 6: Vorgang beendet

2.2 Erstellen des STM32CubeIDE-Projekts

2.2.1 Schritt 1

Im Bereich Datei muss man auf Neu bzw. New klicken und das STM32Cube-Projekt auswählen.

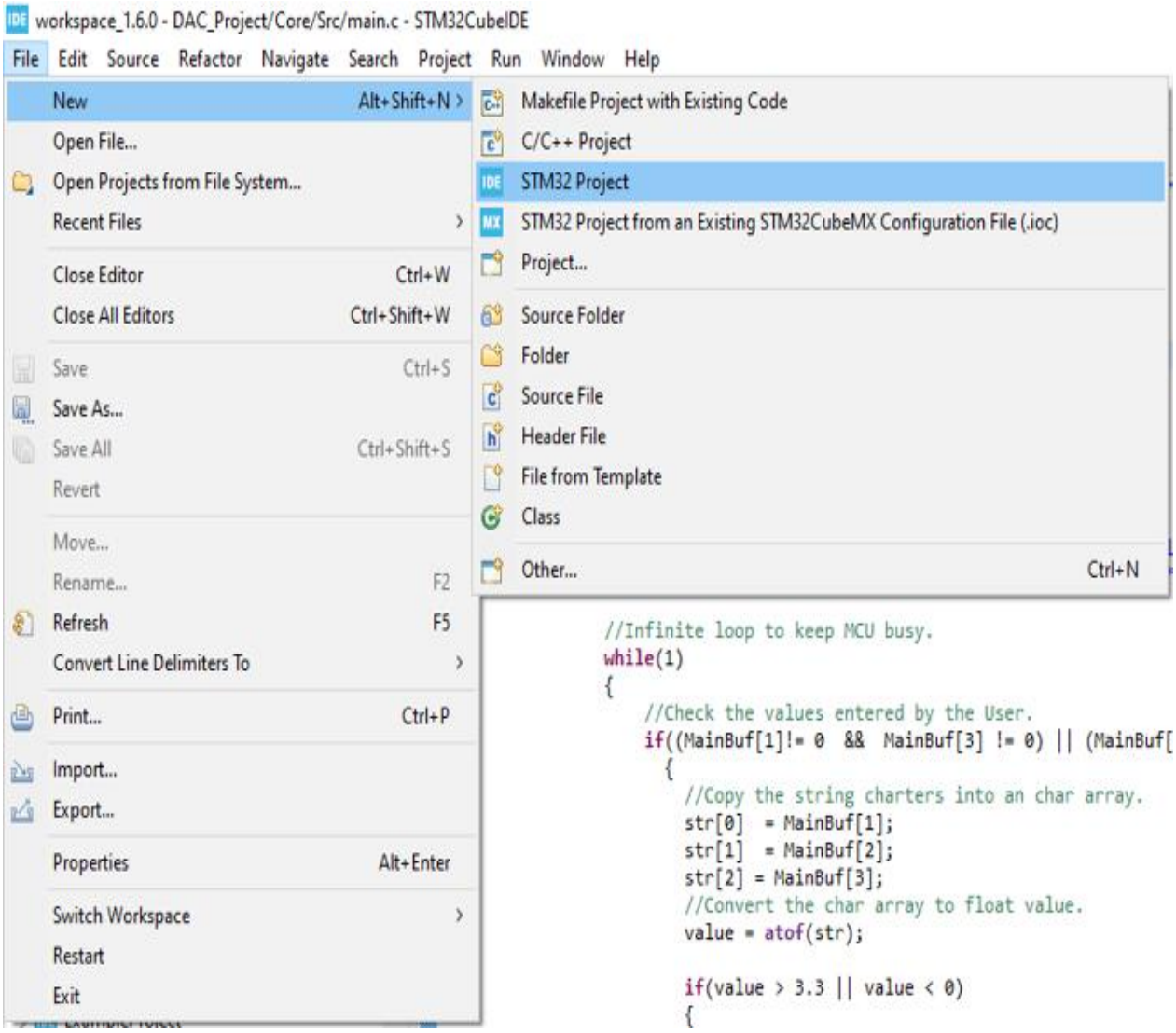


Abbildung 7: STM32Cube – Projekterstellung

Es dauert einige Zeit, bis der STM32 Target Selector initialisiert ist.

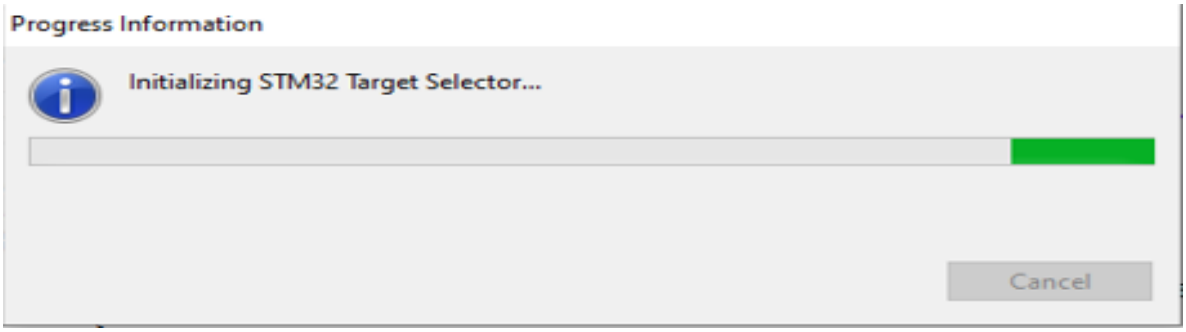


Abbildung 8: Initialisierung des STM32 Target Selectors

2.2.2 Schritt 2

Nun wird die Registerkarte "Board Selector" geöffnet Nucleo 64 ausgewählt und das Nucleo_L476RG Board gesucht. Anschließend wird auf die Schaltfläche Next geklickt.

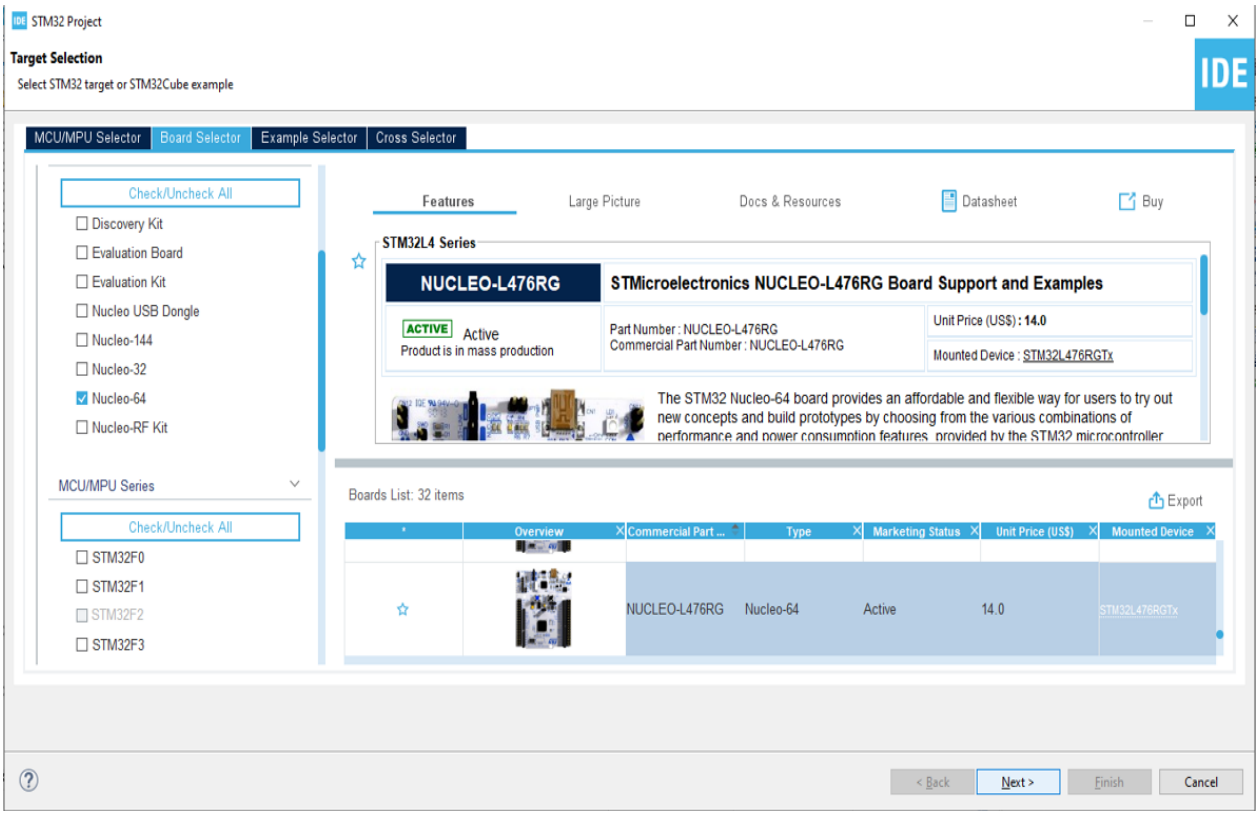


Abbildung 9: STM32 –Platinenauswahl

2.2.3 Schritt 3

Der Name des Projekts wird eingegeben und auf die Schaltfläche Fertigstellen geklickt.

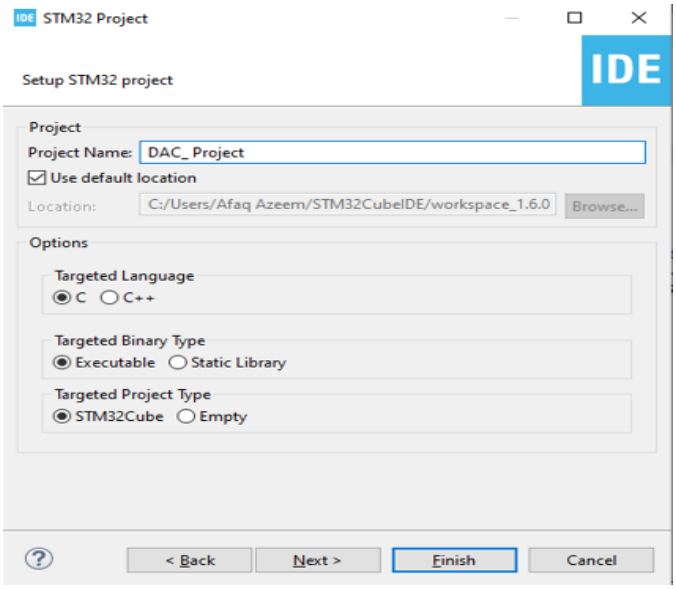


Abbildung 10: STM32 Projekt einstellen

Nachdem auf die Schaltfläche Fertigstellen gedrückt wurde, wird das Fenster Board Project Option angezeigt. Hier ist die Schaltfläche Auswählen anzuklicken.

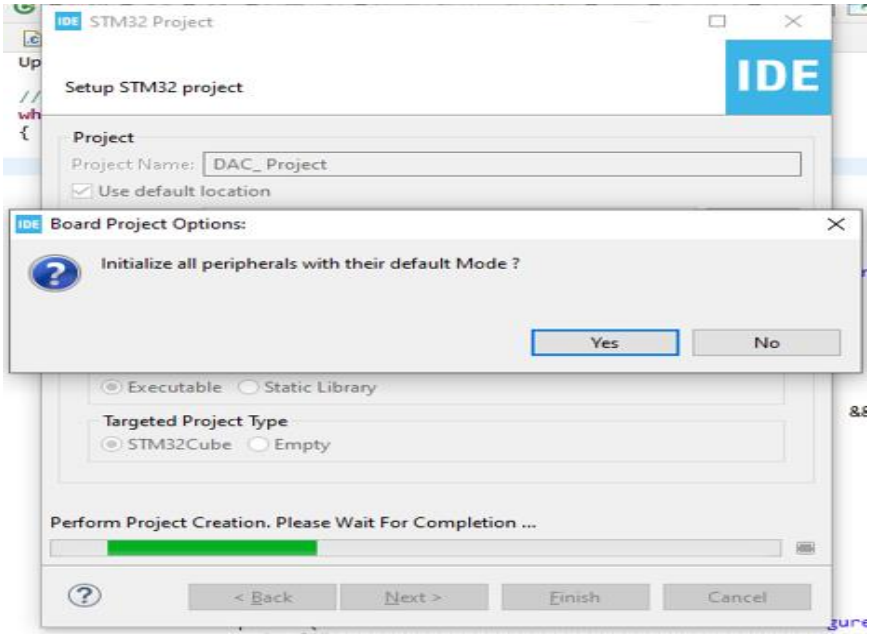


Abbildung 11: STM32 –Projekterstellung abgeschlossen

Danach generiert STM32CubeIDE die Konfigurationseinstellungen für das Nucleo_L476RG Board. Es initialisiert die Peripheriegeräte für das Board und generiert das grundlegende Projekt-Setup, um das Projekt auf dem Nucleo Board implementieren zu können..

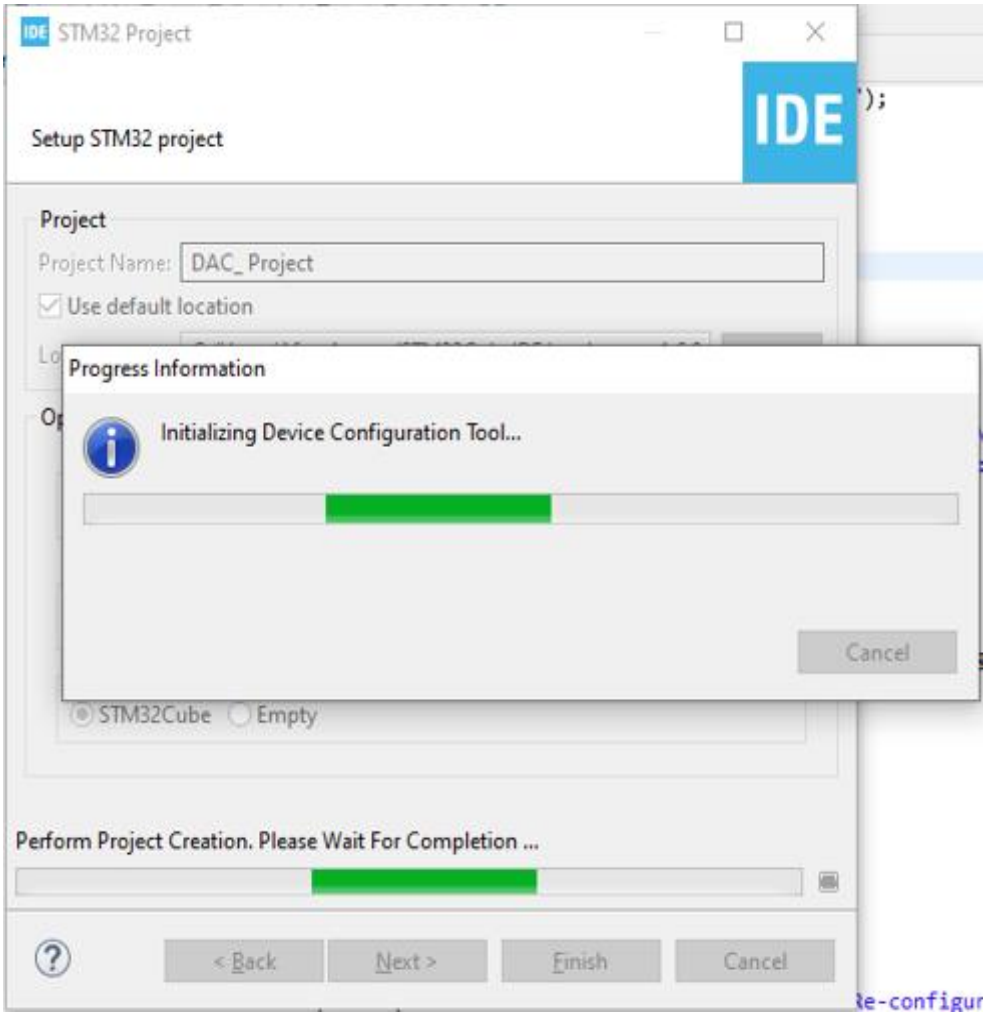


Abbildung 12: Initialisierung der STM32-Karte – Peripheriegeräte

2.3 STM32-Debugger-Einstellung

Nachdem das Projekt erstellt wurde, wird die Schaltfläche Build gedrückt, um das Projekt zu generieren. Um das Projekt anschließend zu debuggen, werden die Debug-Einstellungen durch Drücken der Debug-Schaltfläche gewählt.

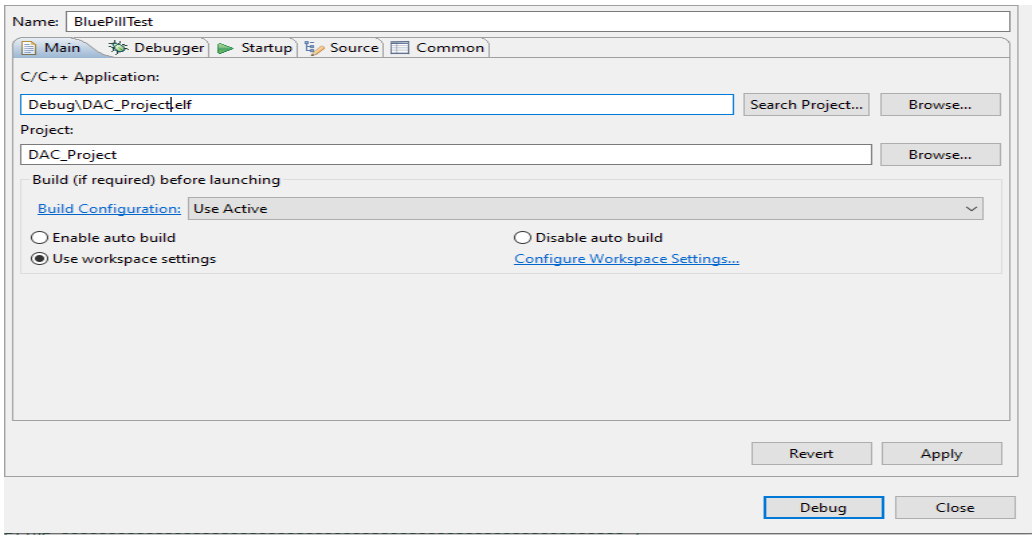


Abbildung 13: STM32-Debugger-Einstellungen

Wenn der ST_link verwendet wird, dann werden die Einstellungen so gelassen, wie sie sind, und falls ein anderer Debugger verwendet wird, lassen sich die Einstellungen der Debugger-Option im Fenster ändern.

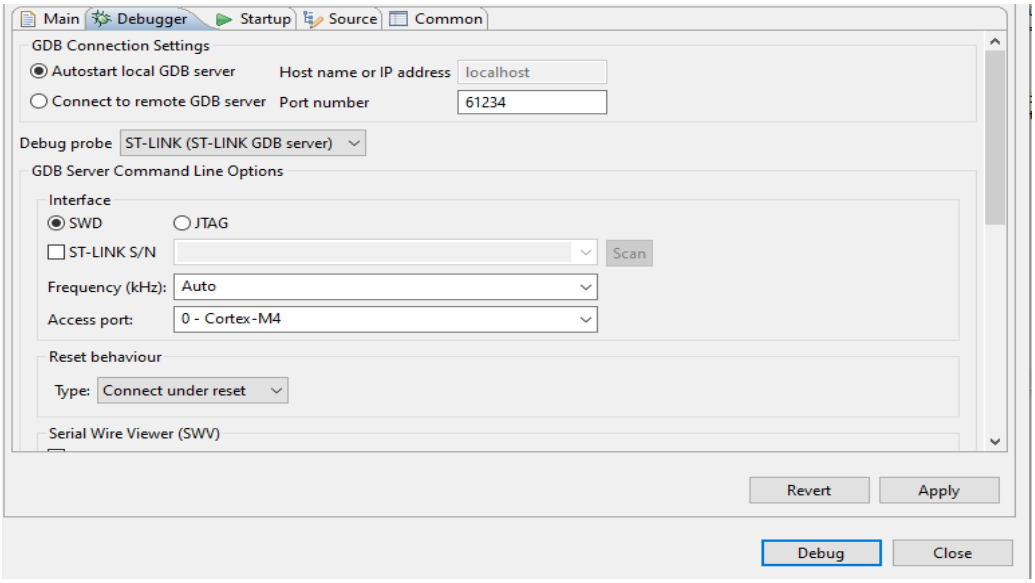


Abbildung 14: STM32 ST-LINK Einstellungen

3 Hardware-Details

3.1 STM32L476 Nucleo-Platine

3.1.1 Beschreibung

Das STM32 Nucleo-64-Board bietet eine erschwingliche und flexible Möglichkeit für Benutzer, neue Konzepte auszuprobieren und Prototypen zu bauen, indem sie aus den verschiedenen Kombinationen von Leistungs- und Stromverbrauchsmerkmalen wählen, die der STM32-Mikrocontroller bietet. Bei den kompatiblen Boards reduziert das externe SMPS den Stromverbrauch im Run-Modus erheblich. Die Unterstützung der ARDUINO Uno V3-Konnektivität und der ST-Morpho-Header ermöglichen eine einfache Erweiterung der Funktionalität der offenen STM32 Nucleo-Entwicklungsplattform mit einer großen Auswahl an spezialisierten Shields. Das STM32 Nucleo-64-Board benötigt kein separates Kommunikations-Interface, da es über den ST-LINK Debugger/Programmer verfügt. Das STM32 Nucleo-64-Board wird mit den umfassenden kostenlosen STM32-Softwarebibliotheken und Beispielen geliefert, die mit dem STM32Cube MCU-Paket erhältlich sind.

