HAW HAMBURG

Bachelorarbeit

Jonas Jegminat

Entwicklung eines achtkanaligen, arbitären Funktionsgenerators zur Netzsimulation

Fakultät Technik und Informatik
Department Informations- und Elektrotechnik

Faculty of Computer Science and Engineering
Department of Information and Electrical Engineering

Jonas Jegminat

Entwicklung eines achtkanaligen, arbitären Funktionsgenerators zur Netzsimulation

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung
im Studiengang *Bachelor of Science Elektro- und Informationstechnik*
am Department Informations- und Elektrotechnik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr.-Ing. Karl-Ragmar Riemschneider
Zweitgutachter: Prof. Dr. Pawel Buczek

Eingereicht am: 10. Juni 2020

Jonas Jegminat

Thema der Arbeit

Entwicklung eines achtkanaligen, arbitären Funktionsgenerators zur Netzsimulation

Stichworte

Demonstrator, Dreiphasen-Simulation, Elektronik, Mikroprozessoren, Embedded, Softwareentwicklung

Kurzzusammenfassung

Diese Arbeit umfasst die Entwicklung und Prototyperstellung eines arbitären Funktionsgenerators. Ziel ist es, eine einfache Lösung zur Simulation eines Dreiphasennetzes zu erarbeiten. Dabei soll die Software geschrieben werden und eine Schaltung erstellt werden, die die Kurvenformen der Spannungen analog ausgibt.

Jonas Jegminat

Title of Thesis

Development of an eight channel, arbitrary generator for power grid simulation

Keywords

Demonstrator, Power grid simulation, Electronics, Microprocessors, Embedded, Software engineering

Abstract

This paper involves the development and prototyping of an arbitrary waveform generator. The purpose is to find a simple solution to simulate a three phase power grid. The software has to be developed as well as a circuit to output the wave analogue.

Inhaltsverzeichnis

1 Einleitung

Diese Bachelorthesis beschreibt die Entwicklung eines Funktionsgenerators. In diesem Kapitel wird die Motivation der Arbeit erläutert. Danach werden die technischen Anforderungen besprochen.

1.1 Motivation

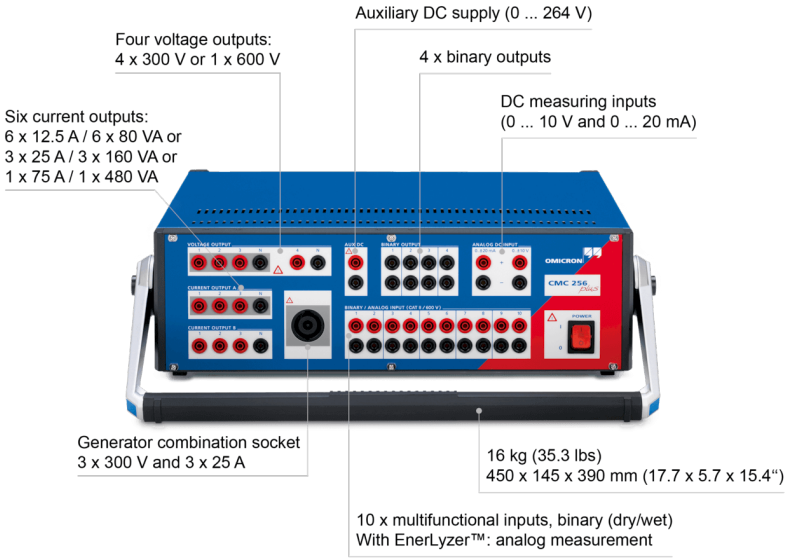
Die Firma Stucke Elektrotechnik GmbH entwickelt und produziert Netzschutz-Relais. Die Prüfung der Schutzmechanismen erfolgt zum Teil mit einem Netz-Simulator. Die Testabläufe müssen von dem Hersteller programmiert werden. Für viele Aufgaben in der Soft- und Hardwareentwicklung sind vereinfachte Parameter und Funktionen ausreichend. Daher soll ein kostengünstiger Funktionsgenerator entwickelt werden, der ein Dreiphasennetz nachbildet.

1.2 Übergeordnete Systembedingungen

Derzeitig wird ein Funktionsgenerator vom Typ OMICRON CMC 256plus eingesetzt, um die Funktionen der Netzschutz-Relais zu testen (siehe Abb. 1.1). In diesem Abschnitt wird die Leistung des Funktionsgenerators betrachtet. Daraufhin wird die Spezifikation für die Entwicklung des neuen Gerätes festgelegt.

In der Gerätesoftware werden die Testabläufe programmiert. Mit digitalen und analogen Messeingängen wird das Verhalten der Prüflinge überwacht. Dadurch wird ein fehlerhaftes Verhalten automatisch erkannt. Für den neuen Funktionsgenerator wird dies nicht benötigt. Die Spannung und die Ströme sollen nur modelliert und ausgegeben

werden. Einige technische Daten des Funktionsgenerators OMICRON CMC 256plus sind in der Tabelle 1.1 zusammengefasst.



Quelle: [3]

Abbildung 1.1: Funktionsgenerator OMICRON CMC 256plus

Parameter	**OMICRON**
Genauigkeit	Fehler < 0,04 %
Klirrfaktor	< 0,07 %
Auflösung Strom	0,004 %
Auflösung Spannung	0,003 %
Frequenzbereich Sinus	10...1000 Hz
Frequenzbereich Harmonische	10...3000 Hz
Frequenzauflösung	< 5 μHz
Bereich Phasewinkel	-360° ... +360°
Auslösung Phasenwinkel	0,001°
Bandbreite (-3dB)	3,1 kHz
Selbsttest	Ja

Tabelle 1.1: Datenblattauszug vom Funktionsgenerator OMICRON CMC 256plus [3]

Besonders bei den Genauigkeiten und den Auflösungen können Abstriche gemacht werden. Ein Fehler von bis zu 1 % ist für die Tests ausreichend. Die Netzschutz-Relais verwenden DAC mit einer Auflösung von 12 Bit. Daher können für die Auflösung von Spannungen und Strömen auch 12 Bit gewählt werden. Für das Projekt vorgegeben wurde ein Frequenzbereich von mindestens 20 Hz bis 75 Hz mit einer Frequenzauflösung von 0,1 Hz. Laut der Entwicklungsabteilung ist es für die Tests wichtig, dass die Frequenz konstant bleibt. Die Bandbreite soll beibehalten werden.

1.3 Detaillierte technische Anforderungen

In dieser Arbeit wird nur ein Teil des Funktionsgenerators entwickelt. Mit einem Entwicklerboard sollen Spannungen mit einer beliebigen Kurvenformen im Bereich -5 V bis 5 V ausgegeben werden. Der Aufbau soll über acht Kanäle verfügen. Vier Kanäle sollen die Netzspannungen abbilden, die anderen vier repräsentieren den Netzstrom. In einer weiteren Bachelorarbeit soll ein nachgeschalteter Verstärker entwickelt werden, der aus den acht Kanälen ein Vierleiter-Drehstromnetz nachbilden soll. Weiterhin soll auch die Entwicklung einer Anwendungsumgebung für den PC an diese Arbeit anschließen.

Parameter	**Anforderung**
Spannungsbereich	-5 V bis 5 V
Genauigkeit	Fehler < 1 %
THD	< 1 %
Auflösung Strom	12 Bit
Auflösung Spannung	12 Bit
Frequenzbereich Sinus	20...75 Hz
Frequenzbereich Harmonische	20...3000 Hz
Frequenzauflösung	0,1 Hz
Standardabweichung der Frequenz	<0,001 Hz
Bereich Phasenwinkel	-360° ...+360°
Auslösung Phasenwinkel	< 1°
Bandbreite (-3dB)	3 kHz
Selbsttest	Nein

Tabelle 1.2: Systemanforderungen für die Entwicklung des neuen Funktionsgenerators

Eine Schwingungsperiode soll durch 512 Werte dargestellt und wiederholt ausgegeben werden. Die Schwingung wird in diesem Teil der Entwicklung nicht geglättet, daher ist ein größerer Klirrfaktor zu erwarten.

Daraus ergeben sich die technischen Anforderungen für das Projekt (siehe Tab. 1.2).

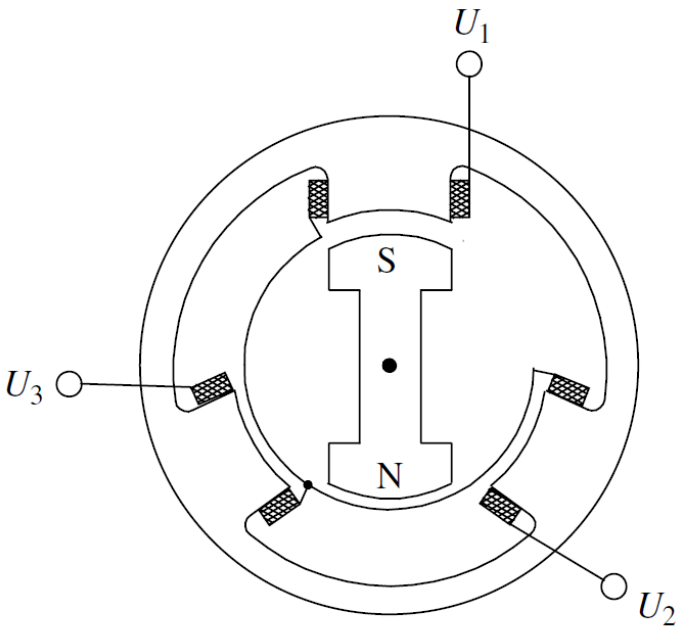
Es ist ein Platinenlayout zu erstellen, auf dem sich die DAC zusammen mit je einem OP befinden. Die OP-Schaltungen bereiten die Ausgangsspannungen der DAC für die weitere Verarbeitung auf. Die Kommunikation zu einem PC ist über eine USB-Schnittstelle herzustellen. Es sollen Steuerungsbefehle in einfachem Dateiformat ASCII sowie Dateien mit neuen Kurvenformen empfangen werden können.

Da weitere Arbeiten an das Projekt anknüpfen, ist eine detaillierte Dokumentation über Struktur und Funktionsweise besonders wichtig. Es wird daher eine kurze Bedienungsanleitung geschrieben. Die gesetzten Rahmenbedingungen, auftretende Probleme und wesentliche Folgerungen werden beschrieben, des Weiteren werden Verbesserungsvorschläge und Ansätze für die Weiterentwicklung genannt.

2 Grundlagen

In diesem Kapitel wird der Aufbau eines Drehstromnetzes erklärt, die Bewertungskriterien für die Qualität des Netzes werden angesprochen und die wichtigen Eigenschaften für die Simulation werden genannt.

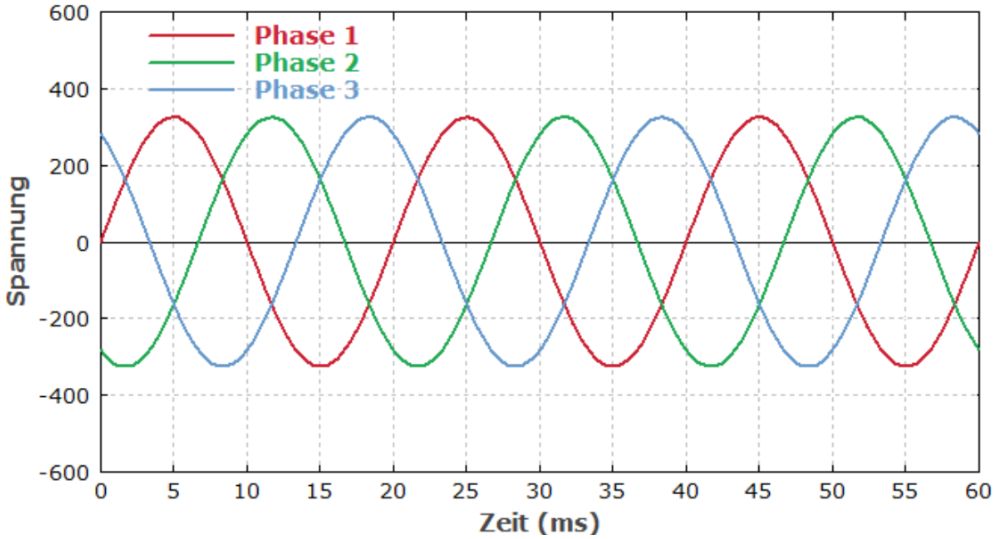
Ein Dreiphasennetz wird allgemein für die Übertragung elektrischer Energie zwischen Erzeuger und Verbraucher verwendet. Man erhält ein solches Spannungssystem, wenn man z.B. den Ständer der Maschine in Abbildung 2.1 mit drei Strängen U,V und W versieht. Die Wicklungen stehen symmetrisch im Winkel 120° zueinander [16, Seite 262].



Quelle: [16, Seite 263]

Abbildung 2.1: Generator mit drei symmetrisch versetzten Wicklungssträngen

Wird die Maschine betrieben, sind die Spannungskurven ebenfalls um 120° versetzt (siehe Abb. 2.2).