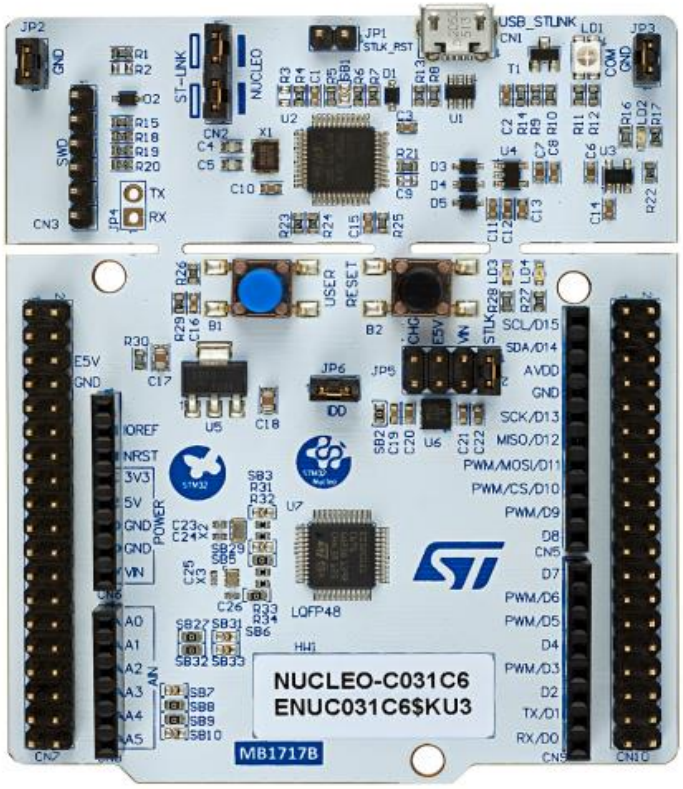


Abbildung 15: STM32476 Nucleo-Platine

3.1.2 MCU-Details

STM32L476RGT6U	
Hersteller	STMicroelectronics
Serie	STM32L4
Paket	Tablett
Teilstatus	Ausgelaufen bei Digi-Key
Kernprozessor	ARM® Cortex®-M4
Kerngröße	32-Bit Single-Core
Geschwindigkeit	80MHz
Konnektivität	CANbus, I²C, IrDA, LINbus, MMC/SD, QSPI, SAI, SPI, SWPMI, UART/USART, USB OTG
Peripheriegeräte	Brown-out-Erkennung/Rücksetzung, DMA, LCD, PWM, WDT
Anzahl der E/A	51
Programmspeichergröße	1MB (1M x 8)
Programmspeicher Typ	FLASH
EEPROM-Größe	-
RAM-Größe	128K x 8
Spannung - Versorgung (Vcc/Vdd)	1,71V ~ 3,6V
Datenumwandler	A/D 16x12b; D/A 2x12b
Oszillatortyp	Intern
Betriebstemperatur	-40°C ~ 85°C (TA)
Montageart	Oberflächenmontage
Verpackung/Koffer	64-LQFP
Lieferant Gerätepaket	64-LQFP (10x10)
Basis-Produktnummer	STM32L476

Tabelle 1: STM32L476RGT6U

4 Peripheriegeräte – Details

4.1 Digital/Analog-Wandler (DAC)

4.1.1 Theorie

Ein DAC-Wandler (Digital-Analog-Wandler) ist ein elektronischer Schaltkreis, der eine digitale Zahl oder einen digitalen Wert als Eingangssignal aufnimmt und in eine analoge Spannung, d. h. in den Spannungswert umwandelt, welcher der binären Zahl im DAC-Ausgangsregister entspricht. Die DAC-Ausgangsspannung ändert sich immer dann, wenn der Wert des DAC-Ausgangsregisters geändert wird, und dieser Abtastprozess kann auf verschiedene Weise ausgelöst werden.

Während ein ADC (A/D) analoge Spannungen in digitale Daten umwandelt, wandelt der DAC (D/A) digitale Zahlen in analoge Spannung am Ausgangspin um.

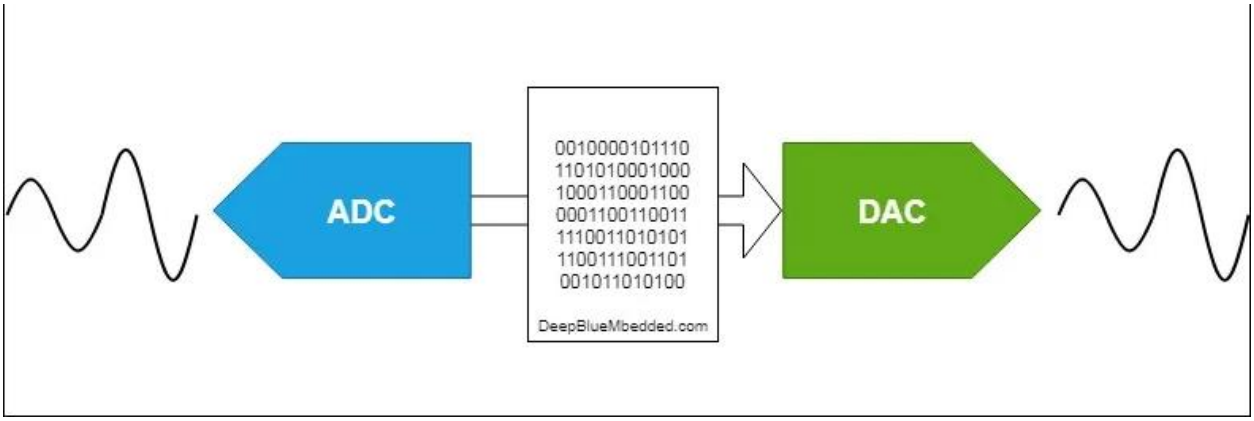


Abbildung 16: DAC- und ADC-Umwandlung

Nicht alle Mikrocontroller verfügen über eine On-Chip-DAC-Peripherie, weshalb externe DAC-ICs oder Techniken wie PWM-DAC-Wandlung angewendet werden.

In diesem Projekt wird das STM32L476 Nucleo Board verwendet, da es über einen DAC mit zwei konfigurierbaren Kanälen verfügt.

4.1.2 STM32 DAC – Kurzbeschreibung

Im STM32L476RG ist das DAC-Modul ein 12-Bit-Digital-Analog-Wandler mit Spannungsausgang. Der DAC kann im 8- oder 12-Bit-Modus konfiguriert und in Verbindung mit dem DMA-Controller verwendet werden. Im 12-Bit-Modus sind die Daten links- oder rechtsbündig. Der DAC verfügt

über bis zu zwei Ausgangskanäle, jeder mit eigenem Wandler. Im Modus mit zwei DAC-Kanälen können die Umwandlungen unabhängig voneinander oder gleichzeitig erfolgen, wenn beide Kanäle für synchrone Aktualisierungsvorgänge gruppiert sind.

Der DAC_OUTx-Pin kann als Allzweck-Eingang/Ausgang (GPIO) verwendet werden, wenn der DAC-Ausgang vom Ausgangspad getrennt und mit On-Chip-Peripheriegeräten verbunden ist. Der DAC-Ausgangspuffer kann optional aktiviert werden, um einen hohen Treiberausgangsstrom zu ermöglichen. Auf jeden DAC-Ausgangskanal kann eine individuelle Kalibrierung angewendet werden. Die DAC-Ausgangskanäle unterstützen einen stromsparenden Modus, den Sample-and-Hold-Modus.

4.1.3 STM32 DAC-Blockdiagramm

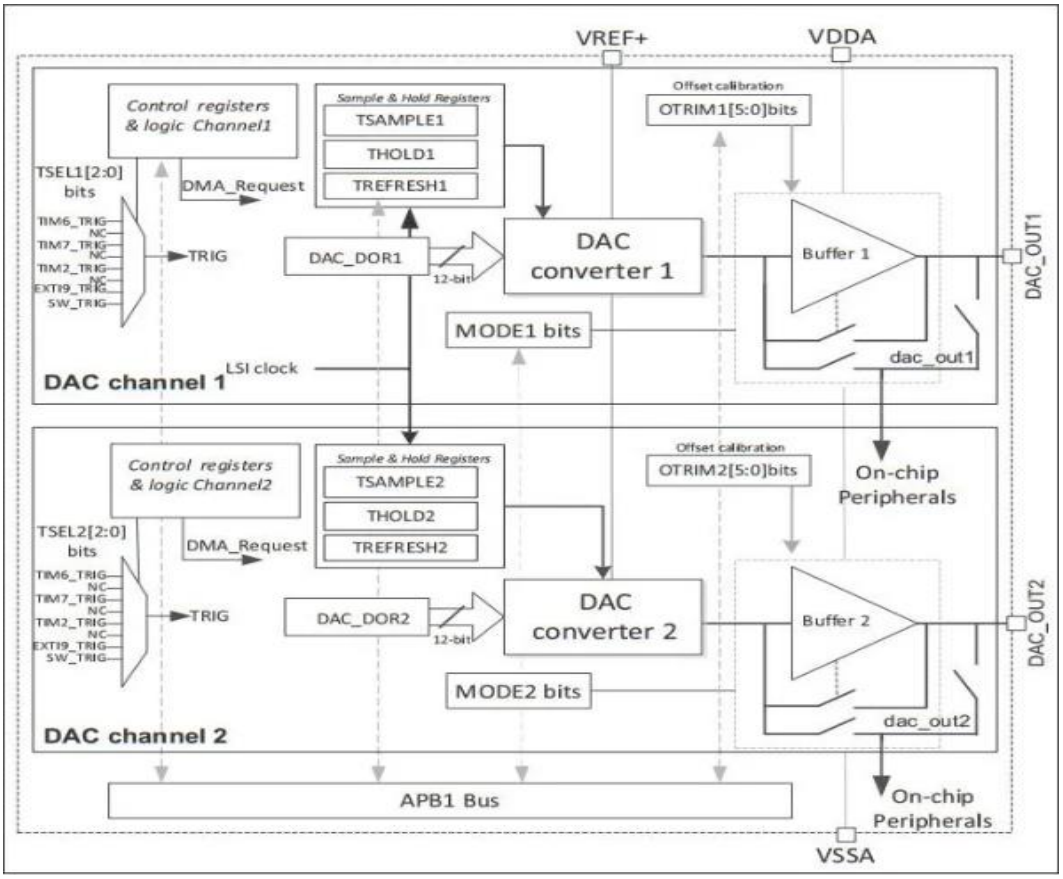


Abbildung 17: STM32 DAC-Blockdiagramm

Jeder der beiden DAC Ausgangskanal kann mit On-Chip-Peripheriegeräten wie z.B. dem Komparator, dem Operationsverstärker und dem ADC werden. In diesem Fall kann der DAC-Ausgangskanal vom DAC_OUTx-Ausgangspin getrennt und der entsprechende GPIO für einen anderen Zweck verwendet werden.

Jeder DAC-Kanal kann durch Setzen des entsprechenden ENx-Bits im DAC_CR-Register eingeschaltet werden. Der DAC-Kanal wird dann nach einer tWAKEUP-Startzeit aktiviert.

4.1.4 STM32 DAC-Auflösung

Der STM32 DAC hat eine Auflösung von 12-Bit, die auch auf 8-Bit konfiguriert werden kann. Je nach Anwendung ist zwischen den beiden Optionen zu wählen.

4.1.5 DAC-Referenzspannung

Die DAC-Referenzspannung kann entweder über einen externen Pin oder intern über das interne VREFBUF-Modul bereitgestellt werden. Die Referenzspannung, die für den DAC-Betrieb gewählt wird, bestimmt den maximal zulässigen Spannungshub sowie die Auflösung der Ausgangsspannung.

4.1.6 Berechnung der DAC Ausgangsspannung

Zur Berechnung der DAC Ausgangsspannung DAC_OUTPUT kann die folgende Gleichung verwendet werden.

(1)

$$DAC_{Output} = V_{REF}\frac{DOR}{DAC\_MaxDigitalValue + 1}$$

Gleichung 1:

Dabei entspricht V_REF der Spanungsreferenz. Bei Verwendung der internen Spannungsreferenz beträgt diese Spannung xyz V. DOR entspricht dem digitalen Wert, der in eine analoge Spannung gewandelt werden soll. DAC_MaxDigitalValue+1 ist die maximale Wert den DOR annehmen kann.

Bei 12-Bit-Auflösung: besitzt DAC_MaxDigitalValue den Wert 0xFFF d.h. 4095

Bei 8-Bit-Auflösung: besitzt DAC_MaxDigitalValue den Wert 0xFF d.h. 255

4.1.7 STM32 DAC – gepufferter Ausgang vs. ungepufferter Ausgang

Jede elektronische Schaltung hat eine Eingangsimpedanz, die als Lastwiderstand für die dieser Schaltung vorgelagerte Stufe darstellt. Bei einem DAC wird die Ausgangsspannung belastet, wenn der Benutzer den DAC_OUT-Pin mit einer Schaltung verbindet, welche die Spannung verstärkt oder filtert. Das Vorhandensein eines Ausgangslastwiderstands zieht einen Strom aus dem DAC, was zu einer Spannungsverschiebung bzw. einem Spannungsabfall am DAC-Ausgang führt.

Daher ist der DAC-Ausgang immer zu puffern, wenn er durch eine andere elektronische Schaltung stark belastet werden soll. Ein Ausgangspuffer ist ein Operationsverstärker, der in einer Spannungsfolger-Konfiguration verschaltet ist, Der Ausgangspuffer liefert den benötigten Strom und stellt sicher , dass die Spannung am Ausgang des Puffers identisch mit der DAC Spannung ist. In Abbildung 19 ist ein Vergleich zwischen einem gepufferten und einem ungepufferten DAC-Ausgang dargestellt. Beide werden in Konfigurationen mit und ohne Last getestet, und beide sind so programmiert, dass sie 1,5 V am DAC_OUT-Pin erzeugen.

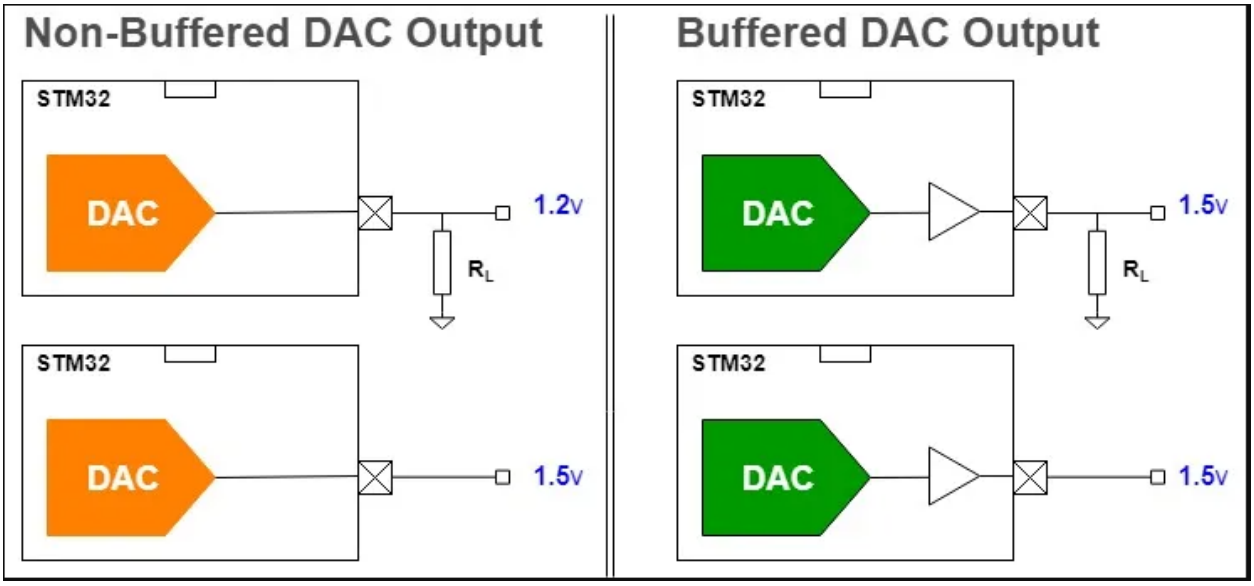


Abbildung 18: DAC – gepufferter vs. ungepufferter Ausgang

4.2 DAC-Projekt-Konfiguration:

In diesem Projekt werden Beide DAC-Kanäle des STM32L476 Nucleo Boards verwendet, um eine Gleichspannung zwischen 0V und 3.3V zu erzeugen.

Die Einstellungen, welche für die DAC-Kanäle im STM32CubeMX anzugeben sind, sind in Abbildung 20 dargestellt.

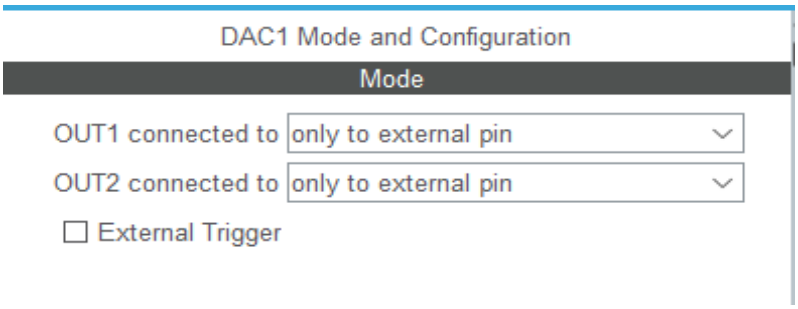


Abbildung 19: DAC-Kanäle – Pin-Einstellungen

Die beiden DAC Ausgänge OUT1 und OUT2 sind mit externen Pins der MCU verbunden. Keine interne Peripherie ist mit diesen Pins verbunden. Außerdem wird kein externer Trigger für den DAC-Ausgang verwendet.

Der Ausgang des DAC-Kanals 1 liegt dabei auf Pin PA4 und der Ausgang des DAC-Kanals 2 auf Pin PA5 des Mikrocontrollers , wie in der folgenden Abbildungen 20 zu sehen ist.