Quelle: [19]

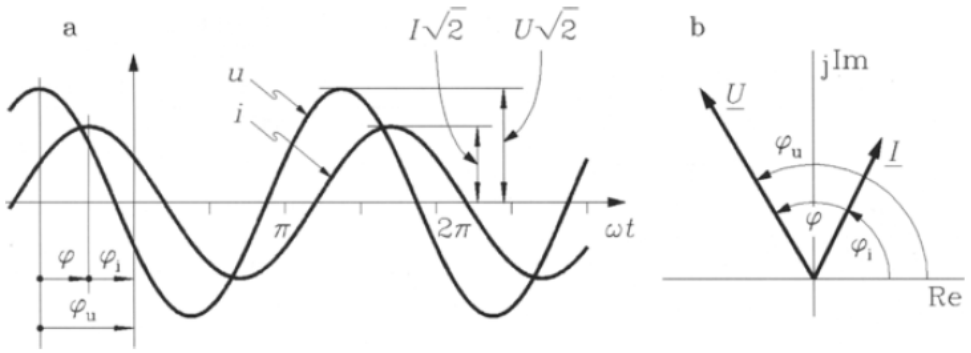
Abbildung 2.2: Spannungsverläufe L1, L2 und L3 zu dem Erdpotenial in einem Drehstromnetz um 120° versetzt

Allgemein wird die Spannung als Effektivwert angeben. Der Effektivwert ist die positive Quadratwurzel aus dem Mittelwert des Quadrates einer periodisch zeitabhängigen Größe [13, Seite 161].

$$X = \sqrt{\frac{1}{T} \cdot \int_0^T x^2 dt}$$

Der Name Effektivwert kommt zustande, da die im zeitlichen Mittel umgesetzte Leistung dem Produkt der Effektivwerte von einem zeitlich abhängigen Strom i und einer zeitlich abhängigen Spannung u entspricht. Der Effektivwert übernimmt damit die Rolle einer äquivalenten Gleichspannung oder einem äquivalenten Gleichstrom [13, Seite 161]. Im Gegensatz zu der zeitabhängigen Größe wird der Effektivwert mit einem Großbuchstaben ausgedrückt. Je nach Netz- und Lastimpedanzen stellen sich die Ströme mit einer bestimmten Phasenverschiebung ein. Die Abbildung 2.3 zeigt beispielhaft einen Versatz zwischen Spannung und Strom. Die Differenz der Winkel von Spannung und Strom nennt man Phasenverschiebungswinkel.

$$\varphi = \varphi_u - \varphi_i$$



Quelle: [13, Seite 163]

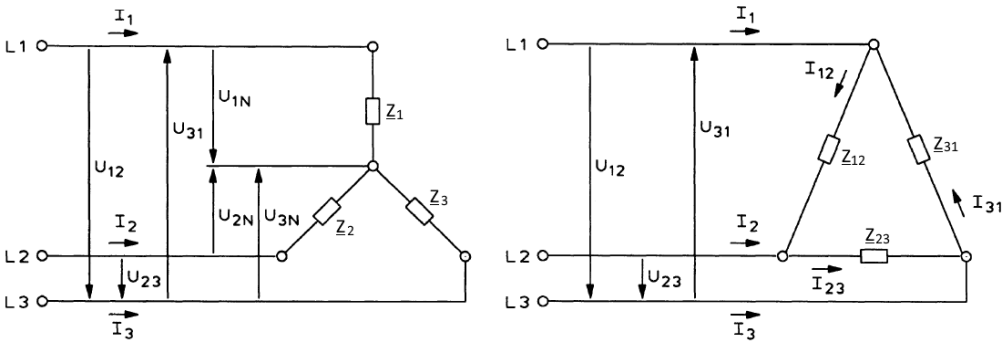
Abbildung 2.3: Phasenverschiebung: a. Sinusspannung und Sinusstrom in reeller Darstellung, b. Komplexe Darstellung in einem Zeigerdiagramm

Aus dem Winkel φ wird der Leistungsfaktor $\cos(\varphi)$ gebildet. Der Leistungsfaktor gibt an, wieviel Prozent der Scheinleistung als Wirkleistung im Verbraucher umgesetzt wird.

$$Leistungsfaktor\ cos(\varphi) = \frac{P}{S}$$

In der Praxis soll der Leistungsfaktor möglichst nahe bei eins liegen, damit die Betriebsmittel nicht unnötig mit Blindstrom belastet werden [20, Seite 290].

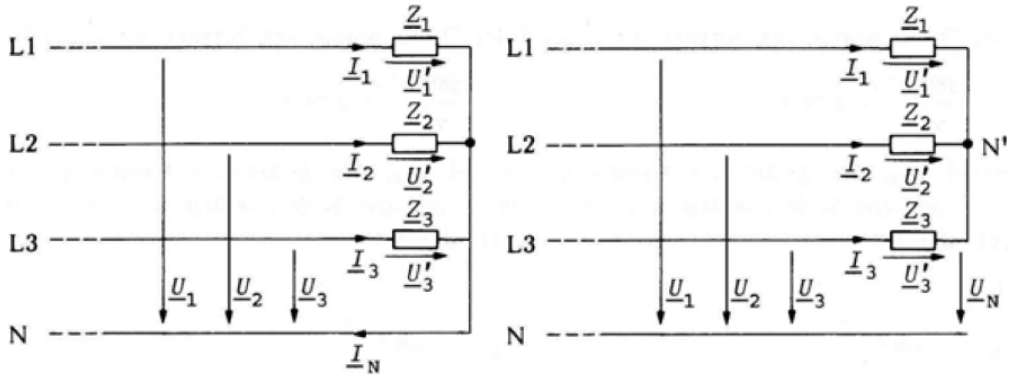
Im Versorgungsnetz können Verbraucher unterschiedlich beschaltet werden. In Abbildung 2.4 ist die Stern und die Dreiecksschaltung dargestellt.



Quelle: [14, Seite 94, angepasst]

Abbildung 2.4: Anschlussvarianten in einem Drehstromnetz. Links Sternschaltung, Rechts Dreieckschaltung

Bei der Sternschaltung kann der Mittelpunkt als zusätzlicher Leiter herausgeführt werden. Dieser Anschluss wird Neutralleiter genannt. Werden die drei spannungsführenden Leiter symmetrisch belastet, fließt kein Strom über den Neutralleiter, in einem unsymmetrisch belasteten Drehstromnetz jedoch immer. Wird ein Drehstromnetz ohne Neutralleiter unsymmetrisch belastet, kommt es zu einer Sternpunktverschiebung [16, 266]. Ein solcher Fall kann z.B. bei einem Leiterbruch auftreten. In der Abbildung 2.5 sind die beiden Fälle dargestellt.



Quelle: [16, Seite 266, angepasst]

Abbildung 2.5: Sternschaltung von Verbrauchern im Vierleiternetz. Links mit angeschlossenem Neutralleiter, rechts ohne angeschlossenem Neutralleiter

Als Folge werden die Ströme, die ansonsten über den Neutralleiter zurückfließen, auf die anderen beiden Außenleiter verteilt. Dadurch können die Strangspannungen von ihren üblichen Werten unter Umständen stark abweichen und Betriebsmittel zerstören.

Eine Last mit nichtlinearer Kennlinie $\mathrm{i(u)}$ zieht bei sinusförmiger Spannung einen verzerrten Strom aus dem Netz, dessen Verlauf periodisch zur Grundfrequenz ist. Jede mit einer Zeit periodische Funktion lässt sich in einer Fourierreihe zerlegen.

$$i = \sum_{\mathrm{v}=1}^{n} \hat{i}_{c\mathrm{v}} \cdot cos(\mathrm{v} \cdot \omega_1 \cdot t) + \hat{i}_{s\mathrm{v}} \cdot sin(\mathrm{v} \cdot \omega_1 \cdot t)$$
$$\hat{i}_{\mathrm{v}} = \sqrt{\hat{i}_{c\mathrm{v}}^2 + \hat{i}_{s\mathrm{v}}^2}$$

Die Komponenten $\mathrm{v} = 1, ..., \mathrm{n}$ werden Harmonische und die Komponenten $\mathrm{v} = 2, ..., \mathrm{n}$ Oberschwingungen genannt [20, Seite 379]. Die wichtigste nichtlineare Last ist der mit festem Zündwinkel angesteuerte Stromrichter. Wird der Gleichstrom so geglättet, dass er konstant ist, ergeben sich im Wechselstromnetz rechteckförmige Ströme. Diese lassen sich durch die Fourierreihe beschreiben mit

$$i_{\mathrm{v}}/i_1 = 1/\mathrm{v} \qquad \mathrm{v} = 1, 3, 5, 7, ...$$

Alle anderen Harmonischen sind nur in geringem Umfang vorhanden. Sie entstehen beispielsweise aus dem unsymmetrischen Aufbau der Stromtransformatoren. Auch Zwischenharmonische (z.B. $\mathrm{v} = 0,57$) können erzeugt werden, beispielsweise bei der Einspeisung von einem Netz in ein anderes unterschiedlicher Frequenz durch einen Umrichter.

Die Qualität eines Netzes kann anhand von verschiedenen Eigenschaften bewertet werden. Einige davon sind der Effektivwert der Spannung, die Frequenzstabilität, der Leistungsfaktor, der Scheitelfaktor, der Formfaktor und der Klirrfaktor/THD. Verhältniszahlen, wie der Scheitelfaktor oder der Formfaktor, werden aus den Mittelwerten von Spannung bzw. Strom gebildet. Sie dienen einer besseren Übersicht bei der Bewertung von Kurvenformen. Der Scheitelfaktor (crest factor) ist der Quotient aus Scheitelwert und Effektivwert und ist bei einem idealen Sinus gleich $\sqrt{2}$ [18, Seite 111].

$$Scheitelfaktor = \frac{Scheitelwert}{Effektivwert} \tag{2.1}$$

Der Formfaktor ist der der Quotient aus Effektivwert und Gleichrichtwert einer Wechselgröße. Bei einem idealen Sinus hat der Formfaktor einen Wert von etwa 1,11 [18, Seite 111].

$$Formfaktor = \frac{Effektivwert}{Gleichrichtwert}$$

Der Klirrfaktor ist ein Kennwert für die Abweichung der nichtsinusförmigen Größe von der Sinusgröße. Dieser setzt die Effektivwerte sämtlicher Oberschwingungen zum Effektivwert der Gesamtschwingung ins Verhältnis [18, Seite 239].

$$Klirrfaktor_u = \frac{\sqrt{\sum_{k=2}^{n} U_k^2}}{U}$$

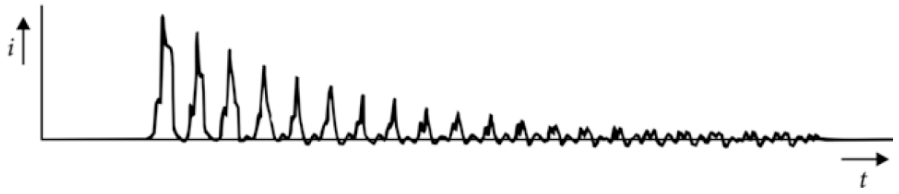
Statt dem Klirrfaktor wird auch häufig der Wert THD benutzt. Klirrfaktor und THD sind nicht identisch, gehen aber für kleine Werte ineinander über.

$$THD_u = \frac{\sqrt{\sum_{k=2}^{n} U_k^2}}{U_1}$$

Aufgabe der Netzsimulation ist es, das Dreiphasensystem und die möglichen Normal- und Fehlerzustände nachzubilden. Folgende Parameter müssen dabei modellierbar sein:

- Spannungen
- Ströme
- Frequenz
- Harmonische
- Phasenverschiebung

Mit den Netzschutz-Relais der Firma Stucke Elektrotechnik GmbH werden Generatoren, Transformatoren und Motoren vor einer Überlastung geschützt. Mit einem Netz-Simulator sollen für das Schutzobjekt teilweise gefährliche Netzzustände abgebildet werden, auf die das Netzschutz-Relais reagieren soll. Beispielsweise ist das Netzschutz-Relais mit einem Differentialschutz ausgestattet. Dabei werden ein- und ausfließende Ströme miteinander verglichen. So kann man auf einen inneren Fehler schließen. Dieser Schutz hat vornehmlich die Aufgabe, Kurzschlüsse innerhalb des Gerätes zu erkennen. Bei dem Einschalten eines Transformators kommt es ebenfalls zu einer Differenz ein- und ausfließender Ströme. Wie bei jedem Einschalten einer Induktivität entstehen Gleichstromglieder, die zur Sättigung des Eisens führen können [20, Seite 55]. Die Abbildung 2.6 zeigt einen solchen Stromverlauf.



Quelle: [20, Seite 56]

Abbildung 2.6: Typischer einphasiger Stromverlauf beim Zuschalten eines Transformators

Damit der Differentialschutz nicht auslöst, muss das charakteristische Einschaltverhalten des Transformators vom Netzschutz-Relais erkannt werden.

3 Einführung und Konzeption

In diesem Kapitel wird ein Konzept für die Umsetzung des Projektes entwickelt. Als erstes wird der Mikrocontroller betrachtet. Danach werden die Hardwarekomponenten ausgewählt und die wichtigen Eigenschaften für das Projekt besprochen. Anschließend wird die Platinenschaltung ausgelegt.

3.1 Analyse

In diesem Abschnitt wird die Eignung des Mikrocontrollers und die zur Verfügung stehenden Möglichkeiten der Umsetzung besprochen. Der Mikrocontroller soll eine Platine zur Energiewandlung steuern. Der schematische Aufbau ist in Abbildung 3.1 dargestellt.

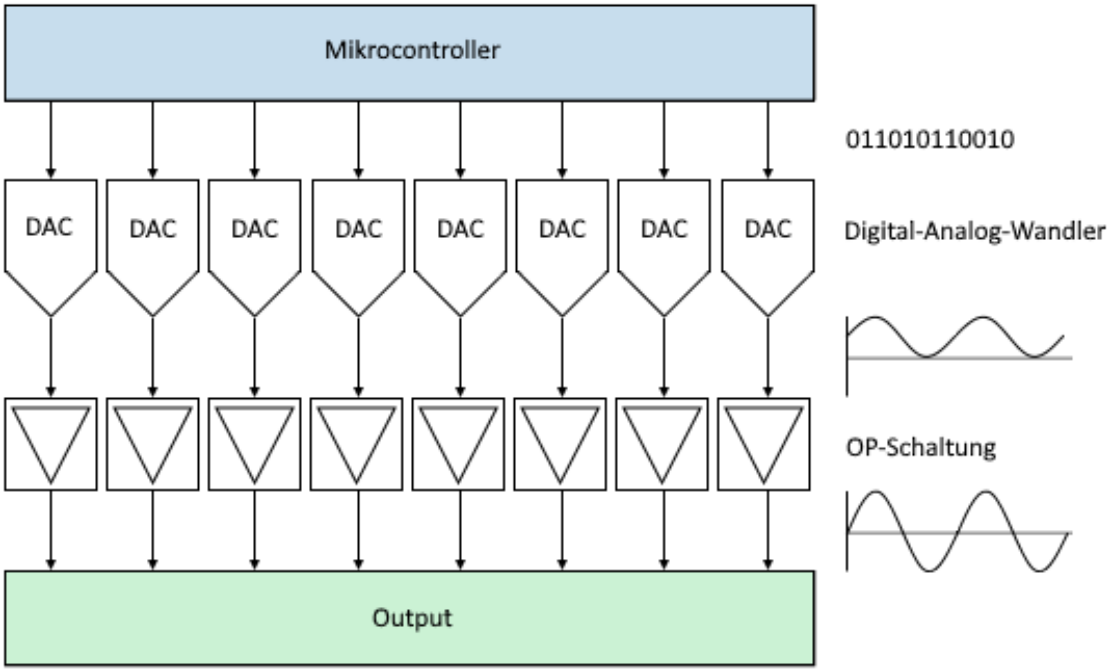
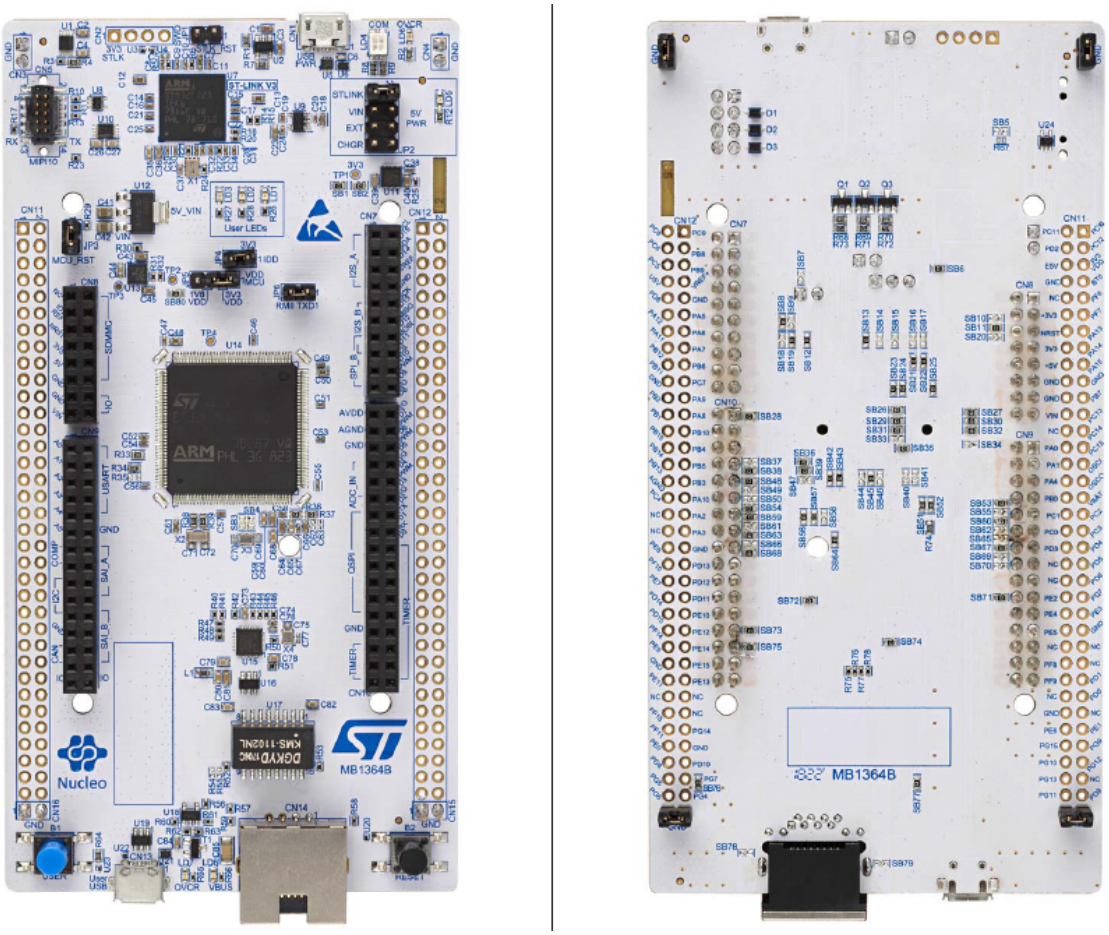


Abbildung 3.1: Prinzipieller Schaltungsaufbau für die Energiewandlung. Die digitalen Werte des Mikrocontrollers werden mit acht DAC und OP in acht um 0 V schwingende Spannungen gewandelt

Die acht Kanäle werden von einem Mikrocontroller gesteuert. Mit den DAC werden die digitalen Werte in eine analoge Spannung gewandelt. Die OP fügen der Spannung einen Offset und eine Verstärkung hinzu. Zusätzlich wird ein DC/DC-Wandler gebraucht, um eine negative Spannung am OP-Ausgang zu ermöglichen.

Das Projekt soll mit dem Entwicklerboard NUCLEO-144 STM32H743ZI umgesetzt werden (siehe Abb. 3.2). Die auf dem Board eingebaute CPU ist ein Arm Cortex-M7 32-bit. Zunächst wird untersucht, auf welche Weise das Projekt mit diesem Board realisierbar ist.



Quelle: [11, Seite 1]

Abbildung 3.2: Entwicklerboard NUCLEO-144 STM32H743ZI, Ober- und Unterseite

3.1.1 Versorgung

Die Energieversorgung kann über die USB-Schnittstelle CN1 oder über eine externe Quelle erfolgen [11, Seite 20]. Eine externe Versorgung mit einer höheren Spannung als 5 V ermöglicht einen größeren Aussteuerungsbereich für die OP. Jedoch ist lediglich eine Ausgangsspannung zwischen -5 V und +5 V gefordert. Außerdem soll der fertige Aufbau nur im Zusammenhang mit einem PC betrieben werden. Zur Einsparung weiterer Bauteile, wird die Energieversorgung über die USB-Schnittstelle CN1 realisiert. Der maximale Versorgungsstrom von 300 mA darf nicht überschritten werden [11, Seite 21]. Auch die maximale Belastung von 20 mA einzelner GPIOs und Kontrollpins muss bei der Auswahl der Bauteile und ihrer Beschaltung berücksichtigt werden [9, Seite 107]. Die Versorgungsspannung hängt damit von dem Entwicklerboard ab.

3.1.2 Kommunikation zu den DAC

Ein entscheidender Teil des Projektes ist die Kommunikation zu den DAC. Die Datenübertragung muss schnell genug sein, um den geforderten Frequenzbereich zu unterstützen. Sie kann parallel oder seriell erfolgen. Die parallele Übertragung ist deutlich schneller, da alle Bits gleichzeitig an die DAC gesendet werden. Diese Methode belegt jedoch einen Pin pro Bit. Unter Einsatz eines Decoders, können die Pins zur Selektierung auf vier begrenzt werden und würde damit 16 Pins belegen. Dennoch wird das serielle Verfahren gewählt. Denn wie in dem folgenden Abschnitt 3.1.3 gezeigt, ist die Geschwindigkeit einer seriellen Übertragung für das Projekt ausreichend. Das Einsparen der genutzten CPU-Abgänge ermöglicht dann ein kompakteres Platinenlayout und mehr Raum für Erweiterungen.

3.1.3 Ausgabefrequenz

Gemäß der Vorgabe sind die Kurvenformen mit einer Frequenz von mindestens 75 Hz auszugeben. Die 512 12 Bit Werte einer Periode müssen 75 Mal pro Sekunde an die

acht DAC gesendet werden. Wird von einer einkanaligen, seriellen Übertragung ausgegangen, berechnet sich die minimal geforderte Taktfrequenz des Transfers wie folgt:

$$Takt_{MIN} = Anzahl_{DAC} \cdot Werte_{pro\ Periode} \cdot Bits_{pro\ Wert} \cdot Frequenz_{MAX}$$
$$Takt_{MIN} = 8 \cdot 512 \cdot 12\ Bits \cdot 75\ Hz$$
$$Takt_{MIN} = \underline{3,69\,MHz}$$

Bei dem gestellten Entwicklerboard ist ein maximaler Peripherietakt von 240 MHz einstellbar [10, Seite 348]. Doch kann dieser je nach gewählter Peripherie auch geringer ausfallen. Alle unterstützten Peripherien können zur seriellen Übertragung auch mit wesentlich schnelleren Taktfrequenzen betrieben werden [10, Seite 352].

3.1.4 Speicher

Es muss genügend RAM-Speicher zur Verfügung stehen, um die Daten der Schwingungsperioden zu speichern. Es sollen zwei Netzprofile gleichzeitig im RAM-Speicher geladen sein. Dies ermöglicht den schnellen Wechsel von einem fehlerfreien zu einem fehlerhaften Netzzustand. Die Kurvenverläufe sollen in einem Array aus $\mathrm{uint16\_t}$ Werten gespeichert werden. Für jede Schwingung werden 512 Werte benötigt. Der dadurch belegte Speicherplatz berechnet sich wie folgt:

$$RAM_{MIN} = Anzahl_{DAC} \cdot Werte_{pro\ Periode} \cdot Bytes_{pro\ Wert} \cdot Anzahl_{Profile}$$
$$RAM_{MIN} = 8 \cdot 512 \cdot 2\,Byte \cdot 2$$
$$RAM_{MIN} = \underline{16\,kBytes}$$

Auf dem Board befindet sich 1 MByte RAM-Speicher [10, Seite 135]. Es ist genug Speicher vorhanden, um auch einen Puffer für den Empfang von Dateien einzurichten. Dadurch kann die Datei geprüft werden, bevor das alte Netzprofil überschrieben wird.

Weiterhin befinden sich auf dem Entwicklerboard 2 MByte Flash-Speicher [10, Seite 147]. Dadurch bietet sich die Möglichkeit, Netzprofile dauerhaft zu speichern. Diese Funktion wird hier nicht weiter verfolgt, da sie im Zusammenhang mit einem PC keinen zusätzlichen Nutzen mit sich bringt.

3.1.5 Kommunikation

Die Kommunikation mit einem PC soll über eine USB-Schnittstelle hergestellt werden. Auf dem Board befinden sich zwei USB-Buchsen (siehe Abb. 3.2). Um den Aufbau so einfach wie möglich zu halten, soll das USB-Kabel zur Energieversorgung auch zum Datentransfer genutzt werden.

3.1.6 OP-Schaltung

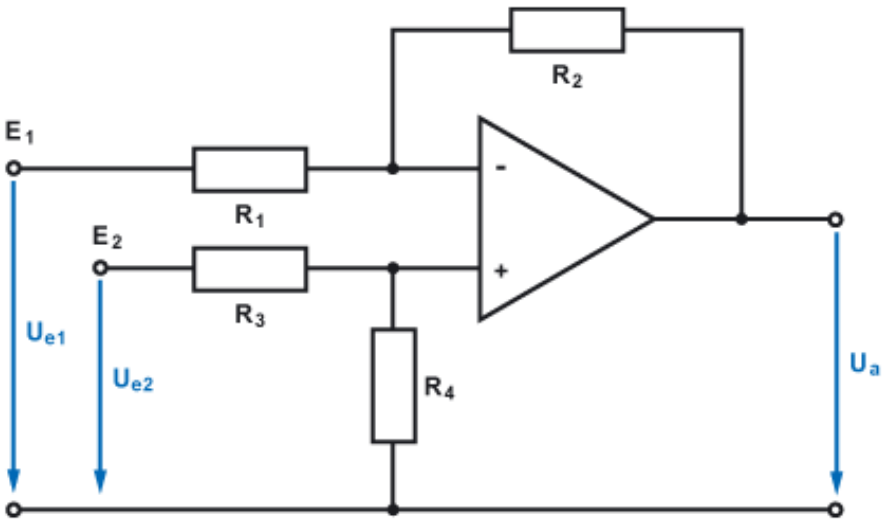
Zur Aufbereitung der Spannung wird pro Kanal ein Differenzverstärker genutzt. Das Verhalten der Schaltung wird mit den Gleichungen (3.1) bis (3.3) beschrieben. Zur Auslegung werden die Widerstandsverhältnisse für (3.2) benutzt. Die Subtraktion wirkt dann unabhängig von der Verstärkung [15, Seite 163].

$$U_a = U_{e2} \cdot \frac{R_4}{R_3 + R_4} \cdot (1 + \frac{R_2}{R_1}) - U_{e1} \cdot \frac{R_2}{R_1} \tag{3.1}$$

$$U_a = \frac{R_2}{R_1} \cdot (U_{e2} - U_{e1}) \quad | \; mit \; R_2 = R_4 \; und \; R_1 = R_3 \tag{3.2}$$

$$U_a = U_{e2} - U_{e1} \quad | \; mit \; R_1 = R_2 = R_3 = R_4 \tag{3.3}$$

Quelle: [15, Seite 163]



Quelle: [12]

Abbildung 3.3: Differenzverstärker: Subtrahierschaltung mit einem Operationsverstärker

3.1.7 Programmierumgebung

Der Mikrocontroller wird in der Programmiersprache C programmiert. Der Hersteller STMicroelectronics stellt dafür die eigene IDE STM32CubeIDE zur Verfügung. Diese basiert auf dem Programm Ecplise und ist bereits mit Informationen und Treibern für das Entwicklerboard ausgestattet. Besonders bei der Initialisierung des Taktes, der Peripherien, GPIO Pins und besonders der USB-Schnittstelle kann hiermit Einarbeitungszeit eingespart werden.

3.2 Auswahl der Hardwarekomponenten

In diesem Abschnitt werden die Bauteile ausgewählt und ihre Eignung für das Projekt begründet. Das wichtigste Kriterium ist die Kompatibilität zu den anderen Bauteilen. Die Kosten sollen möglichst gering gehalten werden. Deshalb, und um Engpässen bei der Lieferung vorzubeugen, werden Standardbauteile bevorzugt.