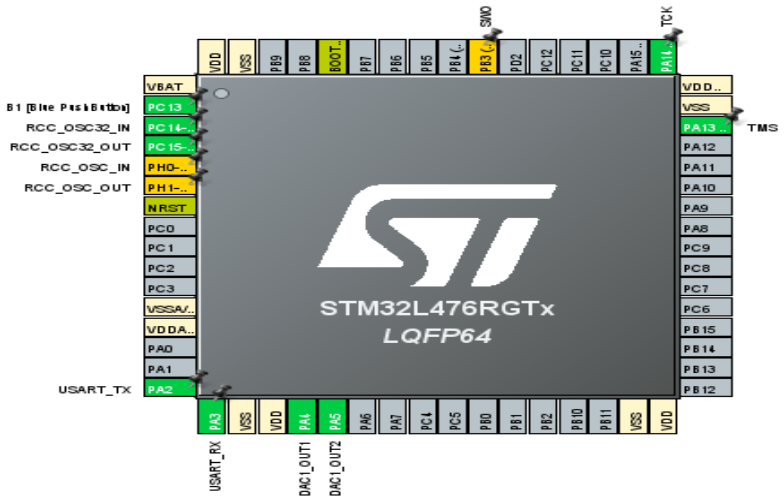


Abbildung 20: STM32l476RG-Anschlussplan

Weitere Einstellungen der DAC-Kanäle sind in Abbildung 21 zu sehen. Der Ausgangspuffer ist aktiviert, jedoch kein Trigger. Das User-Trimmingist auf die Werkseinstellungen eingestellt. Es wird keine Sample-and-Hold-Schaltung verwendet. Beide Kanäle haben die gleichen Konfigurationen.

Außerdem wird kein DMA und kein NVIC-Interrupt verwendet. Es werden keine Benutzerkonstanten eingesetzt.

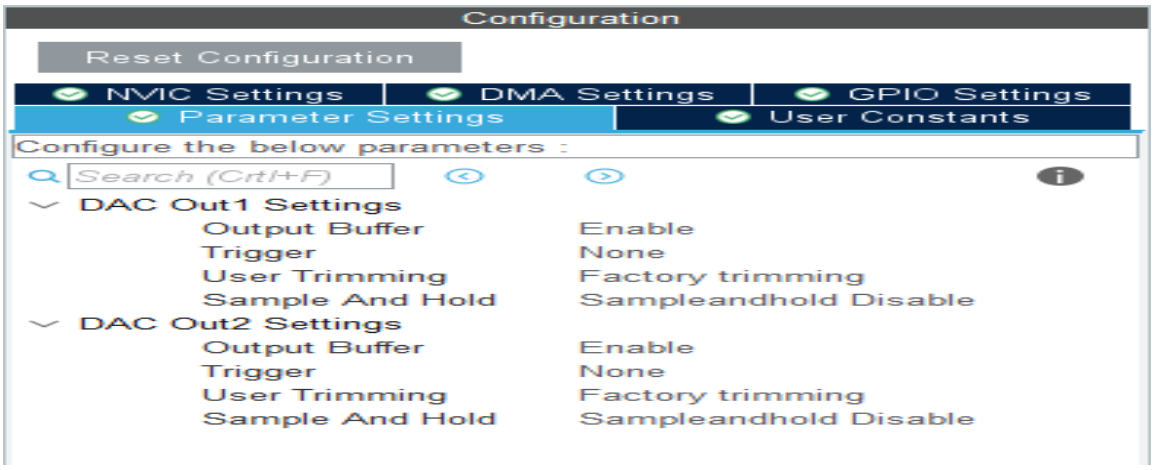
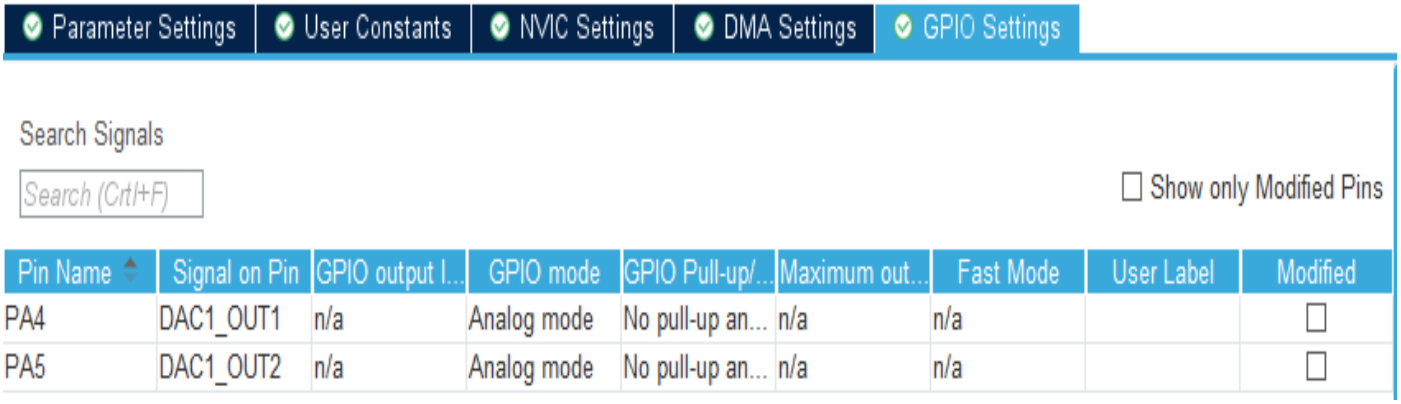


Abbildung 21: Einstellungen der DAC-Parameter

Die GPIO-Einstellungen für beide Pins sind in Abbildungen 22 dargestellt.



Pin Name	Signal on Pin	GPIO output l...	GPIO mode	GPIO Pull-up/...	Maximum out...	Fast Mode	User Label	Modified
PA4	DAC1_OUT1	n/a	Analog mode	No pull-up an...	n/a	n/a		☐
PA5	DAC1_OUT2	n/a	Analog mode	No pull-up an...	n/a	n/a		☐

Abbildung 22: DAC-GPIO-Einstellungen

4.3 Universeller synchroner asynchroner Empfänger/Sender (USART)

4.3.1 Theorie

Universal Asynchronous Receiver/Transmitter (UART) steht für die Hardware-Schaltung (Modul), die für die serielle Kommunikation verwendet wird. Das UART Interface wird als eigenständiger integrierter Schaltkreis (IC) oder als internes Modul in Mikrocontrollern verkauft bzw. ausgeliefert. In diesem Projekt steht das interne UART-Modul des STM32-Mikrocontrollers im Fokus.

Es gibt zwei Arten von UART-Hardware:

UART – Universal Asynchronous Receiver/Transmitter

USART – Universal Synchronous/Asynchronous Receiver/Transmitter.

Der synchrone Sendertyp erzeugt den Datentakt und sendet ihn an den Empfänger, der entsprechend synchronisiert arbeitet. Der asynchrone Sendertyp hingegen erzeugt den Datentakt intern. Und es gibt kein eingehendes serielles Taktsignal. Um eine ordnungsgemäße Kommunikation zwischen den beiden Enden zu erreichen, müssen beide die gleiche Baudrate verwenden.

4.3.2 USART/UART-Hardware in STM32

USART Schnittstellen stellt ein flexibles Mittel für den Vollduplex-Datenaustausch mit externen Geräten dar, die ein dem Industriestandard x Non-Return-To-Zero entsprechendes asynchrones serielles Datenformat benötigen. UARTS bieten bei Verwendung eines fraktionalen Baudratengenerators einen sehr großen Bereich von Baudraten an.

Das System unterstützt synchrone Einwegkommunikation und Halbduplex-Einzeladerkommunikation, außerdem die Spezifikationen LIN (Local Interconnection Network), Smartcard Protocol, IrDA (Infrared Data Association), SIR ENDEC sowie Modemoperationen (CTS/RTS). Es ermöglicht die Kommunikation mit mehreren Prozessoren. Eine Hochgeschwindigkeits-Datenkommunikation ist durch die Verwendung des DMA in einer Multi-Buffer-Konfiguration möglich.

4.3.3 STM32 USART – Hardware-Funktionalitäten

Jede bidirektionale USART-Kommunikation erfordert mindestens zwei Pins: Receive Data In (RX) und Transmit Data Out (TX). Über diese Pins werden serielle Daten im normalen USART-Modus gesendet und empfangen. Der CK-Pin ist für die Schnittstelle im synchronen Modus erforderlich. Die CTS- und RTS-Pins sind für den Hardware-Flusskontrollmodus notwendig.

4.3.4 USART-Blockdiagramm

Wie im digitalen Blockdiagramm für ein UART-Hardwaremodul leicht zu erkennen ist, gibt es zwei getrennte Schieberegister und doppelt gepufferte Eingangs-/Ausgangsdaten für einen Vollduplex-Datenübertragungs- und -empfangsbetrieb. Beide Schieberegister, die die Daten während des Empfangs bzw. der Übertragung ein- oder ausschieben, werden mit der Rate des (Baudratengenerators) am unteren Rand des Diagramms getaktet.

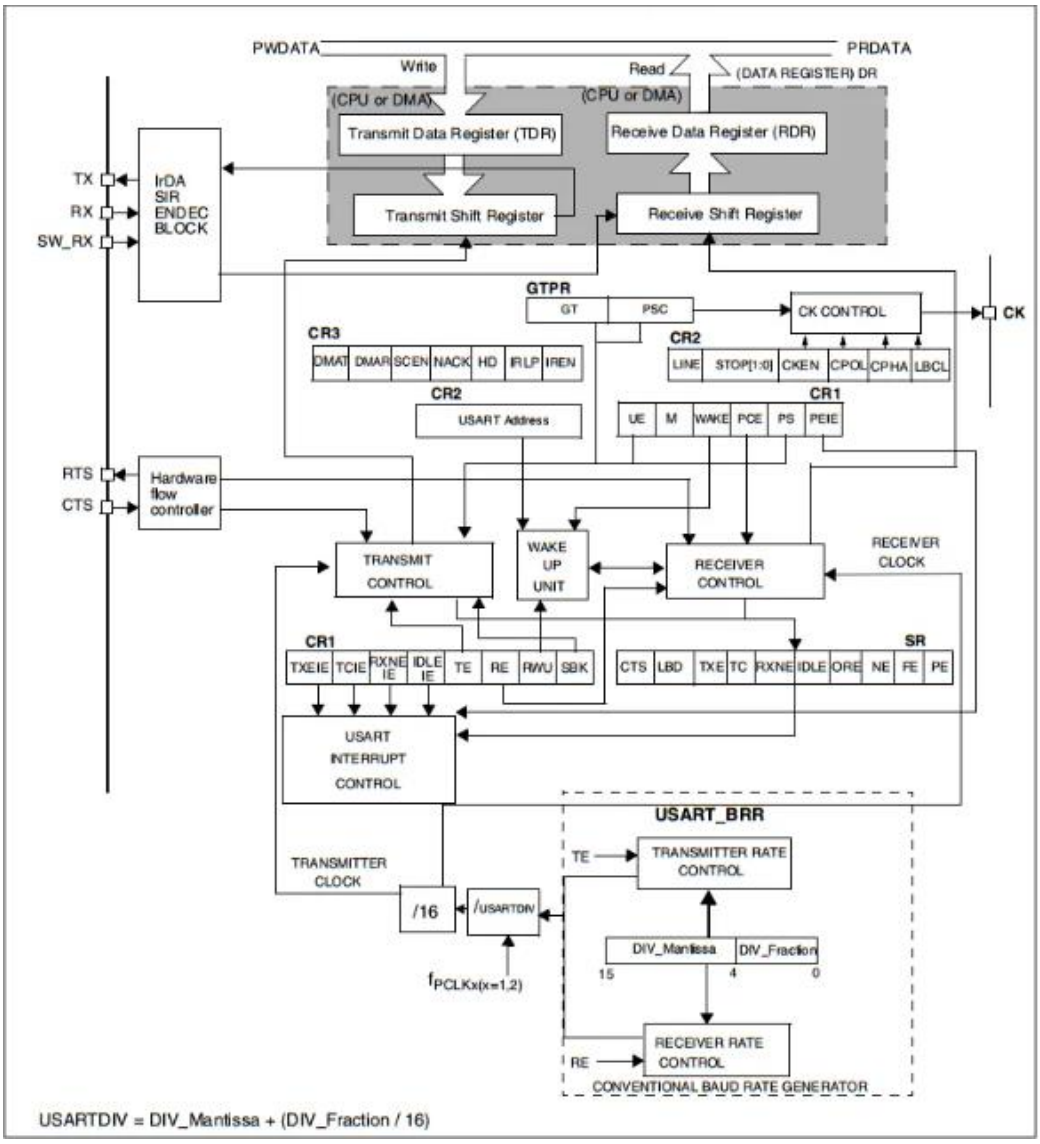


Abbildung 23: STM32 USART-Blockdiagramm

Es gibt ein Adressregister für den Multiprozessor-Kommunikationsmodus, eine Hardware-Datenflusskontrolleinheit zur Unterstützung dieser Funktion, einen IrDA-Decoder-Schaltkreis und eine Interrupsteuerungseinheit, um verschiedene Interrupsignale bei verschiedenen USART-Hardware-Ereignissen zu erzeugen.

4.4 USART-Projekt-Konfigurationen

In diesem DAC-Projekt wird die USART-Peripherie des Nukleo-Boards im asynchronen Modus ohne Hardware-Flusskontrolle verwendet. Die USART-Kommunikation wird mit einer Baudrate von 115200 konfiguriert. Die Wortlänge beträgt 8 Bits. Es wird keine Parität verwendet. Es wird ein Stoppbit genutzt.

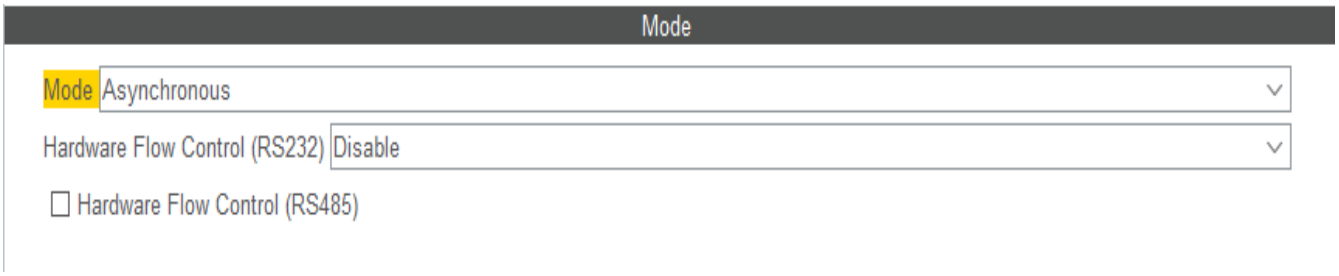


Abbildung 24: USART-Modus-Einstellungen

In den USART-Parametereinstellungen werden keine erweiterten Funktionen eingestellt. In den erweiterten Parametern ist die Datenrichtung Empfang und Senden konfiguriert. Die Überabtastrate beträgt 16, und die Einzelabtastung ist deaktiviert.

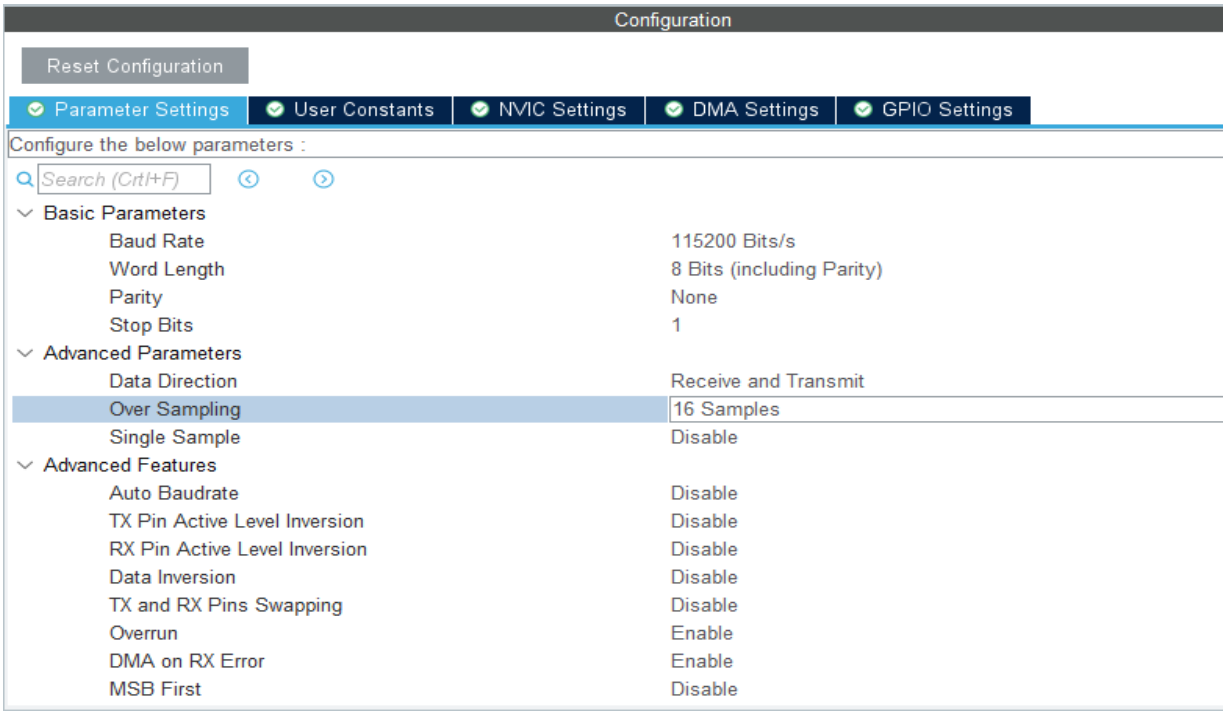


Abbildung 25: USART-Parametereinstellungen

In der NVIC-Einstellung ist der globale DMA-Interrupt aktiviert, der globale USART-Interrupt ebenfalls.

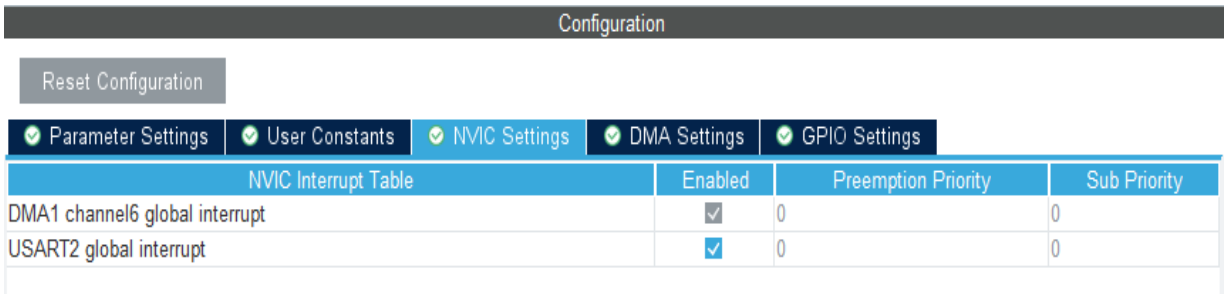


Abbildung 26: USART NVIC-Einstellungen

In den DMA-Einstellungen ist der DMA nur für den RX-Pin konfiguriert, und es wird Kanal 6 verwendet. Die Datenübertragungsrichtung erfolgt vom Peripheriegerät zum Speicher. Die Priorität der DMA-Interrupts wird niedrig gehalten.

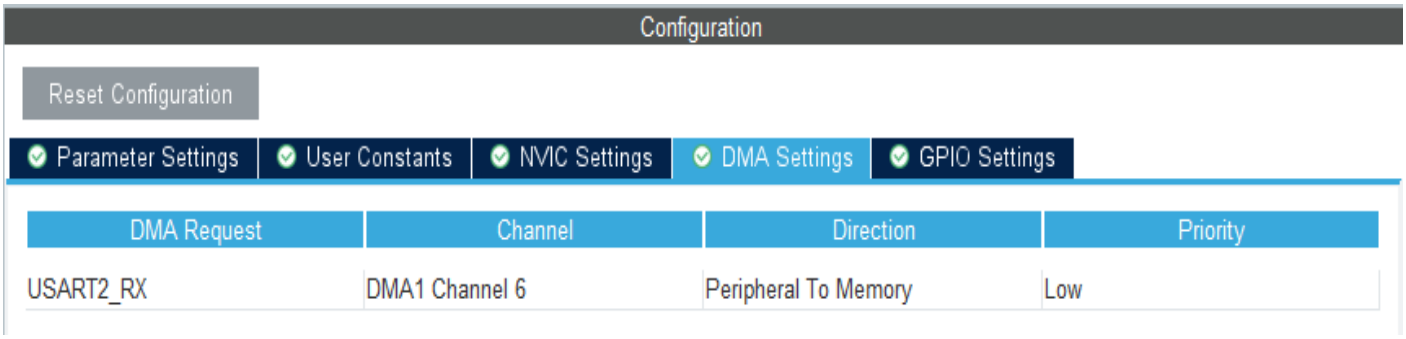


Abbildung 27: USART-DMA-Einstellungen

In den USART-GPIO-Einstellungen wird der PA2-Pin für das Senden und der PA3-Pin für den Empfang verwendet. An diesen Pins wird kein Pull-up-Down verwendet.

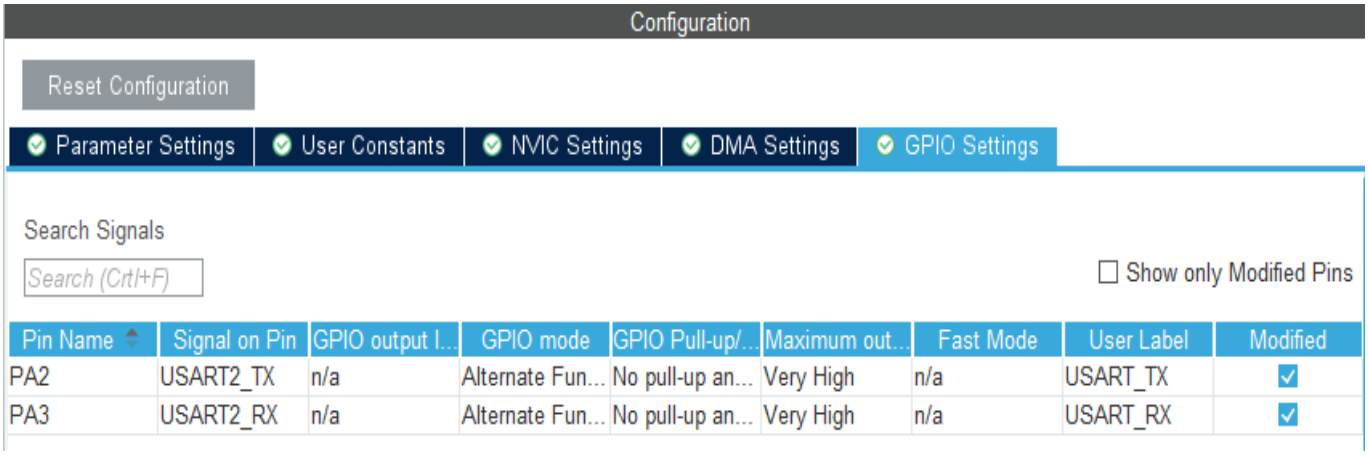


Abbildung 28: USART-Pin-Einstellungen

5 Firmware-Entwurf

Es wurde ein einfaches Firmware-Design implementiert, das beim Programmstart auf den Empfang eines neuen Befehls über UART wartet. Der UART ist interruptbasiert und befindet sich stets im Datenempfangsmodus. Wann immer Daten auf dem UART verfügbar sind, unterbricht der Mikrocontroller die laufende Ausführung, empfängt Daten vom UART und führt die UART-Interrupt-Handler-Funktion aus. Anschließend wird die reguläre Ausführung wieder aufgenommen. In der aktuellen Implementierung sucht der Mikrocontroller nach dem Zeichen "\n". Wenn er es empfangen hat, bedeutet dies, dass ein neuer Befehl vollständig empfangen wurde. Nach dem vollständigen Empfang des Befehls dekodiert der Mikrocontroller den empfangenen Befehl, und führt auf dessen Grundlage wird die entsprechende Funktion aus.

Die implementierten Befehle auf die der Mikrocontroller reagieren soll, sind in Tabelle 2 aufgelistet. Prinzipiell können Befehle, die zur Konfiguration verwendet werden und Befehle, die Auskunft über die aktuelle Konfiguration geben, unterschieden werden. Letztere Befehle, die eine Antwort des Mikrocontrollers provozieren, enden mit einem Fragezeichen. Mit dem Befehl Instrument:Select wird einer der beiden Kanäle selektiert. Mit dem BEfehl SOURce:VOLTage:LEVel wird die Ausgangsspannung des selektierten DAC Kanals gesetzt.

Ein Flussdiagramm der Firmware ist in Abbildung 29 dargestellt.

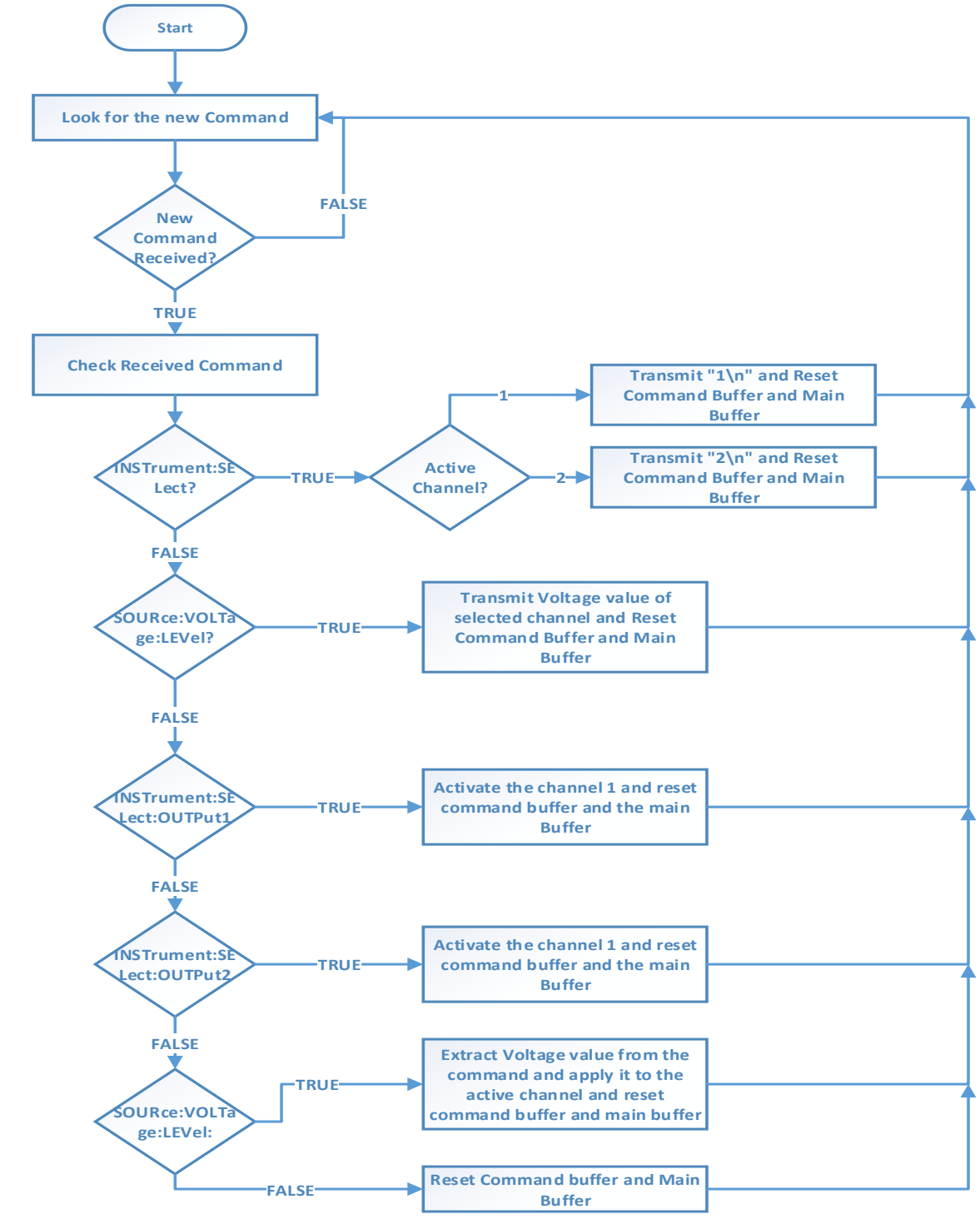


Abbildung 29: Firmware-Flussdiagramm

5.1 Details der Befehle

Nein.	Befehle	Funktion
1	INSTrument:SELect?	Dieser Befehl wird verwendet, um den Mikrocontroller nach dem aktiven DAC-Kanal zu fragen. Wenn dieser Befehl empfangen wird, sendet der Mikrocontroller den Wert des aktiven Kanals zurück.
2	SOURce:VOLTage:LEVel?	Dieser Befehl wird verwendet, um den Mikrocontroller nach dem Spannungswert des aktiven DAC-Kanals zu fragen. Wenn dieser Befehl empfangen wird, sendet der Mikrocontroller den Wert der Spannung des aktiven Kanals zurück.
3	INSTrument:SELect:OUTPut1	Dieser Befehl wird verwendet, um den Kanal zu aktivieren. Jetzt ist der Kanal 1 der aktive Kanal.
4	INSTrument:SELect:OUTPut2	Dieser Befehl wird verwendet, um den Kanal zu aktivieren. Jetzt ist der Kanal 1 der aktive Kanal.
5	SOURce:VOLTage:LEVel:	Dieser Befehl wird verwendet, um den gewünschten Spannungswert an den Ausgang des DAC-Kanals anzulegen. Wenn dieser Befehl empfangen wird, wird der Spannungswert aus dem Befehl extrahiert und der entsprechende digitale Wert berechnet und an den jeweiligen Kanal angelegt.

Tabelle 2: DAC-Konfigurationsbefehle

6 Erläuterung der Firmware

Zunächst wird STM32CubeMX für die Projekterstellung verwendet, und die Peripheriegeräte werden mit dem STM32CubeMx konfiguriert. Dann wird der Initialisierungscode mit STM32CubeMX erzeugt.

Nach der Erstellung des Projekts und der Generierung des Initialisierungscodes wird der Anwendercode für die Anwenderfunktionalitäten in das zuvor generierte Projekt integriert.

In diesem Projekt beginnt der Benutzercode mit den Defines und den Benutzervariablen, die im Projekt für die verschiedenen Zwecke verwendet werden. Ab der Zeile 69-99 besteht der Code aus Benutzerdefinitionen und den Variablen, die im Projekt für verschiedene Funktionen verwendet werden, deren Details nachstehend aufgeführt sind:

In der Zeile 71 wird die Größe des Empfangspuffers definiert und die Größe des Hauptpuffers und des Befehlspuffers in den Zeilen 72 bzw. 73 bestimmt. Diese Definitionen werden im Code verwendet, um Änderungen einfach umsetzen zu können, sodass diese Größen nicht überall im Code geändert werden müssen, sondern bei Bedarf an einer Stelle angepasst können.

Der Empfangspuffer und der Hauptpuffer werden in den Zeilen 75 bzw. 76 definiert. Diese Puffer werden für den Empfang von Daten über die UART Schnittstelle verwendet. Der Rx-Puffer wird zur Übertragung von Daten an den Hauptpuffer verwendet und nach jeder Ausführung des UART-Interrupt-Handlers zurückgesetzt. Die Daten des Hauptpuffers stehen, solange er nicht zurückgesetzt wird, für die weitere Bearbeitung zur Verfügung. Die Variable value in Zeile 77 enthält den über den UART empfangenen Spannungswerts, der mit Hilfe des Befehls zum Einstellen des Spannungspegels übertragen worden ist. Die VAR-Variable in Zeile 78 wird verwendet, um den digitalen DAC Eingangswert zu speichern, der dem gewünschten Spannungspegel entspricht. Der Hauptpufferzähler wird verwendet, um die Anzahl der Elemente im Hauptpuffer zu verfolgen. Das Zeichenarray str in Zeile 80 kommt zur Anwendung, um den String-Wert der Spannung zu speichern, der aus dem Spannungseinstellungsbefehl extrahiert wurde. Die Zeichenvariable ch in Zeile 81 wird verwendet, um am Ende des Spannungsbefehls '\n' anzuhängen.

Der Befehlspuffer in Zeile 84 dient dazu, einen über die UART Schnittstelle empfangenen Befehl zu speichern. Die Variable activeChannel in Zeile 86 wird verwendet, um den Wert des aktiven Kanals zu speichern, und die Variable newCommandReceived in Zeile 88, um den Status des empfangenen neuen Befehls zu speichern. Die Variablen in Zeile 91 und 92 werden verwendet, um den aktiven Kanal zu verfolgen, damit ausgeschlossen werden kann, dass derselbe Kanal erneut aktiviert wird.

Die Zeiger in den Zeilen 95-99 dienen dazu, auf die Adresse des dekodierten Befehls zu verweisen, wenn der Vergleich mit den vordefinierten Befehlen eine Übereinstimmung ergeben hat.

```
70 /* USER CODE BEGIN 0 */
71 #define RxBuf_SIZE   64                    //!< Größe des Rx-Puffers, der zum Empfang von Daten über UART verwendet wird.
72 #define MainBuf_SIZE 64                    //!< Größe des Hauptpuffers, der zum Speichern der über UART empfangenen Daten verwendet wird.
73 #define commandBuf_SIZE 64
74
75 uint8_t RxBuf[RxBuf_SIZE];                 //!< Rx-Puffer für den Empfang von Daten über UART.
76 uint8_t MainBuf[MainBuf_SIZE];             //!< Hauptpuffer, der zum Speichern der über UART empfangenen Daten verwendet wird.
77 //float value  = 0.0;                      //!< Float-Variable zum Speichern des Output wertes des DAC-Kanals.
78 float value1 = 0.0;                        //!< Float-Variable zum Speichern des Output wertes des DAC-Kanals.
79 float value2 = 0.0;                        //!< Float-Variable zum Speichern des Output wertes des DAC-Kanals.
80
81 uint32_t VAR = 0;                          //!< VAR zur Speicherung des jeweiligen Digitalwertes für den Output des DAC.
82 static uint16_t MainBufCounter ;           //!< Statische Variable, die den aktuellen Eintrag des Main-Buffers festhält
83 char str[20]={'0'};
84 char ch = '\n' ;
85
86 uint8_t commandBuf[commandBuf_SIZE];                    //Dieser Puffer wird verwendet, um den empfangenen Befehl zu speichern.
87 uint8_t activeChannel = 0;                              //Dieses Flag wird verwendet, um den aktiven Kanalwert zu speichern
88 bool newCommandReceived = false;                        //Dieses Flag wird verwendet, um zu prüfen, ob der neue Befehl empfangen wurde.
89
90 bool isChannel1Active = false;                          //Diese Flaggen zeigen den Status des Kanals an
91 bool isChannel2Active = false;
92
93 char *retCommand1;                                      //Diese werden zur Überprüfung des empfangenen Befehls verwendet.
94 char *retCommand2;
95 char *retCommand3;
96 char *retCommand4;
97 char *retCommand5;
```

Der folgende Code ist die UART-Interrupt-Handler-Funktion, die jedes Mal ausgeführt wird, wenn Daten über die UART Schnittstelle empfangen werden. In dieser Funktion werden die Daten im Rx-Puffer empfangen und danach in den Hauptpuffer kopiert. Jedes Element wird auf das Zeichen '\n' geprüft. Wenn dieses Zeichen empfangen worden ist, bedeutet dies, dass ein Befehl vollständig empfangen wurde. Dann wird die Variable newCommandReceived auf true gesetzt.

Es wird noch überprüft, ob der Hauptpuffer voll ist, aber noch Daten im Rx-Puffer vorhanden sind. In diesem Szenario soll der Hauptpuffer als Ringpuffer fungieren, und die Anfangswerte des Hauptpuffers werden entsprechend der Anzahl der überzähligen Elemente überschrieben.

Nach erfolgreichem Empfang der Daten wird der Empfangsinterrupts wieder aktiviert und der DMA Interrupt, der bei Übertragung der halben Datenmenge ausgelöst, deaktiviert.

```c
void HAL_UARTEx_RxEventCallback(UART_HandleTypeDef *huart, uint16_t Size)
{
    //Variable, die zum Kopieren von Daten aus dem RX-Puffer in den Hauptpuffer verwendet wird.
    uint8_t Counter = 0 ;

    //Prüfen, ob der Interrupt vom USART2 kommt. Da wir derzeit USART 2 verwenden
    if(huart -> Instance == USART2 )
    {
        //Schleife zum Kopieren der Daten vom Rx-Puffer in den Hauptpuffer.
        while(Counter <= Size)
        {
            //Kopieren von Daten aus dem Rx-Puffer in den Hauptpuffer.
            MainBuf[MainBufCounter] = RxBuf[Counter++] ;

            //Prüfen Sie auf das Zeichen \n, das angibt, dass der vollständige Befehl empfangen worden ist.
            if(MainBuf[MainBufCounter] == '\n')
            {
                for(int i = 0 ; i < (MainBufCounter); i++)  //den empfangenen Befehl in den Befehlspuffer einfügen.
                {
                    commandBuf[i] = MainBuf[i];
                }
                newCommandReceived = true;  //Flagge, die anzeigt, dass der neue Befehl empfangen wurde.
            }

            //Prüfen Sie, dass am Ende jeder Datenkopie keine Null hinzugefügt wird.
            if(Counter - 1 != Size)
            {
                //Inkrementieren des Hauptpufferzählers
                MainBufCounter++;
            }

            //Prüfen, ob der Hauptpuffer voll mit Daten ist. In diesem Fall werden die Daten von Anfang an ersetzt.
            if(MainBufCounter > 64 )
            {
                //Zurücksetzen des Hauptpufferzählers
                MainBufCounter = 0;
            }
        }

        //Diese Funktion wird verwendet, um Daten von UART mit DMA zu empfangen, bis der Datenpuffer voll ist oder die IDLE Line erkannt wird.
        HAL_UARTEx_ReceiveToIdle_DMA(&huart2, RxBuf, RxBuf_SIZE);
        //Deaktivieren Sie den Interrupt für die halb übertragenen Daten.
        __HAL_DMA_DISABLE_IT(&hdma_usart2_rx, DMA_IT_HT);
    }
}
```

Der folgende Code ist die benutzerdefinierte Funktion, die für die Übertragung von Daten über den UART verwendet wird. Die zu versendenden Daten werden mit Hilfe eines String Zeigers übergeben. Die Funktion berechnet die Länge des Strings und überträgt die Daten dann über die UART Schnittstelle.

```c
void Uprintf(char *str)
{
    //Funktion zur Übertragung von Daten auf UART.
    HAL_UART_Transmit(&huart2 ,(uint8_t*) str, strlen(str),1000);
}
```

Der Befehl zur Einstellung des Spannungspegels wird in Form eines Strings über die UART Schnittstelle empfangen, sodass er zunächst in einen Double-Wert umgewandelt werden muss, um dann den entsprechenden digitalen Wert zu berechnen, der auf den DAC angewendet werden soll Hierfür wird die Funktion get_double verwendet, welche den String in einen Double Wert umwandelt. In dieser Funktion wird der Zeiger auf den String solange inkrementiert bis eine Ziffer mit Hilfe der Funktion isdigit() identifiziert wird.

```
159 double get_double(const unsigned char *str)
160 {
161     /* Erstes Überspringen nicht-ziffriger Zeichen */
162         /* Sonderfall zur Behandlung negativer Zahlen */
163     while (*str && !(isdigit(*str) || ((*str == '-' || *str == '+') && isdigit(*(str + 1)))))
164         str++;
165 
166     /* Das Parsen in ein Double */
167     return strtod((const char *)str, NULL);
168 }
```

Di Diese Funktion wird verwendet, um der Zahl eine Linksverschiebung zu geben. Diese Funktion benötigt zwei Argumente: Zahl und Anzahl. Basierend auf der Anzahl der Zählung wird die Zahl verschoben. Diese Funktion wird verwendet, um eine Anzahl von Nullen am Ende der Zehn anzuhängen. Wenn die Zählung 1 und die Zahl 10 ist, gibt diese Funktion 10 zurück, wenn die Zählung 2 und die Zahl 10 ist, gibt diese Funktion 100 zurück und so weiter

```
169 //Diese Funktion wird bei der Umwandlung von Float in String verwendet.
170 int n_tu(int number, int count)
171 {
172     int result = 1;
173     while(count-- > 0)
174         result *= number;
175 
176     return result;
177 }
```