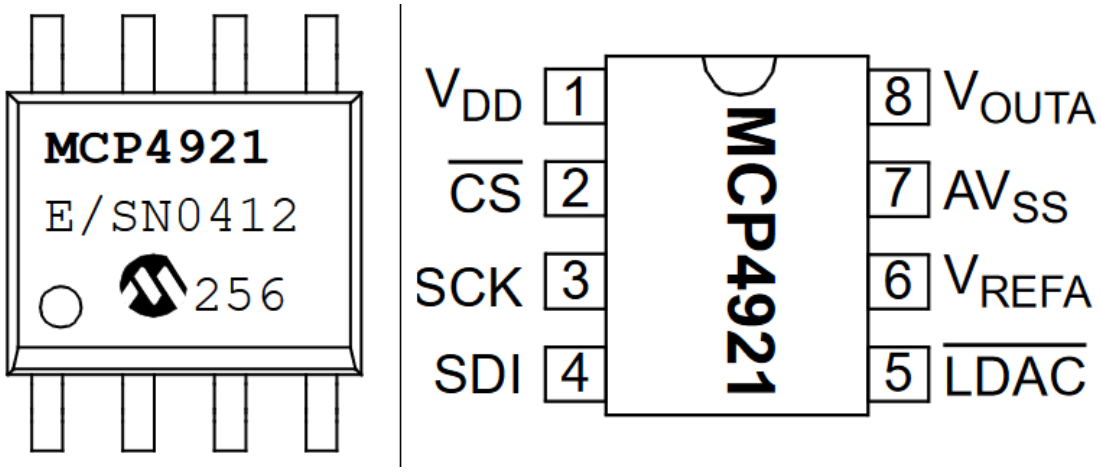
3.2.1 Auswahl des DAC

Als erstes wird der DAC-Typ ausgewählt. Die DAC-Typen in der engeren Auswahl sind in Tabelle 3.1 aufgelistet.

Pos	Typ	Serieller Bus	Beschaltung	Nutzbare Sonderfunktion
1	AD5321	I2C	2-Draht	-
2	MCP4921	SPI	3-Draht	Verriegelung
3	TLV5618A	SPI	3-Draht	-

Tabelle 3.1: Mögliche DAC-Typen mit Art der Ansteuerung

Das Projekt ist mit jedem der drei DAC-Typen realisierbar. Mit dem DAC-Typ AD5321 kann ein Draht eingespart werden. Allerdings können nur vier DAC an einen Bus angeschlossen werden. Die anderen beiden benötigen drei Drähte. Aufgrund der Verriegelungsfunktion wird der DAC-Typ MCP4921T-E/SN verbaut und kann so für eine synchrone Spannungsaktualisierung genutzt werden. Im Weiteren wird der DAC-Typ im Detail betrachtet. Unter anderem wird der Aufbau eines Datenpaketes gezeigt und mit der Slewrate untersucht, ob die Kurvenmodellierung durch den DAC-Typ eingeschränkt wird.



Quelle: [5, Seite 1, Seite 28]

Abbildung 3.4: Verwendeter DAC-Typ MCP4921T-E/SN, Bauart: SOIC, einkanalig, 8-Pin

Dem Datenblatt sind folgende Kenngrößen zu entnehmen:

Parameter	Kürzel	Min	Typ	Max	Einheit	Anforderung
Versorgungsspannung	V_{DD}	2,7		5,5	V	3,3 V bis 5,5 V
Referenzspannung	V_{REFA}	0		5,5	V	
Versorgungsstrom	I_{DD}		175	350	μA	
Ausgangskurzschlussstrom	I_{SC}			24	mA	
Eingangsimpedanz	Z_{IN}		165		kΩ	
Verriegelung	$\overline{LDAC}$	0		5,5	V	
Systemtakt	SCK			20	MHz	>5 MHz
Auflösung			12		Bit	12 Bit
Slewrate	SR		0,55		$\frac{V}{\mu s}$	

Tabelle 3.2: Datenblattauszug des ausgewählten DAC-Typ MCP4921T-E/SN mit den wichtigen Informationen für das Projekt [5]

Versorgung

Damit keine weitere Spannungsquelle eingebaut werden muss, werden die DAC mit der Versorgungsspannung des Entwicklerboards versorgt. Das Entwicklerboard stellt dafür einen Anschlusspin mit einer Spannung von 3,3 V und einen mit 5 V zur Verfügung [11, Seite 40], die Datenübertragung wird jedoch immer mit dem internen Pegel der

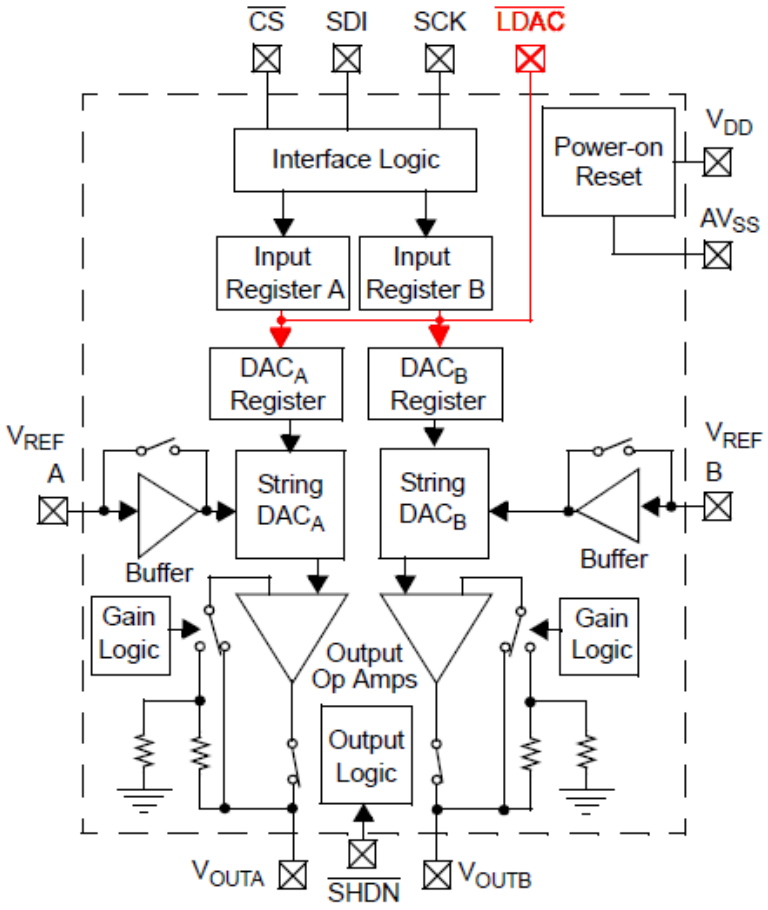
CPU von 3,3 V durchgeführt [11, Seite 27]. Der DAC wird mit einer Spannung von 3,3 V betrieben. Auch an die Referenzspannung werden die 3,3 V angeschlossen.

Die Strombelastung für das Entwicklerboard berechnet sich wie folgt:

$$I_{DAC} = Anzahl_{Kanle} \cdot I_{DD\ MAX}$$
$$I_{DAC} = 8 \cdot 350\,\mu A$$
$$I_{DAC} = \underline{2,8\,mA}$$

Verriegelung

Als Sonderfunktion verfügt der DAC über eine Verriegelung (Latch). Es sind zwei interne Registersektionen vorhanden, die durch die Verriegelung getrennt sind. Die Abb. 3.5 zeigt den Aufbau des DAC in zweikanaliger Bauweise.



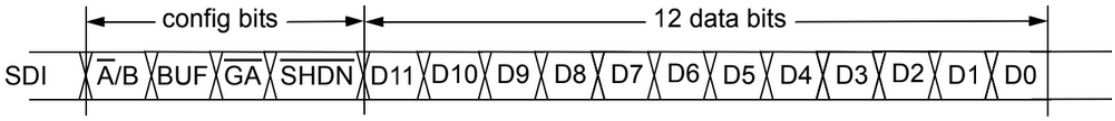
Quelle: [5, Seite 1] Angepasst

Abbildung 3.5: Blockdiagramm des zweikanaligen DAC MCP4922. Zwei interne Registersektionen ermöglichen die Steuerung des Zeitpunktes der Werteaktualisierung

Die Ausgangsspannung ergibt sich zu jedem Zeitpunkt aus der Referenzspannung und den Werten der Register DAC_A und DAC_B. Über SPI werden die Input-Register A und B beschrieben. Wird der $\overline{LDAC}$ Pin auf GND-Potential gesetzt, werden die Werte der Input-Registern A und B in die Register DAC_A und DAC_B übernommen. Diese Funktion ist für die Aufgabe nicht gefordert, sie wird aber dennoch genutzt, um eine synchrone Aktualisierung aller DAC zu garantieren.

Kommunikation

In Abschnitt 3.1.3 wurde bereits die Mindestfrequenz der seriellen Kommunikation bestimmt. Der gewählte DAC-Typ erwartet aber Werte mit 16 Bit. Die ersten vier Bits dienen der Konfiguration (siehe Abb.3.6).



Quelle: [5, Seite 19]

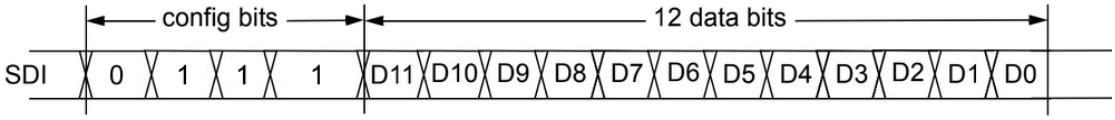
Abbildung 3.6: Aufbau des SPI Datenpakets für den DAC-Typ MCP4921. Vier Bits für zur Konfiguration und 12 Bits für die Daten

Mit den Konfigurationsbits werden die folgenden Einstellungen getroffen:

$\overline{A}$/B:	0 = Beschreiben von Register DAC_A
	1 = Beschreiben von Register DAC_B
BUF:	0 = V_{REF} unbuffered
	1 = V_{REF} buffered
$\overline{GA}$:	0 = Verstärkung: 2
	1 = Verstärkung: 1
$\overline{SHDN}$:	0 = Output hochohmig
	1 = Output aktiviert

Das Bit $\overline{A}$/B hat keine Funktion, da eine einkanalige Variante verbaut wird (siehe Abb. 3.4). Mit dem Bit BUF kann die Referenzspannung auf einen internen Verstärkereingang geführt werden. Dadurch wird ein sehr großer Eingangswiderstand erreicht [5, Seite 17]. Der DAC kann außerdem eine Verstärkung hinzufügen, was aber hier nicht nötig ist, da die Verstärkung nur von der OP-Schaltung eingestellt wird. Über das Bit

$\overline{\text{SHDN}}$ kann der Output abgeschaltet werden. Dies wird nicht genutzt, da die OP-Schaltung dann den niedrigsten Wert ausgibt. Damit beginnt jedes Paket mit der Bitfolge 0111 (siehe Abb. 3.7).



Quelle: [5, Abgeändert Seite 19]

Abbildung 3.7: Gewählte Einstellung der Konfigurationsbits für den DAC. Verstärkung von eins, Output immer eingeschaltet

Die SPI Taktfrequenz muss demnach entsprechend höher gewählt werden. Der DAC erlaubt einen Systemtakt von 20 MHz. Es wird ein Systemtakt von 16 MHz eingestellt.

Slewrate

Es soll auch möglich sein, eine Rechteckspannung mit dem Funktionsgenerator auszugeben. In der Realität ist das nicht ideal möglich, da die Steigung der Spannung nicht unbegrenzt hoch sein kann. Um die Kurvenmodellierung nicht einzuschränken, soll der maximale Spannungsschritt innerhalb einer Aktualisierung erreichbar sein.

Im Datenblatt wird dies unter dem Wert Slewrate (SR) angegeben. Der maximale Spannungsschritt liegt zwischen 0 V und 3,3 V. Innerhalb der Zeit T muss dieser erreicht werden können.

$$
\begin{aligned}
T &= \frac{1}{f_{MAX} \cdot Anzahl_{Werte\ pro\ Periode}} \\
T &= \frac{1}{75\,Hz \cdot 512} \\
T &= \underline{26,04\,\mu s}
\end{aligned}
$$

Die minimale Slewrate berechnet sich dann zu:

$$
\begin{aligned}
SR_{MIN} &= \frac{U_{\Delta MAX}}{T} \\
SR_{MIN} &= \frac{3,3\,V}{26,04\,\mu s} \\
SR_{MIN} &= \underline{0,127\,\frac{V}{\mu s}}
\end{aligned}
$$

Der DAC-Typ erfüllt diese Anforderung mit einer Slewrate von 0,55 $\frac{V}{\mu s}$ (siehe Tab.3.2).

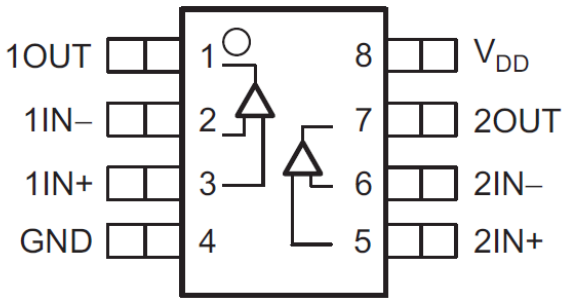
3.2.2 Auswahl des OP

Als nächstes Bauteil wird der OP-Typ gewählt. Drei mögliche OP-Typen sind in der Tabelle 3.3 dargestellt. Der OP soll mit der 5 V Versorgungsspannung des Entwick-

Pos	Typ	Maximale Spannung	Rail-to-Rail	Preis
1	LM358BAIDR	36 V	Nein	0,40 €
2	OPA2210ID	36 V	Ja	4,67 €
3	TLV2372IDR	16 V	Ja	1,52 €

Tabelle 3.3: Mögliche OP-Typen, Preise bei Firma Mouser [4] am 13.02.2020

lerboards betrieben werden. Das Spannungsniveau von -5 V wird mit dem DC/DC-Wandler erzeugt (siehe Abschnitt 3.2.3). Auch der Ausgangspegel soll von -5 V bis +5 V reichen. Es ist also unabdingbar, dass die Ausgangsspannung nahezu den Wert der Versorgungsspannung erreichen kann. Folglich wurde ein OP mit der Eigenschaft Rail-to-Rail ausgewählt. Dieser Begriff weist darauf hin, dass MOSFETs verwendet werden. Im Gegensatz zu der Bauweise mit Bipolartransitoren, entstehen im OP geringere Spannungsverluste. Bis auf wenige Millivolt kann so der Wert der Betriebsspannung erreicht werden [15, Seite 166]. Die beiden OP-Typen OPA2210ID und TLV2372IDR erfüllen die Anforderungen. In Absprache mit dem Layouter wurde sich für eine zweikanalige Bauweise entschieden. Damit soll ein kompakteres Layout erreicht werden. Aufgrund des günstigeren Bestellpreises wird der OP-Typ TLV2372IDR verbaut (siehe Abb. 3.8).