Diese Funktion wird verwendet, um den Float-Wert der Spannung in eine Zeichenkette umzuwandeln. Diese Funktion benötigt zwei Argumente: eine Fließkommazahl und ein Zeichenarray zum Speichern des Ergebnisses. Diese Funktion nimmt eine Fließkommazahl und gibt nach der Verarbeitung die resultierende Zeichenkette an das Zeichenarray zurück. Die Implementierung dieser Funktion ist einfach: Zunächst wird geprüft, ob die Float-Zahl negativ ist. Danach prüft sie

die Länge des Zehntelteils und danach die Länge des Dezimalteils der Float-Zahl, basierend auf diesen Ergebnissen wird die Zeichenkette an das Zeichen-Array angehängt.

```
178 //Diese Funktion wird verwendet, um Float-Werte in Strings umzuwandeln.
179 void float_to_string(float f, char r[])
180 {
181     long long int length, length2, i, number, position, sign;
182     float number2;
183 
184     sign = -1;    // -1 == positive Zahl
185     if (f < 0)
186     {
187         sign = '-';
188         f *= -1;
189     }
190 
191     number2 = f;
192     number = f;
193     length = 0;  // Größe des Dezimalteils
194     length2 = 0; // Größe des Zehntels
195 
196     /* Berechnung des zehnten Teils der Länge2 */
197     while( (number2 - (float)number) != 0.0 && !((number2 - (float)number) < 0.0) )
198     {
199          number2 = f * (n_tu(10.0, length2 + 1));
200          number = number2;
201 
202          length2++;
203     }

205     /* Berechnung der Länge des Dezimalteils */
206     for (length = (f > 1) ? 0 : 1; f > 1; length++)
207         f /= 10;
208 
209     position = length;
210     length = length + 1 + length2;
211     number = number2;
212     if (sign == '-')
213     {
214         length++;
215         position++;
216     }
217 
218     for (i = length; i >= 0 ; i--)
219     {
220         if (i == (length))
221             r[i] = '\0';
222         else if(i == (position))
223             r[i] = '.';
224         else if(sign == '-' && i == 0)
225             r[i] = '-';
226         else
227         {
228             r[i] = (number % 10) + '0';
229             number /=10;
230         }
231     }
232 }
```

Der unten stehende Code entspricht der Hauptfunktion, mit der die Ausführung der Firmware des Mikrocontrollers beginnt. Der Code von Zeile 250 bis 263 entspricht dem Code, der von STM32CubeMx generiert worden ist. Danach werden in Zeile 257 die URAT-Empfangsinterrupts aktiviert und anschließend die halben Empfangsinterrupts deaktiviert. In Zeile 279 wird der erste Kanal als Standardkanal gewählt, falls kein Kanal aktiviert und kein Spannungswert angelegt worden ist.

```
243 int main(void)
244 {
245
246   HAL_Init();
247
248   SystemClock_Config();
249
250   /* Initialisierung aller konfigurierten Peripheriegeräte */
251   MX_GPIO_Init();
252   MX_DMA_Init();
253   MX_DAC1_Init();
254   MX_USART2_UART_Init();
255   /* USER CODE BEGIN 2 */
256   //Diese Funktion wird verwendet, um Daten von UART mit DMA zu empfangen, bis der Datenpuffer voll ist oder die idle line erkannt wird.
257   HAL_UARTEx_ReceiveToIdle_DMA(&huart2, RxBuf, RxBuf_SIZE);
258   //Deaktivieren des Interrupts "Halbe Daten übertragen"..
259   __HAL_DMA_DISABLE_IT(&hdma_usart2_rx, DMA_IT_HT);
260   //Aktivieren Sie den Output von DAC-Kanal 1;
261  HAL_DAC_Start(&hdac1 , DAC_CHANNEL_1);
262  isChannel1Active = true;
263  activeChannel = 1;
```

In unten dargestellten Code wird ein neu empfangener Befehl mit den vordefinierten Befehlen verglichen und bei Übereinstimmung auf der Grundlage des empfangenen Befehls die entsprechende Funktion ausgeführt. In Zeile 296 wird der empfangene Befehl mit dem Befehl "INSTrument:SELect?" verglichen und das Ergebnis in der Variablen reCommand1 gespeichert. In der Zeile 296 wird der empfangene Befehl mit dem Befehl "SOURce:VOLTage:LEVel?" verglichen und das Ergebnis in der Variablen reCommand2 gespeichert. In der Zeile 296 wird der empfangene Befehl mit dem Befehl "INSTrument:SELect:OUTPut1" verglichen und das Ergebnis in der Variablen reCommand3 gespeichert. In der Zeile 296 wird der empfangene Befehl mit dem Befehl "INSTrument:SELect:OUTPut2 "verglichen und das Ergebnis in der Variablen reCommand4 gespeichert. In der Zeile 296 wird der empfangene Befehl mit dem Befehl "SOURce:VOLTage:LEVel:" verglichen und das Ergebnis in der Variablen reCommand5 gespeichert.

```
269   while (1)
270   {
271
272    if(newCommandReceived == true)                  //Prüfen, ob der neue Befehl empfangen wurde.
273        {
274        retCommand1 = strstr((char *)commandBuf, "INSTrument:SELect?");       //Prüfen, ob der "INSTrument:SELect?" Befehl empfangen wurde.
275
276        retCommand2 = strstr((char *)commandBuf, "SOURce:VOLTage:LEVel?");      //Prüfen, ob der "SOURce:VOLTage:LEVel?" Befehl empfangen wird.
277
278        retCommand3 = strstr((char *)commandBuf, "INSTrument:SELect:OUTPut1"); //Prüfen, ob der Befehl "INSTrument:SELect:OUTPut1" empfangen wird.
279
280        retCommand4 = strstr((char *)commandBuf, "INSTrument:SELect:OUTPut2");  //Prüfen, ob der Befehl "INSTrument:SELect:OUTPut2" empfangen wird.
281
282        retCommand5 = strstr((char *)commandBuf, "SOURce:VOLTage:LEVel:");    //Prüfen, ob der "SOURce:VOLTage:LEVel:" Befehl empfangen wird.
```

Der folgende Code wird ausgeführt, wenn der Befehl zur Geräteauswahl empfangen wurde. In diesem Code wird zunächst der aktive Kanal überprüft: Wenn Kanal 1 aktiv ist, wird "1\n" über den UART übertragen, und wenn Kanal 2 aktiv ist, wird "2'\n" über den UART ausgegeben. Nach der erfolgreichen Operation werden der Hauptpuffer und die Befehlspuffer zurückgesetzt.

```
300           if(retCommand1)
301           {
302              if(activeChannel == 1u)     //Prüfung auf den aktiven Kanal 1
303              {
304                Uprintf("1\n");   //Senden Sie den aktiven Kanal-String zurück
305              }
306              else
307              if(activeChannel == 2u)     //Prüfung auf den aktiven Kanal 2
308              {
309                Uprintf("2\n");   //Senden Sie den aktiven Kanal-String zurück
310              }
311              //Rücksetzen des commandBuf und des MainBuf sowie des Flags für den neuen empfangenen Befehl
312              newCommandReceived = false;
313              memset(commandBuf , 0 ,commandBuf_SIZE*(sizeof(commandBuf[0])));
314              MainBufCounter = 0 ;
315              memset(MainBuf , 0 ,MainBuf_SIZE*(sizeof(MainBuf[0])));
316
317           }
```

Der folgende Code wird ausgeführt, wenn der Befehl "SOURce:VOLTage:LEVel?" empfangen wurde. In diesem Code wird zuerst der aktuelle Spannungswert des aktivierten DAC Kanals in eine Zeichenkette umgewandelt und dann über die UART Schnittstelle versendet. Nach der erfolgreichen Ausführung der obigen Operation werden der Hauptpuffer und der Befehlspuffer zurückgesetzt.

```
302           if(retCommand2)
303           {
304             if(activeChannel == 1u)
305             {
306                 float_to_string(value1,str);              //den Spannungswert in einen Stringwert umwandeln.
307                 Uprintf(strncat((char *)str,&ch,1));   //Senden des Spannungswertes über UART
308             }
309             else
310             {
311                 if(activeChannel == 2u)
312                 {
313                     float_to_string(value2,str);              //den Spannungswert in einen Stringwert umwandeln.
314                     Uprintf(strncat((char *)str,&ch,2));   //Senden des Spannungswertes über UART
315                 }
316             }
317             newCommandReceived = false;           //Rücksetzen des commandBuf und des MainBuf sowie des Flags für den neuen empfangenen Befehl
318             memset(commandBuf , 0 ,commandBuf_SIZE*(sizeof(commandBuf[0])));
319             MainBufCounter = 0 ;
320             memset(MainBuf , 0 ,MainBuf_SIZE*(sizeof(MainBuf[0])));
321           }
```

Der folgende Code wird ausgeführt, wenn der Befehl "INSTrument:SELect:OUTPut1" empfangen wurde. In diesem Code wird zuerst der Kanal 1 des DAC aktiviert und die Variable, die den Wert des aktiven Kanals auf 1 geändert. Nach der erfolgreichen Ausführung der obigen Operation werden der Hauptpuffer und der Befehlspuffer zurückgesetzt.

```
322          if(retCommand3)
323          {
324              if(isChannel1Active == false)
325              {
326                //Aktivieren Sie den Output von DAC-Kanal 1;
327                HAL_DAC_Start(&hdac1 , DAC_CHANNEL_1);
328                isChannel1Active = true;
329                //Digitale Wertberechnung zum Schreiben auf den DAC.
330                VAR = value1 * (0xfff + 1) / 3.3;
331                //Diese Funktion schreibt den berechneten digitalen Wert in den DAC-Kanal..
332                HAL_DAC_SetValue(&hdac1 , DAC_CHANNEL_1 , DAC_ALIGN_12B_R , VAR);
333
334              }
335
336              activeChannel = 1u;
337              //Rücksetzen des commandBuf und des MainBuf und des empfangenen neuen Befehls
338            newCommandReceived = false;
339            memset(commandBuf , 0 ,commandBuf_SIZE*(sizeof(commandBuf[0])));
340            MainBufCounter = 0 ;
341            memset(MainBuf , 0 ,MainBuf_SIZE*(sizeof(MainBuf[0])));
342          }
343
```

Der folgende Code wird ausgeführt, wenn der Befehl "INSTrument:SELect:OUTPut2" empfangen wurde. In diesem Code wird zuerst der Kanal 2 des DAC aktiviert und der Wert des aktiven Kanals auf 2 geändert. Nach der erfolgreichen Ausführung der obigen Operation werden der Hauptpuffer und der Befehlspuffer zurückgesetzt.

```
344          if(retCommand4)
345          {
346            if(isChannel2Active == false)
347            {
348                //Aktivieren Sie den Output von DAC-Kanal 2;
349                HAL_DAC_Start(&hdac1 , DAC_CHANNEL_2);
350                isChannel2Active = true;
351                //Digitale Wertberechnung zum Schreiben auf den DAC.
352                VAR = value2 * (0xfff + 1) / 3.3;
353                //Diese Funktion schreibt den berechneten digitalen Wert in den DAC-Kanal.
354                HAL_DAC_SetValue(&hdac1 , DAC_CHANNEL_2 , DAC_ALIGN_12B_R , VAR);
355            }
356
357            activeChannel = 2u;
358            //Rücksetzen des commandBuf und des MainBuf sowie des Flags für den neuen empfangenen Befehl
359            newCommandReceived = false;
360            memset(commandBuf , 0 ,commandBuf_SIZE*(sizeof(commandBuf[0])));
361            MainBufCounter = 0 ;
362            memset(MainBuf , 0 ,MainBuf_SIZE*(sizeof(MainBuf[0])));
363          }
```

Der folgende gemeinsame Code wird ausgeführt, wenn der Befehl "SOURce:VOLTage:LEVel:" empfangen wurde. In diesem Code wird zuerst der in der Zeichenkette empfangene Spannungswert in einen double Wert umgewandelt und dann der entsprechende digitale Wert berechnet, der auf den DAC-Ausgang angewendet werden soll. Danach wird der aktiv DAC-Kanal festgestellt und dann die Spannung an diesen Kanal gelegt. Nach erfolgreicher Ausführung der obigen Operation werden der Hauptpuffer und der Befehlspuffer zurückgesetzt.

```
364             if(retCommand5)
365
366             {
367                 if(activeChannel == 1u)
368                 {
369                     //Extrahieren Sie den Spannungswert aus dem Befehl.
370                     value1 = get_double(commandBuf);
371                     //Digitale Wertberechnung zum Schreiben auf den DAC.
372                     VAR = value1 * (0xfff + 1) / 3.3;
373                     //Diese Funktion schreibt den berechneten digitalen Wert in den DAC-Kanal..
374                     HAL_DAC_SetValue(&hdac1 , DAC_CHANNEL_1 , DAC_ALIGN_12B_R , VAR);
375                 }
376                 else
377                 if(activeChannel == 2u)
378                 {
379                     //Extrahieren Sie den Spannungswert aus dem Befehl.
380                     value2 = get_double(commandBuf);
381                     //Digitale Wertberechnung zum Schreiben auf den DAC.
382                     VAR = value2 * (0xfff + 1) / 3.3;
383                     //Diese Funktion schreibt den berechneten digitalen Wert in den DAC-Kanal.
384                     HAL_DAC_SetValue(&hdac1 , DAC_CHANNEL_2 , DAC_ALIGN_12B_R , VAR);
385                 }
386
387                 //Rücksetzen des commandBuf und des MainBuf sowie des Flags für den neuen empfangenen Befehl
388                 newCommandReceived = false;
389                 memset(commandBuf , 0 ,commandBuf_SIZE*(sizeof(commandBuf[0])));
390                 MainBufCounter = 0 ;
391                 memset(MainBuf , 0 ,MainBuf_SIZE*(sizeof(MainBuf[0])));
392             }
393         }
394     }
```

7 PyQt5 – Überblick:

Die Qt-Anwendung, die für dieses Projekt verwendet wird, wurde mit PyQt5 in der Programmiersprache Python unter Ubuntu entwickelt. PyQt ist ein Toolkit für GUI-Widgets. Es ist eine der beliebtesten und leistungsfähigsten plattformübergreifenden GUI-Bibliotheken. Die PyQt API besteht aus einer großen Anzahl von Modulen, die eine große Anzahl von Klassen und Funktionen enthalten. Die neueste Version von PyQt kann von der Website heruntergeladen werden.

PyQt5 hat eine große Anzahl von Widgets. Die Details der Widgets, die in dieser Qt-Anwendung verwendet werden, sind unten aufgeführt:

Nein.	Widget-Name	Widget-Details.
1	QPushButton	Es handelt sich um eine einfache Schaltfläche, die beim Drücken eine bestimmte Funktion ausführt.
2	Qlabel	Dieses Widget wird zur Anzeige von Text oder Bildern verwendet.
3	Qslider	Dies ist ein klassisches Widget zur Einstellung eines begrenzten Wertes. Der Benutzer kann den Schieberegler horizontal oder vertikal bewegen und der Schieberegler übersetzt diesen Wert in einen ganzzahligen Wert innerhalb des zulässigen Bereichs.
4	QText-Browser	Dieses Widget wird verwendet, um die Textnachrichten als Benutzeranweisungen anzuhängen. In dieser Anwendung werden Nachrichten als Benutzeranweisungen generiert.
5	QDialog	Dieses Widget ist das oberste Fenster, das dazu dient, Eingaben des Benutzer zu sammeln. In dieser Anwendung sind alle Widgets in einem Fenster verfügbar.

Tabelle 3 Widgets-Funktionen

8 Qt Designer:

Qt Designer ist ein Qt-Werkzeug zur Erstellung und Gestaltung einem grafischen Benutzeroberflächen in Qt-Anwendungen. Mit diesem Werkzeug können grafischen Benutzeroberflächen auf der Grundlage von "What you see is what you get" entworfen werden. Zuerst wird die Anwendungs-GUI mit dem Qt Designer entwickelt und dann wird die entwickelte GUI mit den PyQt5-APIs zu einer gemeinsamen Anwendung integriert. Mit dem Qt-Designer können die Widgets per Drag & Drop aus der Widget-Box gezogen und Daraufhin kann ihre Größe sowie gegebenenfalls vorhandene Texte manuell eingestellt werden. Die für die Anwendung generierte GUI ist in Abbildung 31 zu sehen.

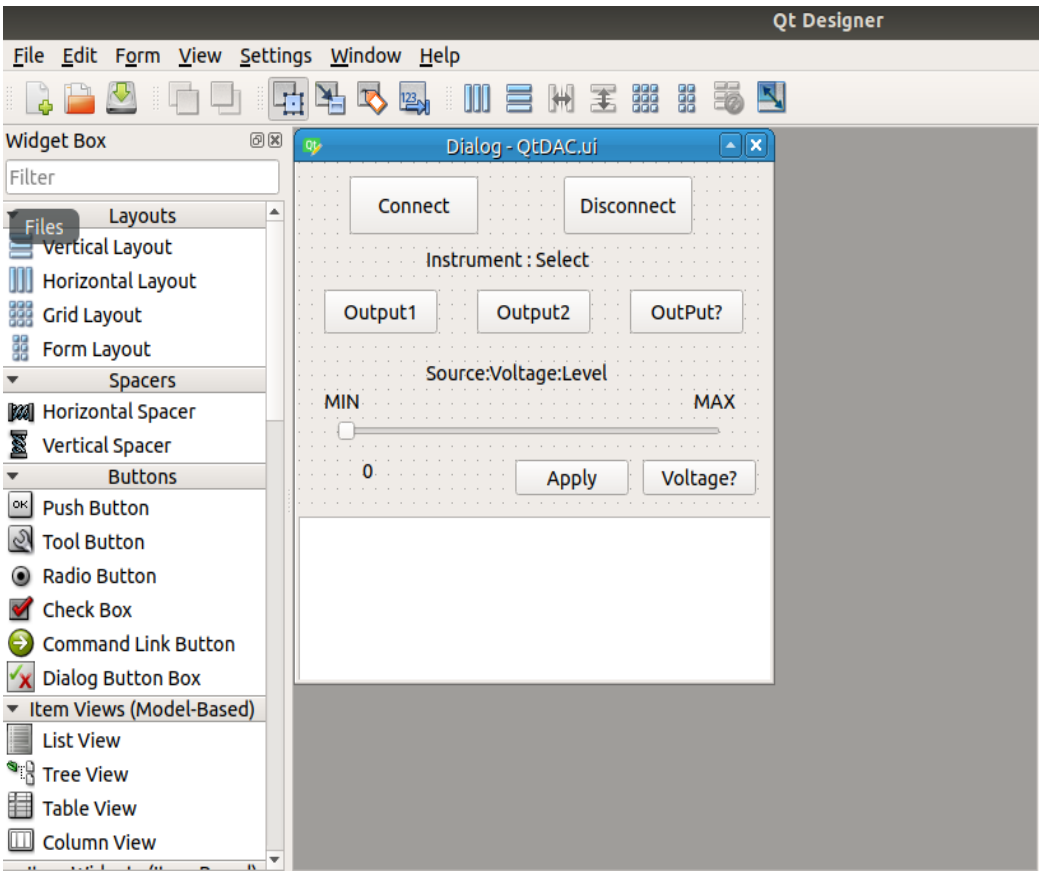


Abbildung 30: Qt-Designer

9 PyQt5 – Installationsbefehle für Ubuntu

Der Installationsprozess für PyQt5 ist unter Ubuntu sehr einfach.