Quelle: [7, Seite 4]

Abbildung 3.8: Verwendeter OP-Typ TLV2372IDR, Bauart: SOIC, zweikanalig, 8-Pin

Dem Datenblatt sind folgende Kenngrößen zu entnehmen:

Parameter	Kürzel	Min	Typ	Max	Einheit	Anforderung
Versorgungsspannung	V_{DD}	±1,35		±8	V	±5 V
Versorgungsstrom	I_{DD}		550	660	$\frac{\mu\mathrm{A}}{\mathrm{Kanal}}$	
Input Noise Voltage	V_{N}		39		$\frac{\mathrm{nV}}{\sqrt{\mathrm{Hz}}}$	

Tabelle 3.4: Datenblattauszug des ausgewählten OP-Typ TLV2372IDR mit den wichtigen Informationen für das Projekt [7]

Im Weiteren wird der Anspruch an die Energieversorgung betrachtet.

Versorgung

Sollte die Spannungsversorgung weniger als 5 V bereitstellen, wird die Ausgangsspannung des OP auch nur weniger als 5 V erreichen können. Es ist also wichtig, dass es zu keinem Spannungseinbruch unter 5 V kommt.

Die maximale Strombelastung durch die OP ergibt sich durch den maximalen Eingangsstrom und den maximalen Ausgangsstrom, der abhängig von der angeschlossenen Last ist. Da es sich um einen Verstärkereingang handeln wird, kann von einem Widerstand im MΩ-Bereich ausgegangen werden. Im Weiteren wird von einem niedrigen Widerstand von 10 kΩ ausgegangen.

$$I_{DD\ MAX} = Anzahl_{Kanale} \cdot (I_{DD} + \frac{U_{OUT}}{R_L})$$
$$I_{DD\ MAX} = 8 \cdot (0.66\ mA + \frac{5\,V}{10\,k\Omega})$$
$$I_{DD\ MAX} = \underline{9,28\,mA}$$

Mit dieser Annahme ergibt sich eine Stromabnahme von 9.28 mA.

Slewrate

Auch die Ausgangsspannung des OP soll den maximalen Spannungsschritt innerhalb einer Aktualisierung erreichen können. Der maximale Spannungsschritt liegt zwischen

-5 V und 5 V. Dieser muss ebenfalls in der Zeit T erreicht werden können. Die minimale Slewrate berechnet sich dann zu:

$$SR_{MIN} = \frac{U_{\Delta MAX}}{T}$$
$$SR_{MIN} = \frac{10\,V}{26,04\,\mu s}$$
$$SR_{MIN} = \underline{0,384\,\frac{V}{\mu s}}$$

Der OP erfüllt diese Anforderung mit einer SR von 2,4 $\frac{V}{\mu s}$ (siehe Tab.3.4).

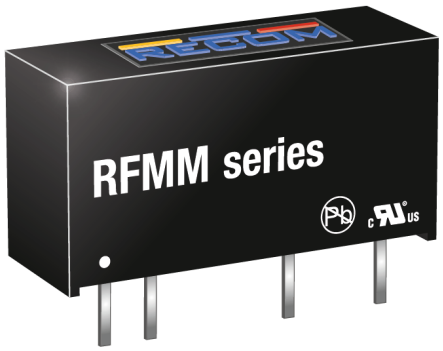
3.2.3 Auswahl des DC/DC-Wandlers

Für die negative Spannungsversorgung der OP wird ein DC/DC-Wandler benötigt. In Tabelle 3.5 sind drei mögliche Typen aufgelistet.

Pos	Typ	Ausgangsstrom	Effizienz	Preis
1	RFMM-0505S	200 mA	80 %	1,52 €
2	EC1SA01N	200 mA	79 %	2,70 €
3	CRR1S0505SC	200 mA	69 %	2,42 €

Tabelle 3.5: Mögliche DC/DC-Wandler, Preise bei Firma Mouser [4] am 13.02.2020

Aufgrund der größeren Effizenz und des geringen Preises wird der Typ RFMM-0505S verbaut (siehe Abb. 3.9).



Quelle: [8, Seite 1]

Abbildung 3.9: Verwendeter DC/DC-Wandler RFMM-0505S

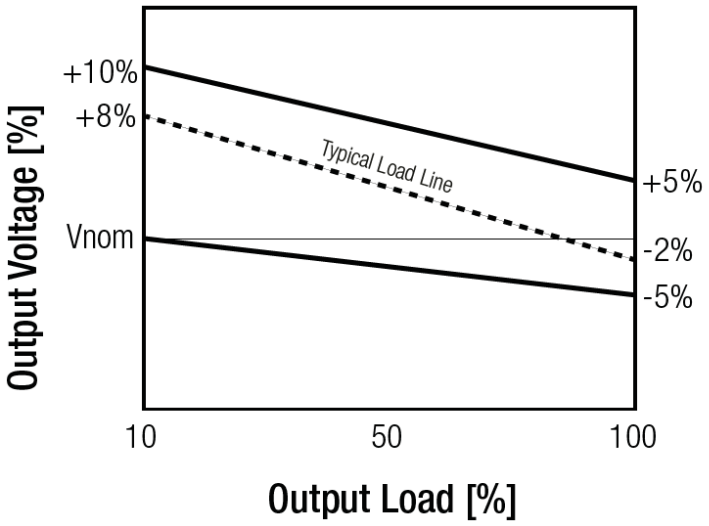
Dem Datenblatt sind folgende Kenngrößen zu entnehmen:

Parameter	Kürzel	Min	Typ	Max	Einheit	Anforderung
Versorgungsspannung	V_{DD}		5		V	5 V
Ausgangsspannung	V_{OUT}		5		V	5 V
Versorgungsstrom	I_{DD}			250	mA	
Ausgangsstrom	I_{OUT}			200	mA	>20 mA
Ruhestrom	I_Q		25	30	mA	

Tabelle 3.6: Datenblattauszug des ausgewählten DC/DC-Wandler RFMM-0505S mit den wichtigen Informationen für das Projekt [8]

Bei diesem Bauteil stellt sich die Nennspannung bei Nennlast ein. In der geplanten Schaltung fällt die Stromabnahme aber sehr viel geringer aus. Die zu versorgenden OP beziehen maximal einen Strom von 9,28 mA (siehe Abschnitt 3.2.2). was 4,6 % der Nennlast entspricht. In dem Datenblatt des RFMM-0505S findet sich dazu ein Diagramm, das das Spannungsverhalten beschreibt (siehe Abb. 3.10).

Die erwartete Auslastung taucht in der Grafik nicht auf. Verhält sich der Kurvenverlauf weiterhin linear, ist eine erzeugte Spannung 5,4 V zu erwarten.

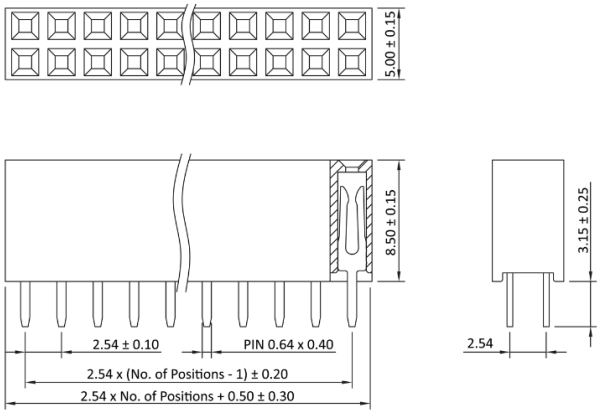


Quelle: [8, Seite 2]

Abbildung 3.10: Spannungsverhalten des DC/DC-Wandlers in Abhängigkeit der Belastung

3.2.4 Auswahl der Schnittstelle

Zuletzt muss sich für eine Verbindungsart zwischen Entwicklerboard und der Platine entschieden werden. Die Position der benötigten Pins ist auf dem Entwicklerboard nur begrenzt wählbar. Die Verbindung kann mit einem einfachen Flachbandkabel hergestellt werden. Alle benötigten Pins können aber auf die Stiftleisten gelegt werden (siehe Abb. 3.2). Deshalb werden auf der Platine passende Buchsenleisten platziert, in die das Entwicklerboard eingesteckt werden kann (siehe Abb. 3.11).



Quelle: [2, Seite 1]

Abbildung 3.11: Buchsenleisten als Platinenschnittstelle

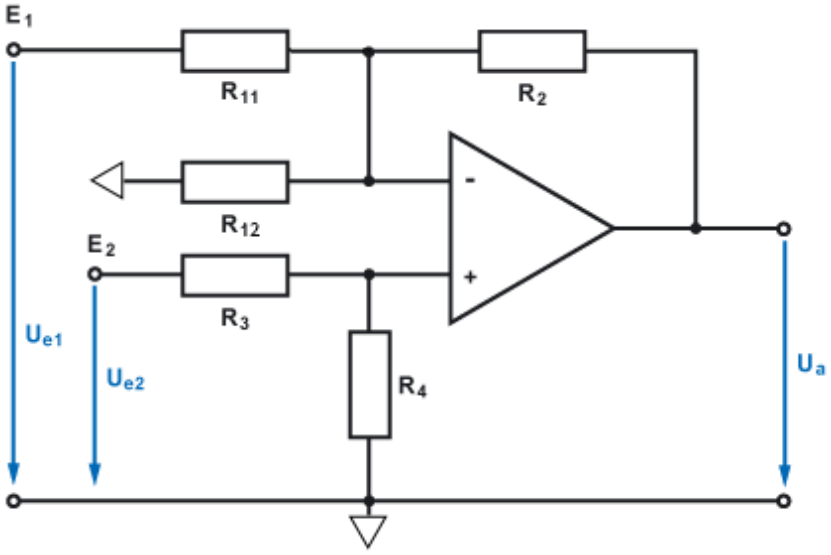
So ist eine stabile Verbindung hergestellt. Folglich dürfen nur Pins der Stiftleisten beim Softwareentwurf verwendet werden.

3.3 Dimensionierung der Verstärkerschaltung

In diesem Abschnitt wird die Verstärkerschaltung ausgelegt. Die Eingangsspannung bewegt sich zwischen 0 V und 3,3 V. Ziel ist es, diese Spannung von -5 V bis 5 V um den Ursprung schwingen zu lassen. Der OP wird als Subtrahierer beschaltet (siehe Abb. 3.3). Zur Auslegung wird die vereinfachte Gleichung (3.2) betrachtet. Aus der Formel ist erkennbar, dass die Verstärkung nach der Subtraktion stattfindet. deshalb wird als erstes der Offset eingestellt.

3.3.1 Offset

Von der Eingangsspannung U_{e2} sollen 1,65 V abgezogen werden, damit sie um das Nullpotential schwingt. Dafür werden 1,65 V an U_{e1} benötigt (siehe Gleichung (3.2)). Da es sich nicht um eine leistungsführende Verbindung handelt, wird das Spannungsniveau mit einem einfachen Spannungsteiler von dem 5 V-Potential abgegriffen. In der Schaltung wird der Widerstand R_1 mit den Widerständen R_{11} und R_{12} ersetzt (siehe Abb. 3.12). Der Betrag von R_1 in der Gleichungen (3.2), ergibt sich nun aus der Parallelschaltung von R_{11} und R_{12}. Damit sind die Gleichungen weiter gültig.



Quelle: [12] Angepasst

Abbildung 3.12: Subtrahiererschaltung mit Spannungsteiler. Der Widerstand R_1 wurd durch die Widerstände R_{11} und R_{12} ersetzt.

Die Widerstände des Spannungsteilers werden wie folgt gewählt:

$$R_{11} = 9,53\ k\Omega$$
$$R_{12} = 4,7\ k\Omega$$

Der Subtrahend ist die Spannung über R_{12}.

$$U_{R_{12}} = U_{e1} \cdot \frac{R_{12}}{R_{11} + R_{12}}$$
$$U_{R_{12}} = 5\,V \cdot \frac{4,7k\Omega}{9,53k\Omega + 4,7k\Omega}$$
$$U_{R_{12}} = \underline{1,651\,V}$$

Das Offset ist damit ausgelegt. Für die weitere Berechnung ist R_1 gleich dem Parallelwiderstand aus R_{11} und R_{12}.

$$R_1 = R_{11}||R_{12}$$
$$R_1 = (R_{11}^{-1} + R_{12}^{-1})^{-1}$$
$$R_1 = ((9,53\,k\Omega)^{-1} + (4,7\,k\Omega)^{-1})^{-1}$$
$$R_1 = \underline{3,148\,k\Omega}$$

Gemäß der Bedingungen an die Gleichung (3.2), ist R_3 gleich dem Widerstand R_1 zu wählen. Für den Widerstand R_3 wird ein 3,16 kΩ Widerstand benutzt.

Durch den Spannungsteiler kommt es außerdem zu einer dauerhaften Strombelastung I_{ST}.

$$I_{ST} = \frac{U}{R_{11} + R_{12}}$$
$$I_{ST} = \frac{5\,V}{9,53\,k\Omega + 4,7\,k\Omega}$$
$$I_{ST} = \underline{351,37\,\mu A}$$

3.3.2 Verstärkung

Der Spannungsbereich muss nun von -1,65 V bis 1,65 V auf -5 V bis 5 V verstärkt werden. Nach der Gleichung (3.2) ergibt sich die Verstärkung aus dem Widerstandsver-

hältnis $\frac{R_2}{R_1}$. Die benötigte Verstärkung A errechnet sich aus dem Verhältnis der Amplituden.

$$A = \frac{\hat{U}_a}{\hat{U}_{e2}}$$
$$A = \frac{5\,V}{1,65\,V}$$
$$A = \underline{3,03}$$

Der Widerstand R_2 beträgt damit:

$$R_2 = A \cdot R_1$$
$$R_2 = 3,03 \cdot 3,148\,k\Omega$$
$$R_2 = 9,538\,k\Omega \approx \underline{9,53\,k\Omega}$$

Für R_2 wird ein Widerstand mit 9,53 kΩ verbaut. Gemäß der Gleichungsbedingung (3.2) ergibt sich der Widerstandswert für R_4.

$$R_4 = R_2$$

Damit ist die Verstärkerschaltung ausgelegt. Die Widerstände wurden mit einer Toleranz von 1 % eingekauft. Um die maximale Abweichung für die Schwingung zu bewerten, werden die Amplitude und der bleibende Gleichanteil mit den ungünstigsten Widerstandsverhältnissen berechnet. Ausschlaggebend dabei sind die Verhältnisse von R_{11} zu R_{12} und R_1 zu R_2. In Tabelle 3.7 sind die Auswirkungen von mehreren Toleranzen bei maximaler Abweichung aufgelistet.

Toleranz 1 %	**Min**	**Typ**	**Max**	**Fehler bezogen auf 5 V**
Gleichanteil	-0,071 V	-0,004 V	0,062 V	1,42 %
Amplitude	4,897 V	4,996 V	5,097 V	2,06 %
Toleranz 0,5 %	**Min**	**Typ**	**Max**	**Fehler bezogen auf 5 V**
Gleichanteil	-0,038 V	-0,004 V	0,029 V	0,76 %
Amplitude	4,946 V	4,996 V	5,046 V	1,08 %
Toleranz 0,25 %	**Min**	**Typ**	**Max**	**Fehler bezogen auf 5 V**
Gleichanteil	-0,021 V	-0,004 V	0,012 V	0,42 %
Amplitude	4,970 V	4,996 V	5,021 V	0,60 %

Tabelle 3.7: Auswirkung der maximalen Widerstandsabweichungen auf die Funktion der Schaltung. Ab einer Toleranz von 0,25 % liegt der maximale Fehler unter 1 %

Da sowohl die Amplitude, als auch die Offset-Spannung von den Widerständen R_{11} und R_{12} abhängen, können die beiden ungünstigsten Minimal- und Maximalwerte nicht zusammen auftreten. Um den entstandenen Fehler unter 1 % zu halten, müssen Widerstände mit einer Toleranz von maximal 0,25 % verbaut werden.

3.4 Gesamter Schaltungsaufbau

Das Platinenlayout ist nicht Teil der Bachelorarbeit. Es wird lediglich der Schaltplan erstellt. Die Firma Stucke Elektronik entwirft daraufhin das Layout. Die Abbildung 3.13 zeigt den Schaltplan für das Platinenlayout. Es wurden Kondensatoren zur Verbesserung der EMV Eigenschaften eingefügt. Aus den Datenblättern der Bauteile wurden dafür die empfohlenen Kapazitäten und Beschaltungen entnommen.

Die maximale Strombelastung I_L für das Entwicklerboard wird berechnet.

$$
\begin{aligned}
I_L &= I_{DAC} + I_{OP} + I_Q + I_{ST} \\
I_L &= 2,80\,mA + 9,28\,mA + 30\,mA + 351,35\,\mu A \\
I_L &= \underline{42,43\,mA}
\end{aligned}
$$

Mit 42,43 mA wird die Versorgung bis zu 14 % belastet. Hierbei wurde von den Maximalwerten ausgegangen. Die Stromabnahme im Normalbetrieb wird geringer ausfallen.

Die Kosten für eine Platine sind exemplarisch in Tabelle 3.8 aufgestellt.

Pos	Komponente	Bezeichnung	Anzahl	Preis	Kosten
1	DAC	MCP4921T-E/SN	8 Stk	0,09 €	0,72 €
2	OP	TLV2372IDR	4 Stk	1,52 €	6,08 €
3	DC/DC-Wandler	RFM0505	1 Stk	1,77 €	1,77 €
4	Widerstand 3,16 kΩ	ERJ-8ENF3161V	8 Stk	0,09 €	0,72 €
5	Widerstand 9,53 kΩ	ERJ-8ENF9531V	24 Stk	0,04 €	0,40 €
6	Widerstand 4,7 kΩ	CRGCQ1206F4K7	8 Stk	0,09 €	0,72 €
7	Kondensator 1 µF	C1206C105M3RACTU	16 Stk	0,071 €	1,14 €
8	Kondensator 10 µF	TMK316BJ106KL-T	2 Stk	0,27 €	0,54 €
9	Buchsenleiste 2x8	8 BLG2X8	1 Stk	0,30 €	0,30 €
10	Buchsenleiste 2x10	10 BLG2X10	1 Stk	0,38 €	0,38 €
11	Buchsenleiste 2x15	15 BLG2X15	1 Stk	0,57 €	0,57 €
12	Buchsenleiste 2x17	17 BLG2X17	1 Stk	0,64 €	0,64 €
13	Platine		1 Stk	22,46 €	22,46 €
				Gesamtkosten:	**36,44 €**

Tabelle 3.8: Gesamtkosten einer Platine, Bestellung bei Firma Mouser und Firma Conrad [4][1] am 13.02.2020

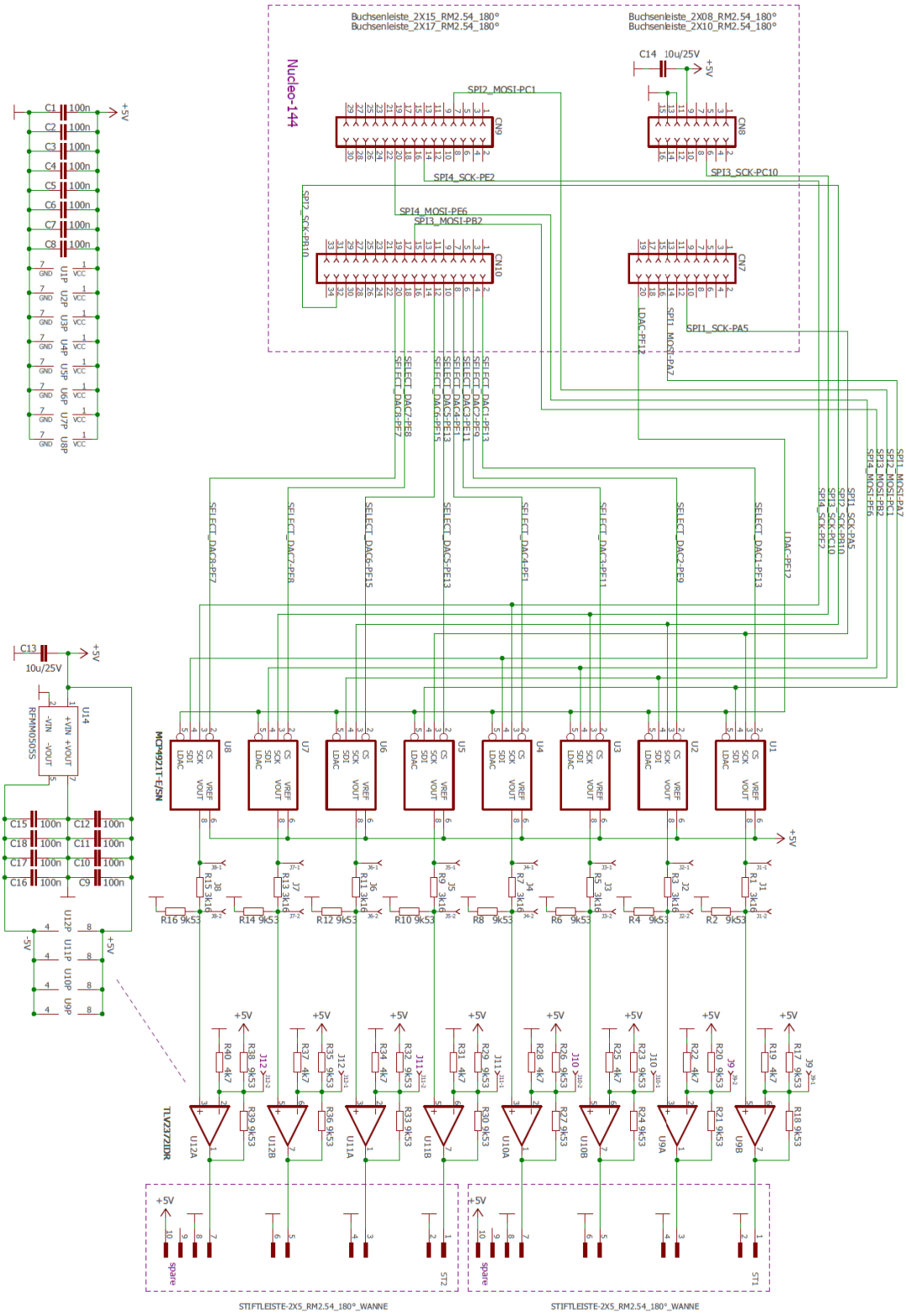


Abbildung 3.13: Schaltplan für das Platinenlayout

4 Simulation und Vorversuche

Vor dem Aufbau der Schaltung werden Vorversuche durchgeführt, um frühzeitig Fehler der Konzeption zu erkennen und anzupassen. In diesem Kapitel wird die ausgelegte Operationsverstärkerschaltung simuliert und die SPI-Ansteuerung zu den DAC getestet.

4.1 Simulation

Mit dem Programm LTSPICE wird die Auslegung aus Abschnitt 3.1.6 simuliert (siehe Abb. 4.1). Das Simulationsergebnis ist in Abbildung 4.2 dargestellt.

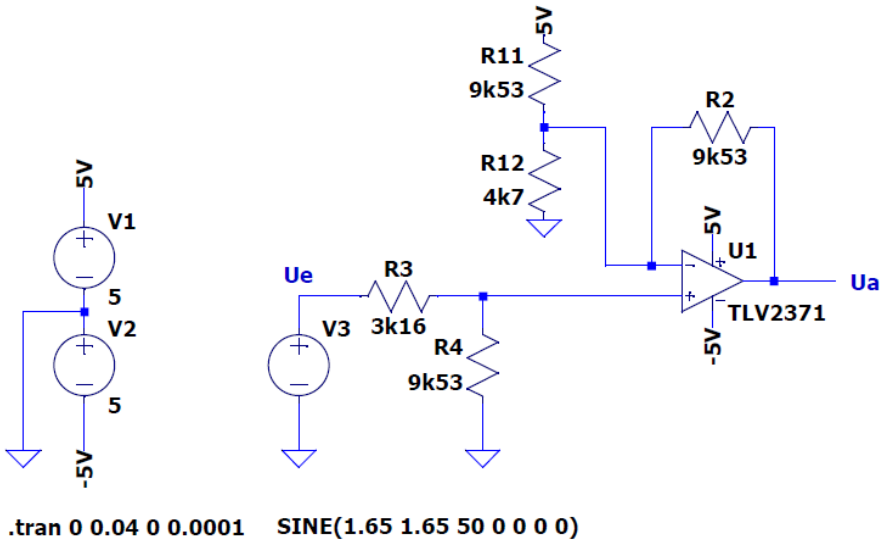


Abbildung 4.1: Aufbau der Verstärkerschaltung in dem Programm LTSPICE

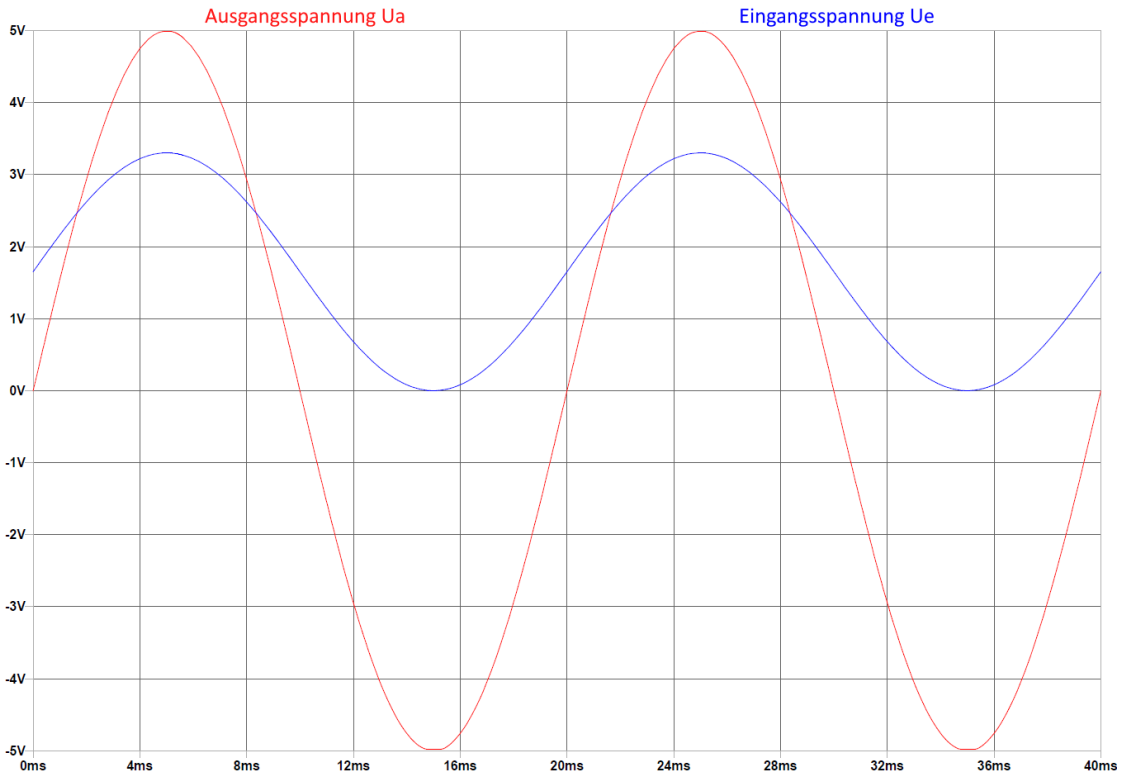


Abbildung 4.2: Simulationergebnis der Verstärkerschaltung mit dem Programm LTSPICE

Wie erwartet, wird eine Ausgangsschwingung von -5 V bis 5 V erzeugt. Allerdings wird bei der negativen Halbwelle die Begrenzung erreicht (siehe Abb. 4.2). Grund dafür ist der Spannungsteiler. Dieser ergibt mit 1,651 V eine geringfügig höhere Spannung als beabsichtigt. Bei der Umsetzung wird die Begrenzung nicht erreicht werden, da ein etwas größerer Aussteuerungsbereich erwartet wird (siehe Abschnitt 3.2.3). Des Weiteren soll die Schwingung die Eingangspannung eines Verstärkers sein. Für diesen ist ein Eingangskondensator geplant, der einen bestehenden Gleichanteil herausfiltert.