9.1 Schritt 1

Das Terminal wird in Ubuntu geöffnet und der Befehl ausgeführt:

sudo apt-get install python3-pyqt5

9.2 Schritt 2

Nach Beendigung des ersten Befehls wird der zweite Befehl eingegeben:

sudo apt-get install qtcreator pyqt5-dev-tools

9.3 Schritt 3

Nach Beendigung des zweiten Befehls wird der dritte Befehl eingegeben:

sudo apt-get install qttools5-dev-tools

10 Qt-Application GUI und die funktionalen Details

Die grafische Benutzeroberfläche und die Funktionen der Qt-Anwendung werden im Folgenden beschrieben:

Wenn die Anwendung gestartet wurde, erscheint zunächst der folgende Dialog:

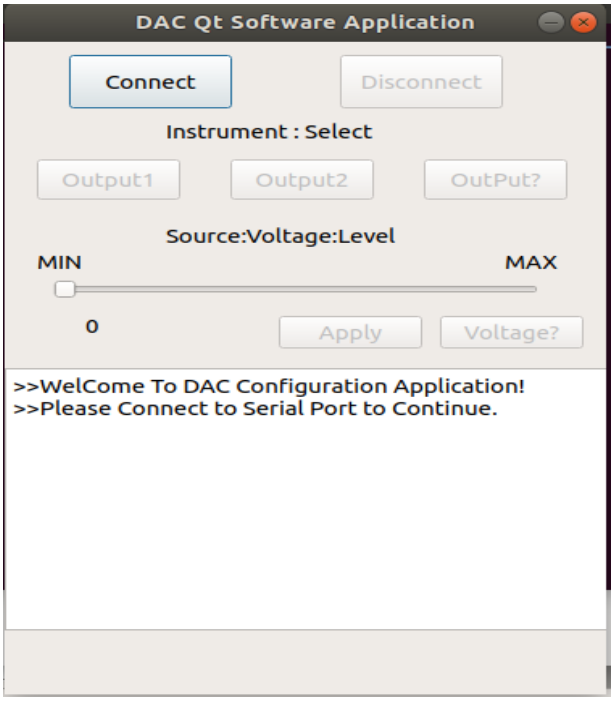


Abbildung 31: Qt DAC-Anwendungs-GUI

In diesem Dialogfeld werden Meldungen im Textbrowser ausgegeben. Als erstes wird die Aufforderung ausgegeben, eine Verbindung mit dem seriellen Anschluss herzustellen und fortzufahren. Sobald auf die Schaltfläche Verbinden geklickt wurde, erscheint bei erfolgreicher Verbindung zum Microcontroller-Board die folgende Ausgabe.

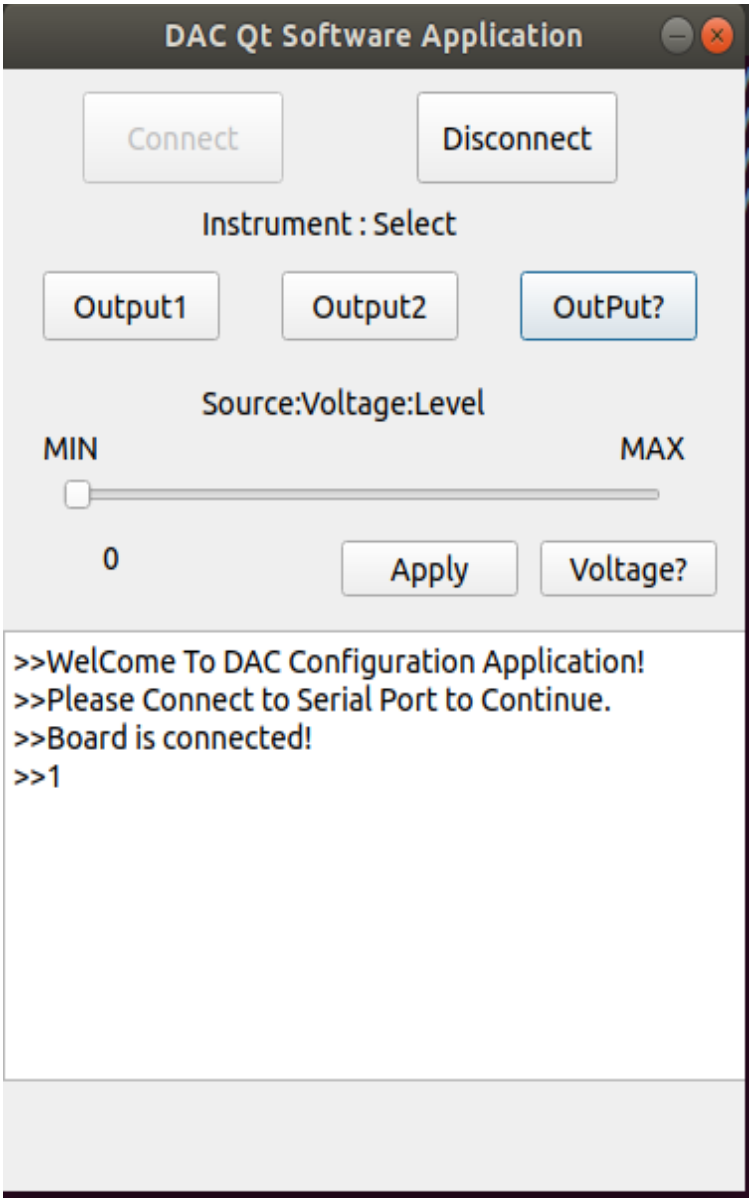


Abbildung 32: Qt-Anwendung nach Betätigung der Schaltfläche Verbinden

Danach stehen verschiedene Optionen zur Verfügung. Wenn der OutPut?-Button gedrückt wird, sendet die Applikation einen Befehl an den Mikrocontroller, um zu fragen, welcher Kanal gerade aktiv ist. Wird beispielsweise zuerst die OutPut2-Taste gedrückt, die den Befehl an den Mikrocontroller sendet, Kanal 2 zu aktivieren, und dann die OutPut?-Taste, antwortet der Mikrocontroller mit 2, wie in der Abbildung unten gezeigt:

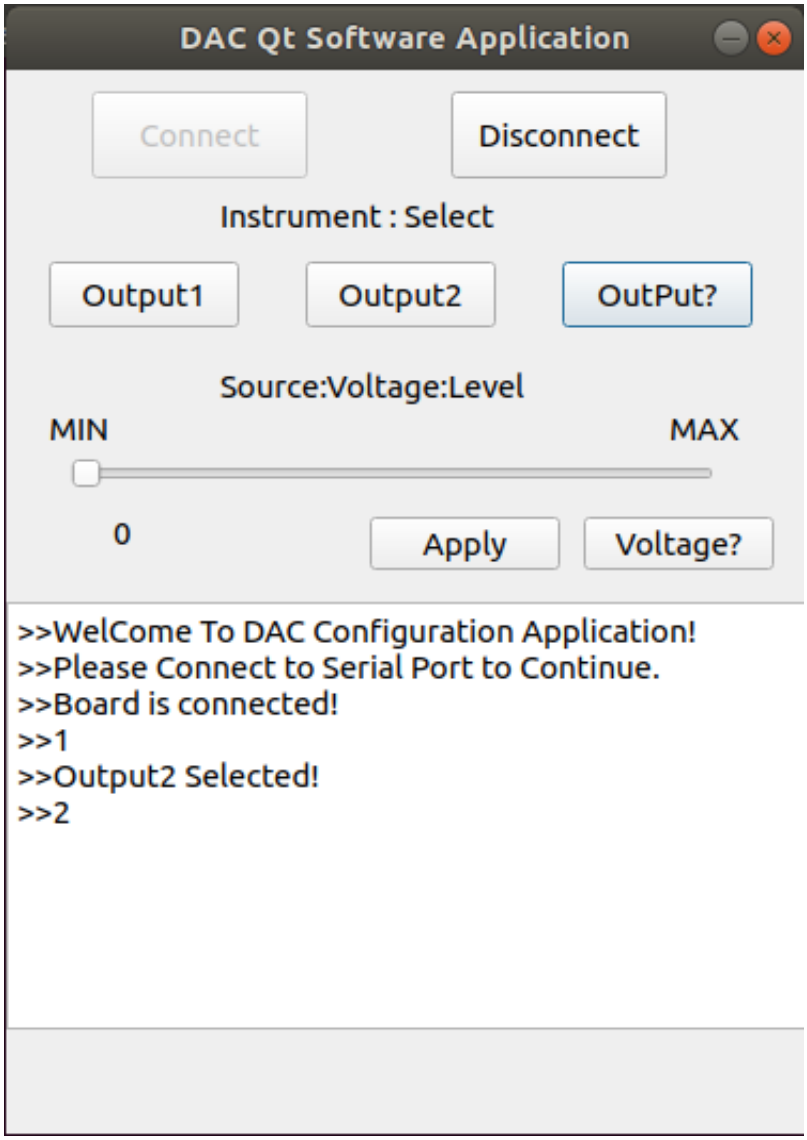


Abbildung 33: Qt-Anwendungskanal-Konfiguration

Das gleiche Szenario wird im Folgenden für Kanal 1 beschrieben:

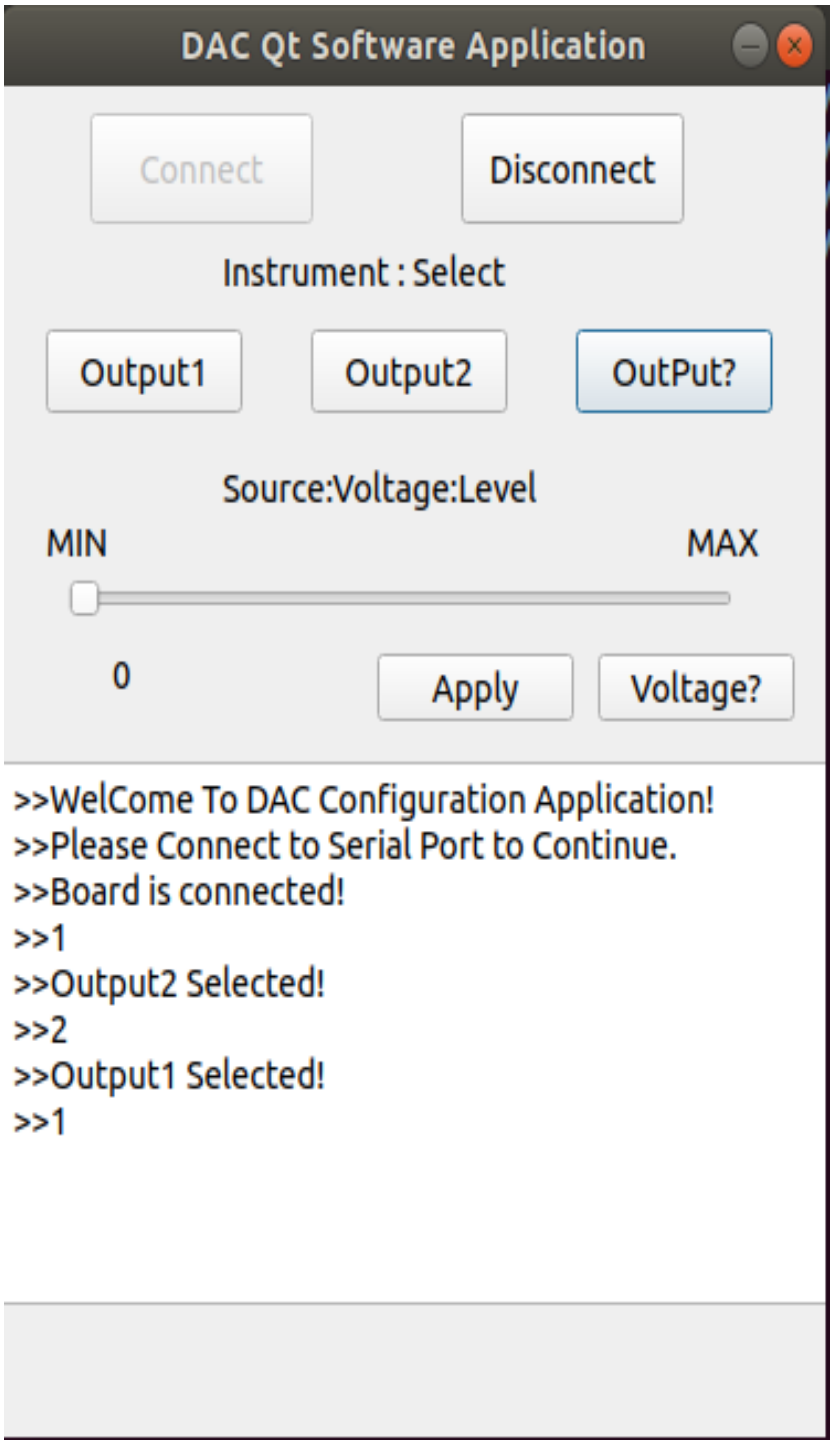


Abbildung 34: Qt-Anwendung – Funktion ausgewählter Kanal

Mit dem Schieberegler wird der Spannungswert eingestellt. Wenn der Schieberegler bewegt wird, ändert sich der Spannungswert, und beim Betätigen der Schaltfläche Apply wird der ausgewählte Spannungswert an den aktiven DAC-Kanal des Mikrocontrollers angelegt. Das Szenario ist unten dargestellt:

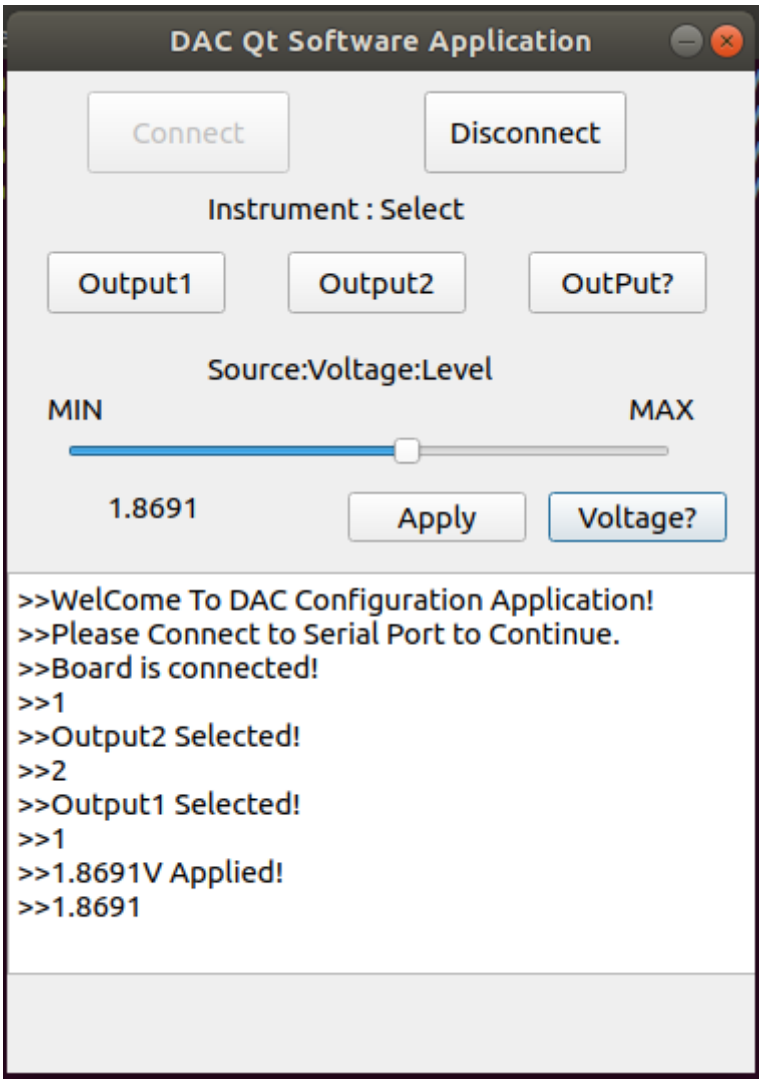


Abbildung 35: Qt-Anwendung – genutzte Spannung

Wenn die Taste Voltage? gedrückt wurde, wird der Befehl an den Mikrocontroller gesendet, um den Spannungswert aktuell aktivierten Kanals zu erfragen. In der obigen Abbildung antwortete die MCU beim Drücken der Voltage?-Taste mit dem Spannungswert, der im Textbrowser angezeigt wird.

Nachstehend finden sich Details zu den Schaltflächen:

Nein.	Schaltfläche	Befehl	Funktion
1	Verbinden	-	Wenn diese Taste gedrückt wird, wird die serielle Schnittstelle geöffnet und die Verbindungstaste deaktiviert, während die übrigen Tasten aktiviert werden.
2	DisConnect	-	Wenn diese Taste gedrückt wird, wird die serielle Schnittstelle geschlossen und die Verbindungstaste aktiviert, während die übrigen Tasten deaktiviert werden.
3	Ausgabe1	INSTrument:SELect:OUTPut1	Wenn diese Taste gedrückt wird, wird der Kanal 1 des DACs aktiviert.
4	Ausgabe2	INSTrument:SELect:OUTPut2	Wenn diese Taste gedrückt wird, wird der Kanal 1 des DACs aktiviert.
5	OutPut?	INSTrument:SELect?	Wenn diese Taste gedrückt wird, gibt die MCU den aktiven Kanalwert zurück und der Wert wird im Textbrowser ausgedruckt.
6	Bewerbung	SOURce:VOLTage:LEVel:	Wenn diese Taste gedrückt wird, wird der Wert des Schiebereglers auf den aktiven DAC-Kanal angewendet.
7	Spannung?	SOURce:VOLTage:LEVel?	Wenn diese Taste gedrückt wird, gibt die MCU den Spannungswert des zuletzt aktiven Kanals zurück und der Spannungswert wird im Textbrowser ausgedruckt.

Tabelle 4: Qt Application Button Funktionen mit Befehlen

11 Qt-Anwendungs-Firmware-Design

Die Qt-Application ist eine GUI-Anwendung, mit der Befehle für die jeweiligen Funktionen an den Mikrocontroller gesendet werden. Es gibt verschiedene Schaltflächen für die unterschiedlichen Funktionen. Zunächst wird eine Meldung ausgegeben, dass eine Verbindung mit dem Controller hergestellt werden muss, um fortzufahren. Wenn die serielle Schnittstelle erfolgreich geöffnet wurde, werden die weiteren Optionen aktiviert. Jede Schaltfläche hat eine bestimmte Funktion und einen bestimmten Befehl, der an den Mikrocontroller gesendet wird In Abbildung 37 ist ein Flußdiagramm dargestellt, dass die Struktur der Qt Applikation erlöutert.. Die Funktion Funktionsweise jeder verwendeten Funktion wird im Folgenden detailliert beschrieben.

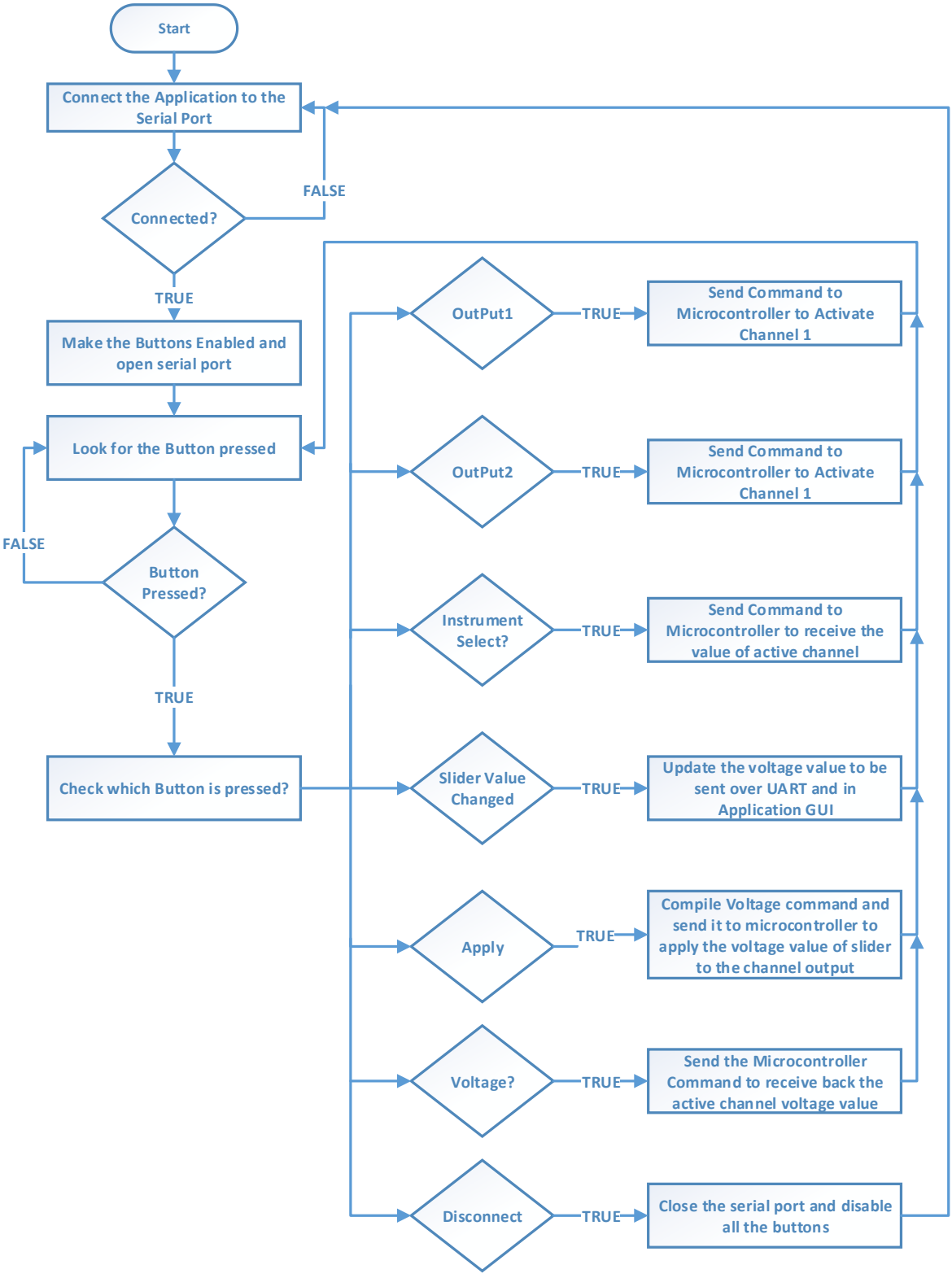


Abbildung 36: Flussdiagramm der Qt-Anwendungssoftware

12 Qt-Anwendungscode – Erläuterung

Der unten aufgeführte Code wird verwendet, um die pyserial-Funktionen und die qt-Widget-APIs in den Python-Code zu importieren. Danach wird eine Instanz der seriellen Klasse erstellt, und die verschiedenen Eigenschaften der seriellen Schnittstelle werden festgelegt.

```
#include the required file to use certain functions.
import sys
import serial
from PyQt5 import QtWidgets
from PyQt5.QtWidgets import QDialog, QApplication
from PyQt5.uic import loadUi

#Create the serial object to use serial functions and setup properties for the serial object
ser=serial.Serial()
ser.port = '/dev/ttyACM0'          #set serial port name
ser.baudrate = 115200              #set serial baud rate
ser.parity   = serial.PARITY_NONE  #set serial parity
ser.stopbits = serial.STOPBITS_ONE #set the stop bit for serial
ser.bytesize = serial.EIGHTBITS    #set the byte size for serial
ser.timeout = 1                    #non-block read
ser.xonxoff=False                  #disable software flow control
ser.rtscts= False                  #disable hardware (RTS/CTS) flow control
ser.dsrdtr= False                  #disable hardware (DSR/DTR) flow control
ser.writeTimeout = 2               #timeout for write
```

Im unten dargestellten Code wird eine Klasse erstellt und QDialog an diese Klasse übergeben. Diese Klasse enthält alle Schaltflächen, die Schiebereglerfunktion und die Textbrowserfunktion. Nach der Erstellung dieser Klasse wird die Konstruktorfunktion dieser Klasse definiert. In dieser Funktion wird zuerst die GUI-Datei geladen, die mit dem Qt Designer erstellt wurde, danach werden verschiedene Funktionen mit den verschiedenen Buttons der GUI-Datei verknüpft. In dieser Funktion werden die Limits und die Slider-Funktion definiert, verschiedene Buttons deaktiviert und die ersten Meldungen im Textbrowser ausgegeben.

```
#Create the DAC class and pass the QDialog to it
class DAC(QDialog):
   #Variable to store seleted channel number
   channelSelect = 1
   #Variable to store slider step size
   size = 0.0
   #Constructor function of the DAC Class
   def __init__(self):
       super(DAC,self).__init__()
       #load the ui file that we created using the qt designer
       loadUi("QtDAC.ui",self)
       #Connect the functions to the Buttons.
       self.ConnectButton.clicked.connect(self.Connectfunction)
       self.DisconnectButton.clicked.connect(self.Disconnectfunction)
       self.Channel1Button.clicked.connect(self.Channel1function)
       self.Channel2Button.clicked.connect(self.Channel2function)
       self.ApplyButton.clicked.connect(self.Applyfunction)
       self.OutPutQButton.clicked.connect(self.OutPutQfunction)
       self.VoltageQButton.clicked.connect(self.VoltageQfunction)
       #Set slider minimum value
       self.Slider.setMinimum(0)
       #Set slider maximum value
       self.Slider.setMaximum(4096)
       #Set slider step size
       self.Slider.setTickInterval(1)
       #Connect the function to the slider that when ever its value change this function is called
       self.Slider.valueChanged.connect(self.valuechange)
       #Disable the button before the connection to serial port.
       self.DisconnectButton.setEnabled(0)
       self.Channel1Button.setEnabled(0)
       self.Channel2Button.setEnabled(0)

      self.Slider.setEnabled(0)
      self.Slider.setEnabled(0)
      self.ApplyButton.setEnabled(0)
      self.OutPutQButton.setEnabled(0)
      self.VoltageQButton.setEnabled(0)
      #Append the following text in the text browser
      self.textBrowser.append(">>WelCome To DAC Configuration Application!")
      self.textBrowser.append(">>Please Connect to Serial Port to Continue.")
```

Der folgende Code entspricht der Funktion, die ausgeführt wird, wenn die OutPut?-Taste gedrückt wird. Diese Funktion sendet zuerst den Befehl an die MCU, liest dann den Wert des aktiven Kanals zurück und druckt ihn dann im Textbrowser aus.

```
#function that is called when the output? button is pressed.
def OutPutQfunction(self):
    #Command to send when this button is pressed.
    Command = "INSTrument:SELect?\n"
    #Send the command over serial.Encode function change the string to send as bytes e.g: \n
    ser.write(Command.encode())
    #Receive the data over serial and cast it to string
    receivedLine = str(ser.readline())
    #variable used to remove unecessary data form the received data
    N = len(receivedLine) - 3
    #Append the required data in the text browser
    self.textBrowser.append(">>" + receivedLine[2:N])
```

Der folgende Code entspricht der Funktion, die ausgeführt wird, wenn die Voltage?-Taste gedrückt wird. Sie sendet zunächst den Befehl an die MCU, liest den Spannungswert des aktiven Kanals zurück und gibt ihn dann im Textbrowser aus.

```
#function that is called when the Voltage? button is pressed.
def VoltageQfunction(self):
    #Command to send when this button is pressed.
    Command = "SOURce:VOLTage:LEVel?\n"
    #Send the command over serial.Encode function change the string to send as bytes e.g: \n
    ser.write(Command.encode())
    #Receive the data over serial and cast it to string
    receivedLine2 = str(ser.readline())
    #variable used to remove unecessary data form the received data
    N = len(receivedLine2) - 3
    #Append the required data in the text browser
    self.textBrowser.append(">>" + receivedLine2[2:N])
```

Der folgende Code implementiert die Funktion, die ausgeführt wird, wenn der Schiebereglerwert geändert wird. Sie ruft die Schiebereglerposition ab, übersetzt sie in einem Spannungswert und zeigt sie in der Beschriftung unter dem Schieberegler an.

```
#function that is called when value is changed in the slider.
def valuechange(self):
    #get the value of slider and store it in size variable
    self.size = self.Slider.value()
    #Normalize the slider step size to the voltage value
    self.size = self.size/4096
    self.size = self.size*3.3
    #round off the voltage value to 4 decimal places.
    self.size=round(self.size,4)
    #Show the voltage value in the lable text.
    self.label_5.setText(str(self.size))
```

Der folgende Code wird ausgeführt, wenn die Schaltfläche Verbinden gedrückt wird. Diese Funktion versucht zunächst, die serielle Schnittstelle zu öffnen. Wenn sie offen ist, aktiviert sie die übrigen Schaltflächen, deaktiviert die Schaltfläche Verbinden und gibt die Meldung im Textbrowser aus, dass das Board verbunden ist. Wenn die Schnittstelle nicht geöffnet werden kannt gibt sie die Meldung aus, dass die Schnittstelle nicht geöffnet ist.

```
#function that is called when the connect buttion is pressed.
def Connectfunction(self):
    #try to open the serial port
    try:
     ser.open()
    #if serial port is not opened then execute these instructions
    except:
     self.textBrowser.append(">>Could not open COM port!")
    #check if the serial port is open.
    if ser.isOpen():
     #disable the connect button.
     self.ConnectButton.setEnabled(0)
     #Enable remaining buttons.
     self.DisconnectButton.setEnabled(1)
     self.Channel1Button.setEnabled(1)
     self.Channel2Button.setEnabled(1)
     self.Slider.setEnabled(1)
     self.Slider.setEnabled(1)
     self.ApplyButton.setEnabled(1)
     self.OutPutQButton.setEnabled(1)
     self.VoltageQButton.setEnabled(1)
     #Append the text in text browser.
     self.textBrowser.append(">>Board is connected!")
```

Der nächste Code -Auszug wird verwendet, wenn die Taste zum Trennen der Verbindung gedrückt wird. Diese Funktion prüft zunächst, ob die serielle Schnittstelle offen ist, und versucht dann, sie zu schließen. Wenn sie geschlossen ist, werden alle Tasten deaktiviert und die Verbindungstaste aktiviert. Wenn die Schnittstelle nicht geschlossen werden konnte, wird eine Meldung ausgegeben, dass die Schnittstelle nicht geschlossen ist.

```
#function that is called when the disconnect buttion is pressed.
def Disconnectfunction(self):
    #check if the serial port is open.
    if ser.isOpen():
     #try to close the serial port
     try:
       ser.close()
     #if serial port is not closed then execute these instructions
     except:
       self.textBrowser.append(">>Could not Close COM port!")
    #Check if serial port is closed.
    if not(ser.isOpen()):
     #enable the Connect button
     self.ConnectButton.setEnabled(1)
     #disbale all the remaining buttons
     self.DisconnectButton.setEnabled(0)
     self.Channel1Button.setEnabled(0)
     self.Channel2Button.setEnabled(0)
     self.Slider.setEnabled(0)
     self.Slider.setEnabled(0)
     self.ApplyButton.setEnabled(0)
     self.OutPutQButton.setEnabled(0)
     self.VoltageQButton.setEnabled(0)
     #Append the text in text browser.
     self.textBrowser.append(">>Board is Disconnected!")
```

Der unten dargestellte Code entspricht der Funktion, die ausgeführt wird, wenn die Schaltfläche OutPut1 gedrückt wird. Diese Funktion sendet den Befehl an den Mikrocontroller, der den Kanal 1 aktiviert und gibt die Meldung im Textbrowser aus, dass Ausgang1 ausgewählt worden ist.

```
#This function is called when output1 button is pressed.
def Channel1function(self):
    #save the selected channel value.
    self.channelSelect = 1
    #Append the text in text browser.
    self.textBrowser.append(">>Output1 Selected!")
    #Command that is sent over serial when output1 button is pressed.
    Command = "INSTrument:SELect:OUTPut1\n"
    #Send the command over serial.
    ser.write(Command.encode())
```

Der nächste Code -Auszug wird immer dann ausgeführt, wenn die Schaltfläche OutPut2 gedrückt wird. Diese Funktion sendet ähnlich wie die vorherige Funktion den Befehl an den Mikrocontroller, den Kanal 2 zu aktivieren und gibt die Meldung im Textbrowser aus, dass Ausgang 2 ausgewählt worden ist.

```
#This function is called when output2 button is pressed.
def Channel2function(self):
    #save the selected channel value.
    self.channelSelect = 2
    #Append the text in text browser.
    self.textBrowser.append(">>Output2 Selected!")
    #Command that is sent over serial when output2 button is pressed.
    Command = "INSTrument:SELect:OUTPut2\n"
    #Send the command over serial.
    ser.write(Command.encode())
```

Die Funktion wird ausgeführt, wenn die Taste apply gedrückt wird. Diese Funktion ruft den Spannungswert vom Schieberegler ab und verknüpft ihn mit dem Befehl, der an den Mikrocontroller gesendet werden soll, und gibt die Meldung aus, dass der Spannungswert angelegt wurde.

```
#This function is called when output2 button is pressed.
def Applyfunction(self):
    #Make the command the to send over serial
    SourceVoltageLevel = "SOURce:VOLTage:LEVel:" + str(self.size) + "\n"
    #Write the command over serial
    ser.write(SourceVoltageLevel.encode())
    #Append the text in text browser.
    self.textBrowser.append(">>"+ str(self.size) + "V Applied!")
```

Zuletzt wird der Hauptanwendungscode gelistet. In diesem Code-Teil wird die Anwendung erstellt und dann ausgeführt. Hierzu wird ein Objekt der DAC-Klasse erstellt. Danach wird ein Widget erstellt, das dem Hauptfenster der Anwendung entspricht. Schließlich wird das Hauptfenster-Widget angezeigt und die Anwendung ausgeführt.

```
#Create the Qt application
app = QApplication(sys.argv)
#Create the main window based to the DAC class
mainwindow = DAC()
#create a QtWidget with QStackedWidget properties
widget = QtWidgets.QStackedWidget()
#Add the mainwindow to the widget
widget.addWidget(mainwindow)
#Set the Titlle of the widget
widget.setWindowTitle("DAC Qt Software Application")
#Set the fixed geometry of the widget
widget.setFixedSize(373,500)
#show the widget
widget.show()
#execute the application
app.exec_()
```

13 Fazit

Im Rahmen dieser Bachelorarbeit sollte ein Mikrocontroller mit einem integrierten DAC-Baustein so konfiguriert werden, dass er als Referenzspannungsquelle fungiert, die mit SCPI-Befehlen gesteuert werden kann. Das STM32L476 Nucleus Board mit zwei unabhängigen DAC-Kanälen wird in diesem Projekt verwendet. Die beiden DAC-Kanäle sind so konfiguriert, dass sie auf der Grundlage von Benutzereingaben einfache Gleichspannungen erzeugen.

Eine Qt-Anwendung wurde entwickelt, um mit dem Board zu kommunizieren und die Programmierung des Mikrocontrollers zu testen. Die Qt-Anwendung sendet einen Befehl an den Mikrocontroller. Der Mikrocontroller empfängt den Befehl und führt daraufhin die entsprechende Anweisung aus.

14 Referenzen

- https://www.st.com/resource/en/user_manual/um2563-stm32cubeide-installation-guide-stmicroelectronics.pdf
- https://www.st.com/en/development-tools/stm32cubeide.html
- https://www.st.com/en/development-tools/stm32cubemx.html
- https://www.digikey.com/en/products/detail/stmicroelectronics/stm32l476rgt6u/7313369
- https://www.st.com/en/evaluation-tools/nucleo-l476rg.html
- https://deepbluembedded.com/stm32-dac-tutorial-example-hal-code-analog-signal-genreation/
- https://deepbluembedded.com/stm32-usart-uart-tutorial/

Danksagung

Ich möchte mich bei mehreren Personen bedanken welche mich während meines Studiums unterstützten und begleiteten. Zunächst möchte ich mich bei Herrn Prof. Dr. Ing Michael Karagounis bedanken, welcher mir stets ein offenes Ohr für meine Fragen anbot und dabei mit seiner authentischen und zuversichtlichen Ausstrahlung meine zuweilen auftretenden Bedenken während des Studiums nahm. Darüber hinaus engagiert sich Herr Prof. Dr. Ing Michael Karagounis stets für die Studenten des Fachbereiches 3 der Fachhochschule Dortmund um Ihnen eine erstklassige Ausbildung zu ermöglichen.

Des Weiteren möchte ich mich bei Herrn M.A. Alexander Walsemann, welcher mir in allen Situationen stets seine helfende Hand anbot und mir immer Zuversicht zu sprach.