4.2 Kommunikation über SPI

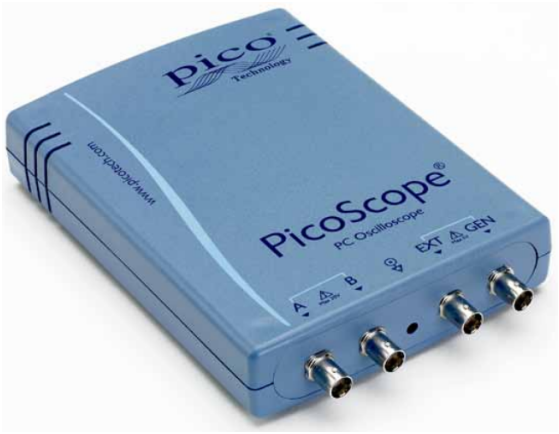
In der Konzeptionsphase des Projektes wurde bereits versucht, einen DAC anzusteuern. Zuerst wurde die Treiberfunktion von der IDE verwendet. Jedoch ist die Treiberfunktion so konzipiert, dass möglichst viele Fehler vom Anwender erkannt werden. Dadurch entsteht bei jeder Übertragung eine Verzögerung. Sie eignet sich gut für größere Datenmengen. Bei diesem Projekt handelt es sich aber um viele Übertragungen mit kleinen Datenmengen.

Es wurde eine eigene Funktion geschrieben (siehe Abb. 4.3). Diese wählt den DAC aus. Die Register des SPI-Controllers werden direkt beschrieben. Sicherheitsprüfungen werden nicht durchgeführt. Es wird auf das EOT-Flag gewartet und der DAC wieder abgewählt. Anschließend wird der Pin $\overline{\mathrm{LDAC}}$ des DAC geschaltet.

```
while (1)
{
    //-------------- select DAC 1
    DAC1_GPIO_Port->BSRR = (uint32_t)DAC1_Pin << GPIO_NUMBER;   // dac_1 css pin Low

    //-------------- start transmition
    hspi1.Instance->TXDR = DAC_CONTROL_BITS+2048;   // Fill FIFO
    hspi1.Instance->CR1 |= SPI_CR1_CSTART;          // master start

    //-------------- wait for EOT
    while(!(hspi1.Instance->SR & SPI_FLAG_EOT));

    //-------------- flag reset
    hspi1.Instance->IFCR |= (0x7fc);

    //-------------- deselect DAC 1
    DAC1_GPIO_Port->BSRR = DAC1_Pin; // dac_1 css pin High

    //-------------- toggle LDAC
    LDAC_GPIO_Port->BSRR = (uint32_t)LDAC_Pin << GPIO_NUMBER;   // LDAC Pin Low
    G_hControl.i = 0;                   // set back wave index
    LDAC_GPIO_Port->BSRR = LDAC_Pin;    // LDAC Pin High
}
```

Abbildung 4.3: Testprogramm für die Ansteuerung eines DAC mit dem Entwicklerboard

Die erreichte Übertragungsgeschwindigkeit wird mit einem PC-Oszilloskop gemessen (siehe Abb. 4.4). Um die Durchlaufzeit der Schleife zu erfassen, wurde eine Messung an dem Pin $\overline{\mathrm{LDAC}}$ durchgeführt (siehe Abb. 4.5). Auch der Pin SCK wurde aufgenommen. So lässt sich erkennen, welchen Anteil die Daten bei der Übertragung haben.



Quelle: [6]

Abbildung 4.4: Das verwendete PC-Oszilloskop für die Messung, Typ: PICOSOPE 3206B

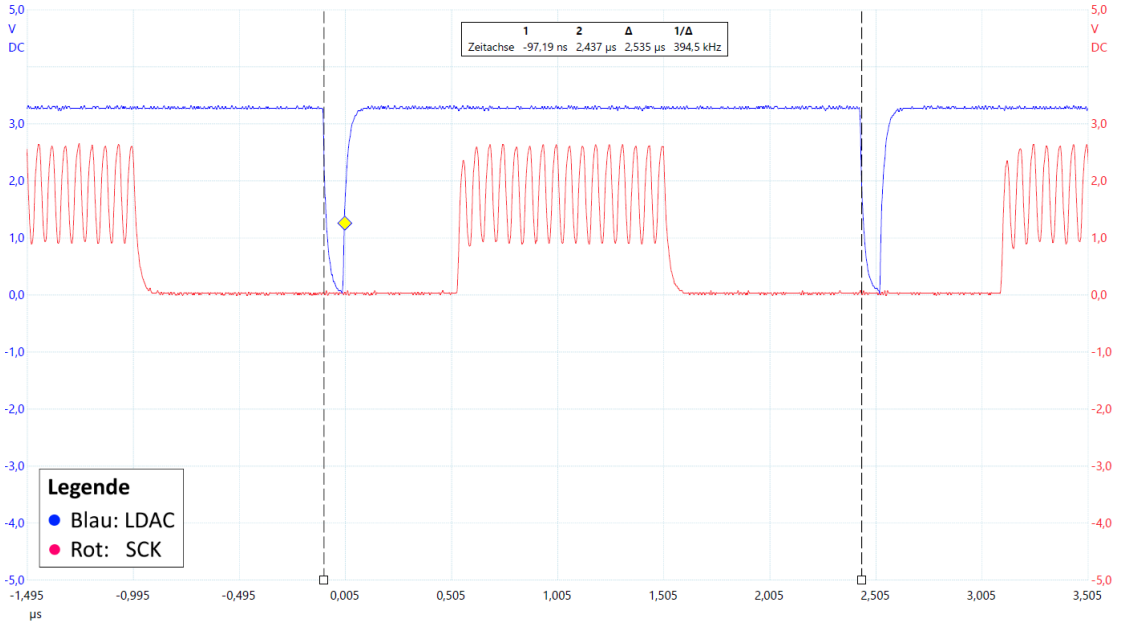


Abbildung 4.5: Messung der Übertragungsdauer an einen DAC. Spannungsverlauf am PIN LDAC in blau, Spannungsverlauf an dem Pin SCK in rot. Zeitdifferenz zwischen den Markierungen: 2,535 μs

Ein Schleifendurchlauf benötigt eine Zeit T_S von 2,535 μs. Die erreichbare Ausgabefrequenz für acht DAC wird berechnet.

$$F_{MAX} = (T_S \cdot Anzahl_{DAC} \cdot Werte_{Pro\ Periode})^{-1}$$
$$F_{MAX} = (2,535\,\mu s \cdot 8 \cdot 512)^{-1}$$
$$F_{MAX} = \underline{96,31\,Hz}$$

Es kann eine Ausgabefrequenz von 96,31 Hz erreicht werden. Diese Vorgehensweise erfüllt die Anforderungen. Dennoch wurde entschieden, die Übertragung auf vier SPI-Kanäle aufzuteilen, weil so weitere Anweisungen in der Schleife hinzugefügt werden können, ohne das Erreichen der geforderten Ausgabefrequenz zu gefährden. Die Testfunktion wurde angepasst (siehe Abb. 4.6).

```
while (1)
{
    //-------------- select DAC 1-4
    DAC1_GPIO_Port->BSRR = (uint32_t)DAC1_Pin << GPIO_NUMBER;   // dac_1 css pin Low
    DAC2_GPIO_Port->BSRR = (uint32_t)DAC2_Pin << GPIO_NUMBER;   // dac_2 css pin Low
    DAC3_GPIO_Port->BSRR = (uint32_t)DAC3_Pin << GPIO_NUMBER;   // dac_3 css pin Low
    DAC4_GPIO_Port->BSRR = (uint32_t)DAC4_Pin << GPIO_NUMBER;   // dac_4 css pin Low

    //-------------- start transmition
    hspi1.Instance->TXDR = DAC_CONTROL_BITS+2048;   // Fill FIFO
    hspi1.Instance->CR1 |= SPI_CR1_CSTART;          // master start
    hspi5.Instance->TXDR = DAC_CONTROL_BITS+2048;   // Fill FIFO
    hspi5.Instance->CR1 |= SPI_CR1_CSTART;          // master start
    hspi3.Instance->TXDR = DAC_CONTROL_BITS+2048;   // Fill FIFO
    hspi3.Instance->CR1 |= SPI_CR1_CSTART;          // master start
    hspi4.Instance->TXDR = DAC_CONTROL_BITS+2048;   // Fill FIFO
    hspi4.Instance->CR1 |= SPI_CR1_CSTART;          // master start

    //-------------- wait for EOT
    while(!(hspi1.Instance->SR & SPI_FLAG_EOT));
    while(!(hspi5.Instance->SR & SPI_FLAG_EOT));
    while(!(hspi3.Instance->SR & SPI_FLAG_EOT));
    while(!(hspi4.Instance->SR & SPI_FLAG_EOT));

    //------------- flag reset
    hspi1.Instance->IFCR |= (0x7fc);
    hspi5.Instance->IFCR |= (0x7fc);
    hspi3.Instance->IFCR |= (0x7fc);
    hspi4.Instance->IFCR |= (0x7fc);

    //-------------- deselect DAC 1-4
    DAC1_GPIO_Port->BSRR = DAC1_Pin; // dac_1 css pin High
    DAC2_GPIO_Port->BSRR = DAC2_Pin; // dac_2 css pin High
    DAC3_GPIO_Port->BSRR = DAC3_Pin; // dac_3 css pin High
    DAC4_GPIO_Port->BSRR = DAC4_Pin; // dac_4 css pin High

    //-------------- toggle LDAC
    LDAC_GPIO_Port->BSRR = (uint32_t)LDAC_Pin << GPIO_NUMBER;   // LDAC Pin Low
    G_hControl.i = 0;                   // set back wave index
    LDAC_GPIO_Port->BSRR = LDAC_Pin;    // LDAC Pin High
}
```

Abbildung 4.6: Testprogramm für die Ansteuerung von vier DAC mit dem Entwicklerboard

Vier DAC werden ausgewählt. Die FIFOs der vier SPI-Kanäle werden beschrieben. Die SPI-Controller senden die Daten zeitgleich an die DAC. Die CPU wartet auf die vier EOT-Flags. Anschließend werden die DAC wieder abgewählt und der Pin $\overline{\mathrm{LDAC}}$ wird geschaltet. Die Messung wird mit dieser Methode wiederholt (siehe Abb. 4.7).

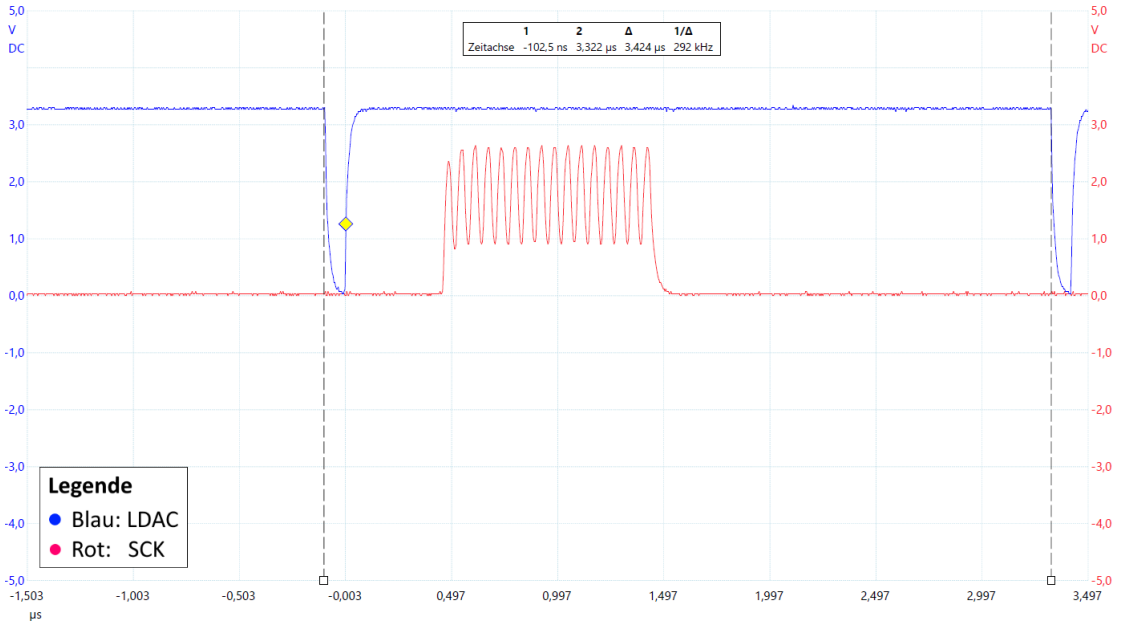


Abbildung 4.7: Messung der Übertragungsdauer mit vier SPI-Kanälen. Spannungsverlauf an dem PIN LDAC in blau, Spannungsverlauf an dem Pin SCK eines DAC in rot. Zeitdifferenz zwischen den Markierungen: 3,424 μs

Der Schleifendurchlauf benötigt eine Zeit T_S von 3,424 μs. Die erreichbare Ausgabefrequenz wird wieder berechnet.

$$F_{MAX} = \frac{1}{T_S \cdot Anzahl_{DAC\ Pro\ Kanal} \cdot Werte_{Pro\ Periode}}$$

$$F_{MAX} = \frac{1}{3,424\,\mu s \cdot 2 \cdot 512}$$

$$F_{MAX} = \underline{285,21\,Hz}$$

So kann die Kurvenform mit einer Frequenz von 285,21 Hz ausgegeben werden. Im Vergleich zum einkanaligen Betrieb hat sich die Frequenz fast verdreifacht.