15 STM32-Board-Firmware

```
/* USER CODE BEGIN Überschrift */
/**
  ******************************************************************************
  * @Datei : main.c
  * @brief : Hauptteil des Programms
  ******************************************************************************
  * @Aufmerksamkeit
  *
  * <h2><center>&copy; Copyright (c) 2022 STMicroelectronics.
  * Alle Rechte vorbehalten.</center></h2>
  *
  * Diese Softwarekomponente wird von ST unter der BSD 3-Clause-Lizenz lizenziert.
  * Die Datei darf nur in Übereinstimmung mit der
  * Lizenz verwendet werden. Eine Kopie der Lizenz ist erhältlich unter:
  * opensource.org/licenses/BSD-3-Clause
  *
  ******************************************************************************
  */
/* USER CODE END Header */
/* Includes ------------------------------------------------------------------*/
#include "main.h"

/* Private includes ----------------------------------------------------------*/
/* USER CODE BEGIN Enthält */
#include "string.h"
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include <ctype.h>
#include <stdbool.h>
#include <ctype.h>

/* USER CODE END Includes */

/* Privat typedef -----------------------------------------------------------*/
/* BENUTZERCODE BEGINNT PTD */

/* BENUTZERCODE ENDE PTD */

/* Private Definition -------------------------------------------------------*/
/* BENUTZERCODE BEGINNT PD */
/* BENUTZERCODE ENDE PD */

/* Privates Makro -----------------------------------------------------------*/
/* USER CODE BEGIN PM */

/* BENUTZERCODE ENDE PM */

/* Private Variablen ---------------------------------------------------------*/
DAC_HandleTypeDef hdac1;

UART_HandleTypeDef huart2;
DMA_HandleTypeDef hdma_usart2_rx;
```

```
/* BENUTZERCODE BEGINNT PV */

/* BENUTZERCODE ENDE PV */

/* Private Funktionsprototypen -----------------------------------------------*/
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_DMA_Init(void);
static void MX_DAC1_Init(void);
static void MX_USART2_UART_Init(void);
/* BENUTZERCODE BEGINNT MIT PFP */

/* BENUTZERCODE ENDE PFP */

/* Privater Benutzercode -----------------------------------------------------*/
/* BENUTZERCODE AB 0 */
#define RxBuf_SIZE 64
      //!< Größe des Rx-Buffers, der zum Empfang von Daten über UART verwendet wird.
#define MainBuf_SIZE 64 //!< Größe des Hauptpuffers, der zum Speichern der über UART
empfangenen Daten verwendet wird.
#define commandBuf_SIZE 64

uint8_t RxBuf[RxBuf_SIZE]; //!< Rx Buffer wird verwendet, um Daten über UART zu emp-
fangen.
uint8_t MainBuf[MainBuf_SIZE];                                          //!<
Main Buffer wird verwendet, um die über UART empfangenen Daten zu speichern.
float value = 0.0; //!< Float-Variable zur Speicherung des Ausgangswerts des DAC-Ka-
nals.
uint32_t VAR = 0;
      //!< VAR zum Speichern des jeweiligen Digitalwertes für den Ausgang des DAC.
static uint16_t MainBufCounter ; //!< Statische Variable, die den aktuellen Eintrag
des Main-Buffers festhält.
char str[20]={'0'};
char ch = '\n' ;

//Dieser Puffer wird verwendet, um den empfangenen Befehl zu speichern.
uint8_t commandBuf[commandBuf_SIZE];
//Dieses Flag wird verwendet, um den aktiven Kanalwert zu speichern.
uint8_t activeChannel = 0;
//Dieses Flag wird verwendet, um zu prüfen, ob der neue Befehl empfangen wurde.
bool newCommandReceived = false;

//Diese Flaggen zeigen den Status des Kanals an.
bool isChannel1Active = false;
bool isChannel2Active = false;

//Diese werden zur Überprüfung des empfangenen Befehls verwendet.
char *retCommand1;
char *retCommand2;
char *retCommand3;
char *retCommand4;
char *retCommand5;
```

```
/* Diese Funktion ist die Callback-Funktion des UART-Empfangsinterrupts.
   Diese Funktion wird immer dann ausgeführt, wenn Daten am RX-Pin des UART verfügbar
sind.
*/
void HAL_UARTEx_RxEventCallback(UART_HandleTypeDef *huart, uint16_t Size)
{
	//Variable, die zum Kopieren von Daten aus dem RX-Puffer in den Hauptpuffer
verwendet wird.
	uint8_t Zähler = 0 ;

	//Prüfen, ob der Interrupt vom USART2 kommt. Da wir derzeit USART 2 verwenden
	if(huart -> Instanz == USART2 )
	{
		//Schleife zum Kopieren der Daten vom Rx-Puffer in den Hauptpuffer.
		while(Zähler <= Größe)
		{
			//Kopieren von Daten aus dem Rx-Puffer in den Hauptpuffer.
			MainBuf[MainBufCounter] = RxBuf[Counter++] ;
			//Prüfen Sie auf das Zeichen \n, das angibt, dass der gesamte Be-
fehl empfangen wurde.
if(MainBuf[MainBufCounter] == '\n')
			{ //den empfangenen Befehl in den Befehlspuffer einfügen.
				for(int i = 0 ; i < (MainBufCounter); i++)
				{
					commandBuf[i] = MainBuf[i];
				}
				//Flagge, die anzeigt, dass der neue Befehl empfangen
wurde.
				newCommandReceived = true;
			}

		//Prüfen Sie, dass am Ende jeder Datenkopie keine Null hinzugefügt wird.
			if(Zähler - 1 != Größe)
			{
				//Erhöhen des Zählers des Hauptpuffers
				HauptBufZähler++;
			}

			//Prüfen, ob der Hauptpuffer voll mit Daten ist. In diesem Fall
werden die Daten von Anfang an ersetzt.
			if(MainBufCounter > 64 )
		{
				//Rücksetzen des Hauptpufferzählers.
			HauptBufZähler = 0;
			}
		}

		//Diese Funktion wird verwendet, um Daten von UART mit DMA zu empfangen,
bis der Datenpuffer voll ist oder die Leerlaufleitung erkannt wird.
		HAL_UARTEx_ReceiveToIdle_DMA(&huart2, RxBuf, RxBuf_SIZE);
		//Deaktivieren Sie den Interrupt für die halb übertragenen Daten.
		__HAL_DMA_DISABLE_IT(&hdma_usart2_rx, DMA_IT_HT);

	}
}
```

```
/*
 * Benutzerdefinierte Funktion zur Übertragung von String-Daten auf UART.
 * Nimmt eine Zeichenkette vom Benutzer entgegen.
 * gibt keine zurück.
 * */
void Uprintf(char *str)
{
      //Funktion zur Übertragung von Daten auf UART.
      HAL_UART_Transmit(&huart2 ,(uint8_t*) str, strlen(str),1000);
}
//Diese Funktion wird verwendet, um den Float-Wert aus einer Zeichenkette zu erhal-
ten.
double get_double(const unsigned char *str)
{
    /* Erste nicht-zifferige Zeichen auslassen */
    /* Sonderfall zur Behandlung negativer Zahlen */
    while (*str && !(isdigit(*str) || ((*str == '-' || *str == '+') && isdigit(*(str
+ 1)))))
        str++;

    /* Das Parsen in ein Double */
    return strtod((const char *)str, NULL);
}
//Diese Funktion wird bei der Umwandlung von Fließkommazahlen in Zeichenketten ver-
wendet.
int n_tu(int number, int count)
{
    int Ergebnis = 1;
    while(count-- > 0)
        Ergebnis *= Zahl;

    Ergebnis zurückgeben;
}
//Diese Funktion wird verwendet, um Float-Werte in Strings umzuwandeln.
void float_to_string(float f, char r[])
{
    long long int length, length2, i, number, position, sign;
    Schwimmer Nummer2;

    Vorzeichen = -1; // -1 == positive Zahl
    wenn (f < 0)
    {
        Vorzeichen = '-';
        f *= -1;
    }

    Nummer2 = f;
    Zahl = f;
    length = 0; // Größe des Dezimalteils
    length2 = 0; // Größe des Zehntels

    /* Berechnung des zehnten Teils der Länge2 */
    while( (number2 - (float)number) != 0.0 && !((number2 - (float)number) < 0.0) )
    {
         number2 = f * (n_tu(10.0, length2 + 1));
```

```
        Nummer = Nummer2;

        Länge2++;
    }

    /* Berechnung der Länge des Dezimalteils */
    for (length = (f > 1) ? 0 : 1; f > 1; length++)
        f /= 10;

    Position = Länge;
    Länge = Länge + 1 + Länge2;
    Nummer = Nummer2;
    wenn (Vorzeichen == '-')
    {
        Länge++;
        Position++;
    }

    for (i = length; i >= 0 ; i--)
    {
        wenn (i == (Länge))
            r[i] = '\0';
        sonst if(i == (Position))
            r[i] = '.';
        else if(Vorzeichen == '-' && i == 0)
            r[i] = '-';
        sonst
        {
            r[i] = (Zahl % 10) + '0';
            Zahl /=10;
        }
    }
}



/* BENUTZERCODE ENDE 0 */

/**
  * @brief Der Einstiegspunkt der Anwendung.
  * @retval int
  */
int main(void)
{
  /* BENUTZERCODE ANFANG 1 */

  /* BENUTZERCODE ENDE 1 */

  /* MCU Configuration--------------------------------------------------------*/

  /* Reset aller Peripheriegeräte, Initialisierung der Flash-Schnittstelle und des
Systicks. */
  HAL_Init();
```

```
  /* USER CODE BEGIN Init */

  /* USER CODE END Init */

  /* Konfigurieren Sie die Systemuhr */
  SystemClock_Config();

  /* BENUTZER CODE BEGIN SysInit */

  /* BENUTZER CODE END SysInit */

  /* Initialisierung aller konfigurierten Peripheriegeräte */
  MX_GPIO_Init();
  MX_DMA_Init();
  MX_DAC1_Init();
  MX_USART2_UART_Init();
  /* BENUTZERCODE ANFANG 2 */
  //Diese Funktion wird verwendet, um Daten von UART mit DMA zu empfangen, bis der
Datenpuffer voll ist oder die Leerlaufleitung erkannt wird.
  HAL_UARTEx_ReceiveToIdle_DMA(&huart2, RxBuf, RxBuf_SIZE);
  //Deaktivieren Sie den Interrupt für die halb übertragenen Daten.
  __HAL_DMA_DISABLE_IT(&hdma_usart2_rx, DMA_IT_HT);
  //Standardmäßig ist Kanal 1 aktiv.
  HAL_DAC_Start(&hdac1 , DAC_CHANNEL_1);
  isChannel1Active = true;
  activeChannel = 1;
  /* BENUTZERCODE ENDE 2 */

  /* Endlosschleife */
  /* BENUTZERCODE BEGIN WHILE */
  //Endlosschleife, um die MCU zu beschäftigen.
  während (1)
  {
      /* BENUTZERCODE END WHILE */

      /* BENUTZERCODE ANFANG 3 */
        //Prüfen, ob der neue Befehl empfangen wurde.
  if(newCommandReceived == true)
          {
             //Prüfen, ob der "INSTrument:SELect?" Befehl empfangen wurde.
        retCommand1 = strstr((char *)commandBuf, "INSTrument:SELect?");
             //Prüfen, ob der "SOURce:VOLTage:LEVel?" Befehl empfangen wird.
        retCommand2 = strstr((char *)commandBuf, "SOURce:VOLTage:LEVel?");
             //Prüfen, ob der Befehl "INSTrument:SELect:OUTPut1" empfangen wird.
        retCommand3 = strstr((char *)commandBuf, "INSTrument:SELect:OUTPut1");
             //Prüfen, ob der Befehl "INSTrument:SELect:OUTPut2" empfangen wird.
        retCommand4 = strstr((char *)commandBuf, "INSTrument:SELect:OUTPut2");
             //Prüfen, ob der "SOURce:VOLTage:LEVel:" Befehl empfangen wird.
        retCommand5 = strstr((char *)commandBuf, "SOURce:VOLTage:LEVel:");

      if(retBefehl1)
            {
             //Prüfung auf den aktiven Kanal
         if(activeChannel == 1u)
               {
```

```
                //Senden Sie den aktiven Kanalstring zurück
                Uprintf("OutPut 1\n");
               }
          sonst
              //Prüfung auf den aktiven Kanal
          if(activeChannel == 2u)
                {
                //Senden Sie den aktiven Kanalstring zurück
                Uprintf("OutPut 2\n");
               }
          //Rücksetzen des commandBuf und des MainBuf sowie des Flags für den neuen
empfangenen Befehl
                newCommandReceived = false;
          memset(commandBuf , 0 ,commandBuf_SIZE*(sizeof(commandBuf[0])));
                HauptBufZähler = 0 ;
          memset(MainBuf , 0 ,MainBuf_SIZE*(sizeof(MainBuf[0])));

              }
       if(retBefehl2)
              {
       //den Spannungswert in einen Stringwert umwandeln.
               float_to_string(wert,str);
               //Senden des Spannungswertes über UART
              Uprintf(strncat((char *)str,&ch,1));
             //Rücksetzen des commandBuf und des MainBuf und des Flags für den neuen
empfangenen Befehl
              newCommandReceived = false;
               memset(commandBuf , 0 ,commandBuf_SIZE*(sizeof(commandBuf[0])));
              HauptBufZähler = 0 ;
               memset(MainBuf , 0 ,MainBuf_SIZE*(sizeof(MainBuf[0])));
              }
       if(retBefehl3)
              {
            if(isChannel1Active == false)
                  {
             //Aktivieren Sie den Ausgang von DAC-Kanal 1;
                    HAL_DAC_Start(&hdac1 , DAC_CHANNEL_1);
                    isChannel1Active = true;
                  }

                  activeChannel = 1u;
           //Rücksetzen des commandBuf und des MainBuf sowie des Flags für den neuen
empfangenen Befehl
                newCommandReceived = false;
          memset(commandBuf , 0 ,commandBuf_SIZE*(sizeof(commandBuf[0])));
                HauptBufZähler = 0 ;
          memset(MainBuf , 0 ,MainBuf_SIZE*(sizeof(MainBuf[0])));
              }

       if(retBefehl4)
              {
               if(isChannel2Active == false)
              {
            //Aktivieren Sie den Ausgang von DAC-Kanal 1;
                  HAL_DAC_Start(&hdac1 , DAC_CHANNEL_2);
```

```
                isChannel2Active = true;
            }

            activeChannel = 2u;
             //Rücksetzen des commandBuf und des MainBuf sowie des Flags für den
neuen empfangenen Befehl
            newCommandReceived = false;
             memset(commandBuf , 0 ,commandBuf_SIZE*(sizeof(commandBuf[0])));
            HauptBufZähler = 0 ;
             memset(MainBuf , 0 ,MainBuf_SIZE*(sizeof(MainBuf[0])));
            }
      if(retBefehl5)
            {
               //Extrahieren Sie den Spannungswert aus dem Befehl.
                value = get_double(commandBuf);
         //Digitalwertberechnung zum Schreiben auf den DAC.
                VAR = Wert * (0xfff + 1) / 3.3;
         if(activeChannel == 1u)
                {
       //Diese Funktion schreibt den berechneten digitalen Wert in den DAC-Kanal.
              HAL_DAC_SetValue(&hdac1 , DAC_CHANNEL_1 , DAC_ALIGN_12B_R , VAR);
                }
         sonst
         if(activeChannel == 2u)
             {
          //Diese Funktion schreibt den berechneten digitalen Wert in den DAC-Kanal.
                  HAL_DAC_SetValue(&hdac1 , DAC_CHANNEL_2 , DAC_ALIGN_12B_R , VAR);
             }
         //Rücksetzen des commandBuf und des MainBuf sowie des Flags für den neuen
empfangenen Befehl
                newCommandReceived = false;
         memset(commandBuf , 0 ,commandBuf_SIZE*(sizeof(commandBuf[0])));
                HauptBufZähler = 0 ;
         memset(MainBuf , 0 ,MainBuf_SIZE*(sizeof(MainBuf[0])));

            }
          }


  }
  /* BENUTZERCODE ENDE 3 */
}

/**
  * @brief System Clock Konfiguration
  * @retval Keine
  */
void SystemClock_Config(void)
{
  RCC_OscInitTypeDef RCC_OscInitStruct = {0};
  RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};
  RCC_PeriphCLKInitTypeDef PeriphClkInit = {0};

  /** Initialisiert die RCC-Oszillatoren gemäß den angegebenen Parametern
  * in der Struktur RCC_OscInitTypeDef.
```

```
  */
  RCC_OscInitStruct. OszillatorTyp = RCC_OSCILLATORTYPE_HSI;
  RCC_OscInitStruct. HSIState = RCC_HSI_ON;
  RCC_OscInitStruct. HSICalibrationValue = RCC_HSICALIBRATION_DEFAULT;
  RCC_OscInitStruct. PLL. PLLState = RCC_PLL_ON;
  RCC_OscInitStruct. PLL. PLLSource = RCC_PLLSOURCE_HSI;
  RCC_OscInitStruct. PLL. PLLM = 1;
  RCC_OscInitStruct. PLL. PLLN = 10;
  RCC_OscInitStruct. PLL. PLLP = RCC_PLLP_DIV7;
  RCC_OscInitStruct. PLL. PLLQ = RCC_PLLQ_DIV2;
  RCC_OscInitStruct. PLL.PLLR = RCC_PLLR_DIV2;
  if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
  {
    Error_Handler();
  }
  /** Initialisiert die CPU-, AHB- und APB-Bus-Takte
  */
  RCC_ClkInitStruct. ClockType = RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
                               |RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
  RCC_ClkInitStruct. SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
  RCC_ClkInitStruct. AHBCLKDivider = RCC_SYSCLK_DIV1;
  RCC_ClkInitStruct. APB1CLKDivider = RCC_HCLK_DIV1;
  RCC_ClkInitStruct. APB2CLKDivider = RCC_HCLK_DIV1;

  if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_4) != HAL_OK)
  {
    Error_Handler();
  }
  PeriphClkInit. PeriphClockSelection = RCC_PERIPHCLK_USART2;
  PeriphClkInit. Usart2ClockSelection = RCC_USART2CLKSOURCE_PCLK1;
  if (HAL_RCCEx_PeriphCLKConfig(&PeriphClkInit) != HAL_OK)
  {
    Error_Handler();
  }
  /** Konfigurieren Sie die Ausgangsspannung des internen Hauptreglers
  */
  if (HAL_PWREx_ControlVoltageScaling(PWR_REGULATOR_VOLTAGE_SCALE1) != HAL_OK)
  {
    Error_Handler();
  }
}

/**
  * @brief DAC1 Initialisierungsfunktion
  * @param Keine
  * @retval Keine
  */
static void MX_DAC1_Init(void)
{

  /* BENUTZER CODE BEGIN DAC1_Init 0 */

  /* BENUTZER CODE END DAC1_Init 0 */

  DAC_ChannelConfTypeDef sConfig = {0};
```

```
  /* BENUTZERCODE BEGIN DAC1_Init 1 */

  /* BENUTZER CODE END DAC1_Init 1 */
  /** DAC-Initialisierung
  */
  hdac1.Instanz = DAC1;
  if (HAL_DAC_Init(&hdac1) != HAL_OK)
  {
    Error_Handler();
  }
  /** Konfiguration des DAC-Kanals OUT1
  */
  sConfig.DAC_SampleAndHold = DAC_SAMPLEANDHOLD_DISABLE;
  sConfig.DAC_Trigger = DAC_TRIGGER_NONE;
  sConfig.DAC_OutputBuffer = DAC_OUTPUTBUFFER_ENABLE;
  sConfig.DAC_ConnectOnChipPeripheral = DAC_CHIPCONNECT_DISABLE;
  sConfig.DAC_UserTrimming = DAC_TRIMMING_FACTORY;
  if (HAL_DAC_ConfigChannel(&hdac1, &sConfig, DAC_CHANNEL_1) != HAL_OK)
  {
    Error_Handler();
  }
  /** Konfiguration des DAC-Kanals OUT2
  */
  if (HAL_DAC_ConfigChannel(&hdac1, &sConfig, DAC_CHANNEL_2) != HAL_OK)
  {
    Error_Handler();
  }
  /* BENUTZERCODE BEGIN DAC1_Init 2 */

  /* BENUTZER CODE END DAC1_Init 2 */

}

/**
  * @brief USART2 Initialisierungsfunktion
  * @param Keine
  * @retval Keine
  */
static void MX_USART2_UART_Init(void)
{

  /* BENUTZERCODE BEGIN USART2_Init 0 */

  /* BENUTZER CODE END USART2_Init 0 */

  /* BENUTZERCODE BEGIN USART2_Init 1 */

  /* BENUTZER CODE END USART2_Init 1 */
  huart2.Instanz = USART2;
  huart2.Init. BaudRate = 115200;
  huart2.Init. WortLänge = UART_WORDLENGTH_8B;
  huart2.Init. StopBits = UART_STOPBITS_1;
  huart2.Init. Parität = UART_PARITY_NONE;
  huart2.Init. Modus = UART_MODE_TX_RX;
```

```
  huart2.Init. HwFlowCtl = UART_HWCONTROL_NONE;
  huart2.Init. OverSampling = UART_OVERSAMPLING_16;
  huart2.Init. OneBitSampling = UART_ONE_BIT_SAMPLE_DISABLE;
  huart2.AdvancedInit. AdvFeatureInit = UART_ADVFEATURE_NO_INIT;
  if (HAL_UART_Init(&huart2) != HAL_OK)
  {
    Error_Handler();
  }
  /* BENUTZERCODE BEGIN USART2_Init 2 */

  /* BENUTZER CODE END USART2_Init 2 */

}

/**
  * DMA-Controller-Takt einschalten
  */
static void MX_DMA_Init(void)
{

  /* Freigabe des DMA-Controller-Takts */
  __HAL_RCC_DMA1_CLK_ENABLE();

  /* DMA-Interrupt-Initialisierung */
  /* DMA1_Channel6_IRQn Interrupt-Konfiguration */
  HAL_NVIC_SetPriority(DMA1_Channel6_IRQn, 0, 0);
  HAL_NVIC_EnableIRQ(DMA1_Channel6_IRQn);

}

/**
  * @brief GPIO Initialisierungsfunktion
  * @param Keine
  * @retval Keine
  */
static void MX_GPIO_Init(void)
{
  GPIO_InitTypeDef GPIO_InitStruct = {0};

  /* Taktfreigabe der GPIO-Anschlüsse */
  __HAL_RCC_GPIOC_CLK_ENABLE();
  __HAL_RCC_GPIOH_CLK_ENABLE();
  __HAL_RCC_GPIOA_CLK_ENABLE();
  __HAL_RCC_GPIOB_CLK_ENABLE();

  /*GPIO-Pin konfigurieren: B1_Pin */
  GPIO_InitStruct. Pin = B1_Pin;
  GPIO_InitStruct. Modus = GPIO_MODE_IT_FALLING;
  GPIO_InitStruct. Pull = GPIO_NOPULL;
  HAL_GPIO_Init(B1_GPIO_Port, &GPIO_InitStruct);

}

/* BENUTZERCODE ANFANG 4 */
```

```
/* BENUTZERCODE ENDE 4 */

/**
  * @brief Diese Funktion wird bei Auftreten eines Fehlers ausgeführt.
  * @retval Keine
  */
void Error_Handler(void)
{
  /* USER CODE BEGIN Error_Handler_Debug */
  /* Der Benutzer kann seine eigene Implementierung hinzufügen, um den HAL-Fehler-
rückgabezustand zu melden */
  __disable_irq();
  während (1)
  {
  }
  /* USER CODE END Error_Handler_Debug */
}

#ifdef USE_FULL_ASSERT
/**
  * @brief Meldet den Namen der Quelldatei und die Quellzeilennummer
  * wo der assert_param-Fehler aufgetreten ist.
  * @param file: Zeiger auf den Namen der Quelldatei
  * @param line: assert_param Fehlerzeile Quellennummer
  * @retval Keine
  */
void assert_failed(uint8_t *file, uint32_t line)
{
  /* BENUTZERCODE AB 6 */
  /* Der Benutzer kann seine eigene Implementierung hinzufügen, um den Dateinamen und
die Zeilennummer zu melden,
     ex: printf("Falscher Parameterwert: Datei %s in Zeile %d\r\n", Datei, Zeile) */
  /* BENUTZERCODE ENDE 6 */
}
#endif /* USE_FULL_ASSERT */

/************************ (C) COPYRIGHT STMicroelectronics *****END OF FILE****/
```