5 Entwicklung und Aufbau

In diesem Kapitel wird die Umsetzung des Konzeptes beschrieben. Die fertiggestellte Platine wird gezeigt und danach das entwickelte Programm beschrieben.

5.1 Platine

Die Platine wird bei Beta LAYOUT GmbH bestellt. Abbildung 5.1 zeigt das gelieferte Produkt. Das Layout wurde auf einer zweilagigen Platine erstellt. Das von der Firma Stucke Elektronik GmbH erstellte Platinenlayout befindet sich im Anhang B. Auf den Folgeseiten sind die Schritte des Herstellungsprozesses sichtbar.

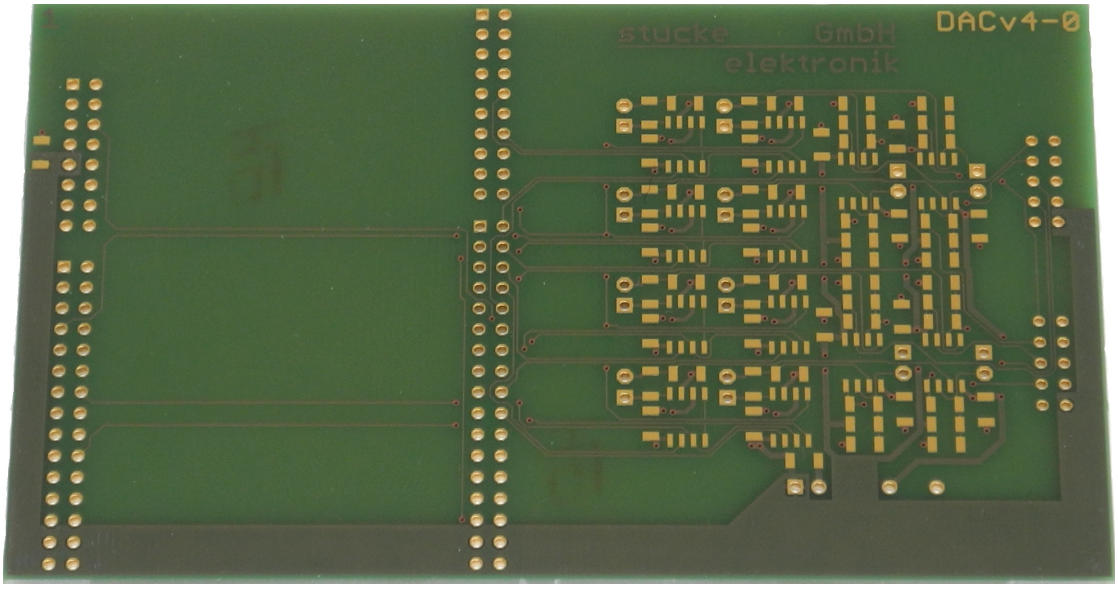


Abbildung 5.1: Fotografie der gelieferten Platine.

Die Bauteile wurden auf die Platine gelötet. Alle Kontakte und Lötstellen wurden überprüft, dabei fiel ein Designfehler auf. Der Schaltplan, der an den Mitarbeiter für das Platinenlayout gegeben wurde, enthielt zwei Fehler.

Die Pins der DAC V_{DD} und V_{REV} wurden mit 5 V anstatt 3,3 V versorgt.

Zweitens waren bei einem SPI-Kanal die Pins am Entwicklerboard verwechselt. Grund dafür war ein Tippfehler.

Da es sich um einen Prototyp für einen Teil eines Funktionsgenerators handelt, wurde entschieden, weiter mit der fehlerhaften Platine zu arbeiten. So konnte eine Verzögerung durch lange Lieferzeiten umgangen werden. Die Versorgungsspannung für die DAC von 3,3 V werden mit einem Draht vom Entwicklerboard abgegriffen. Die vertauschten Pins werden ebenfalls mit einer zusätzlichen Verbindung korrigiert.

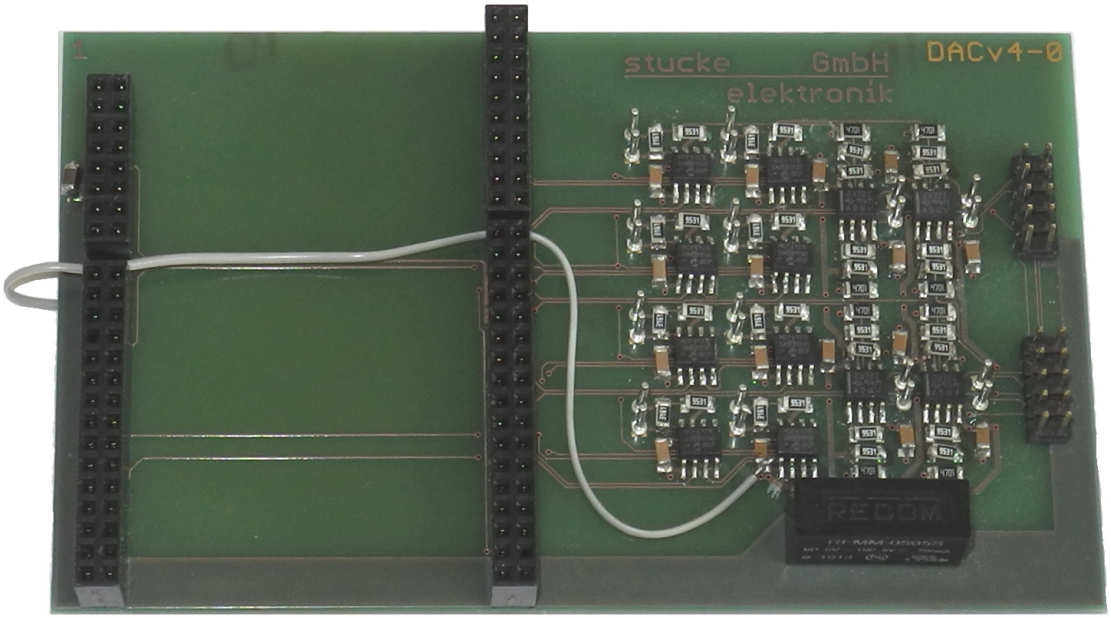


Abbildung 5.2: Fotografie der bestückten und korrigierten Platine. Steckverbindung für das Entwicklerboard auf der linken Seite, Schaltung der DAC und OP auf der rechten Seite.

5.2 Softwareentwicklung

In diesem Abschnitt wird die Funktionsweise des entwickelten Steuerprogrammes beschrieben. Für die Software wurde ein Ablaufdiagramm entworfen, das die Struktur für die Entwicklung vorgibt. In Abbildung 5.3 ist der Entwurf für die Softwarestruktur dargestellt.

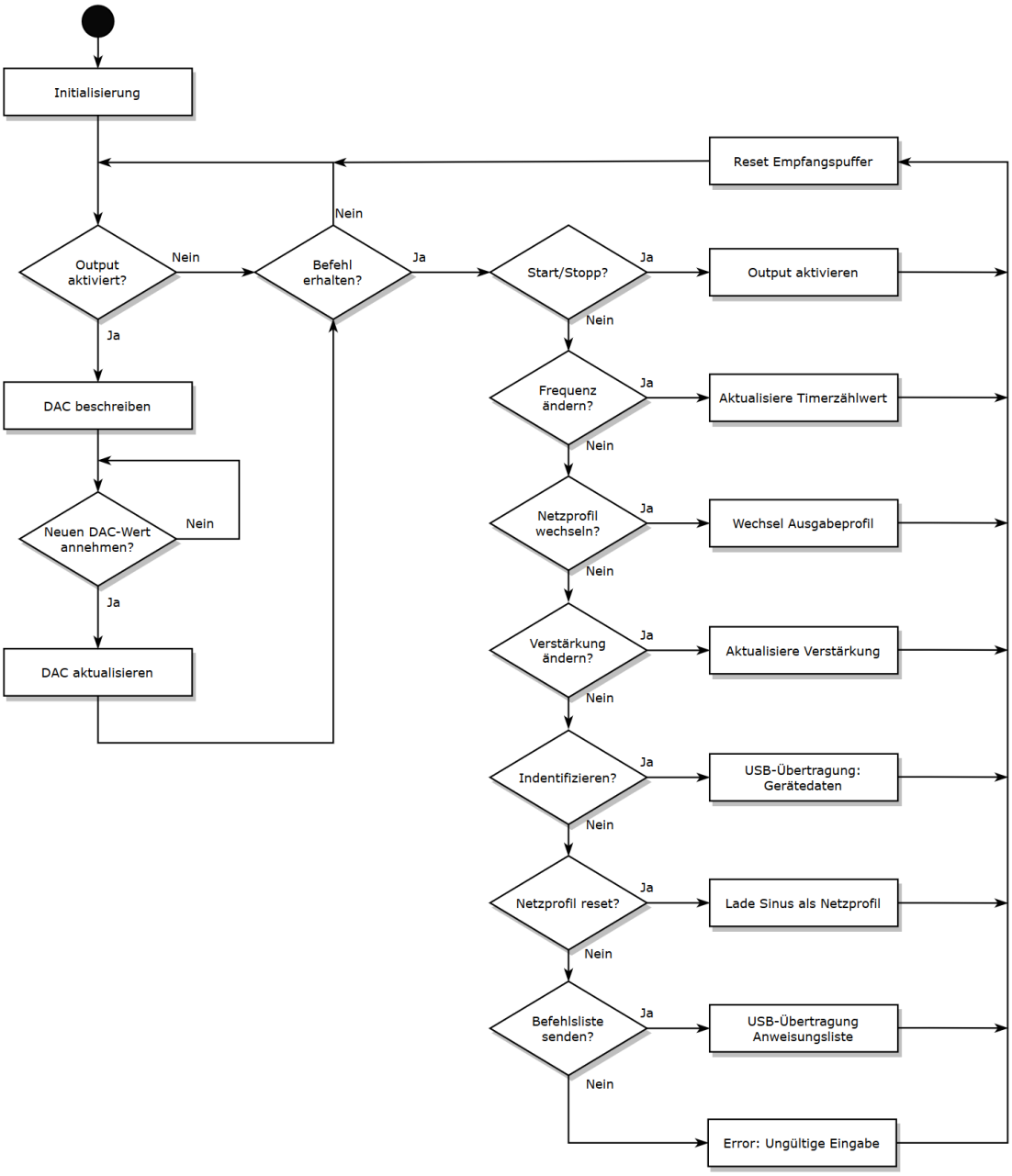


Abbildung 5.3: Ablaufschema der Grundfunktionen auf oberer Ebene

Zuerst wird das System initialisiert und in einen definierten Zustand gebracht. Der Systemzustand nach dem Start soll folgende Eigenschaften haben:

- Die Ausgangsspannung ist gleich 0 V
- Die Ausgabe des Netzprofils ist deaktiviert
- Die Ausgabefrequenz ist 50 Hz
- Die Daten eines Netzprofils mit idealisierten Parametern sind geladen

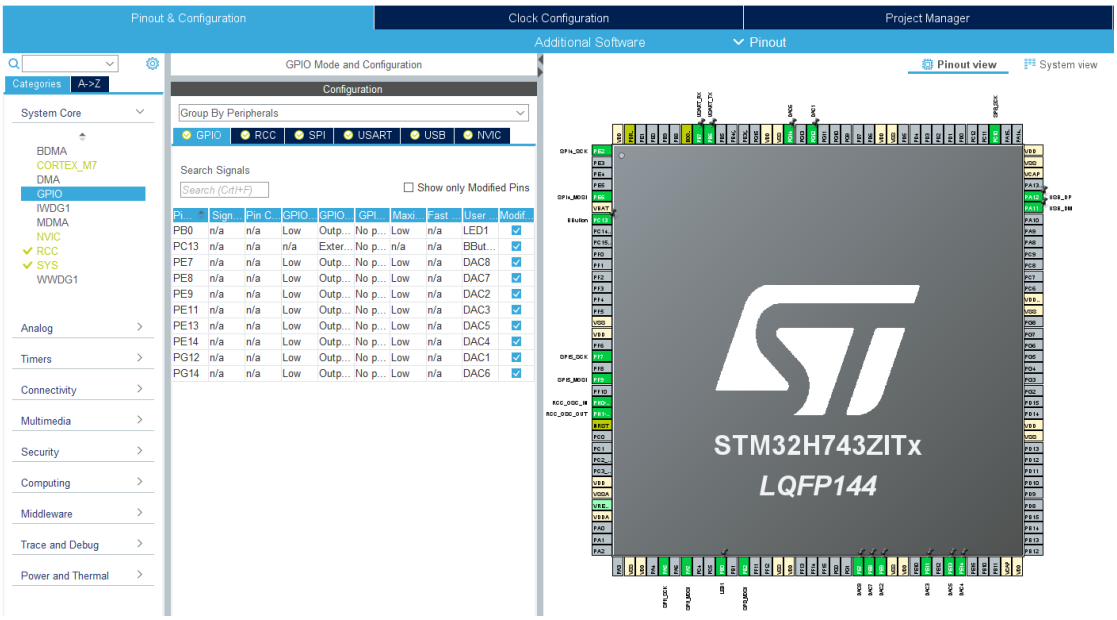
Danach befindet sich das Programm in der Hauptschleife. Um eine möglichst hohe Wiederholungsrate zu erreichen, soll die Schleife mit möglichst wenig Anweisungen programmiert werden. Mit einer if-Schleife soll kontrolliert werden, ob die Kurvenform ausgegeben werden soll. Mit einer weiteren soll kontrolliert werden, ob ein Befehl über USB erhalten wurde.

Ist die Ausgabe aktiviert, werden die DAC per SPI beschrieben. Dann wird auf den periodischen Timer gewartet und die Ausgangsspannung mit dem Pin $\overline{\mathrm{LDAC}}$ aktualisiert.

Wurde eine Nachricht über USB erhalten, wird der Inhalt mit der Befehlsliste verglichen. Der Befehl wird bearbeitet oder eine Fehlermeldung zurückgegeben.

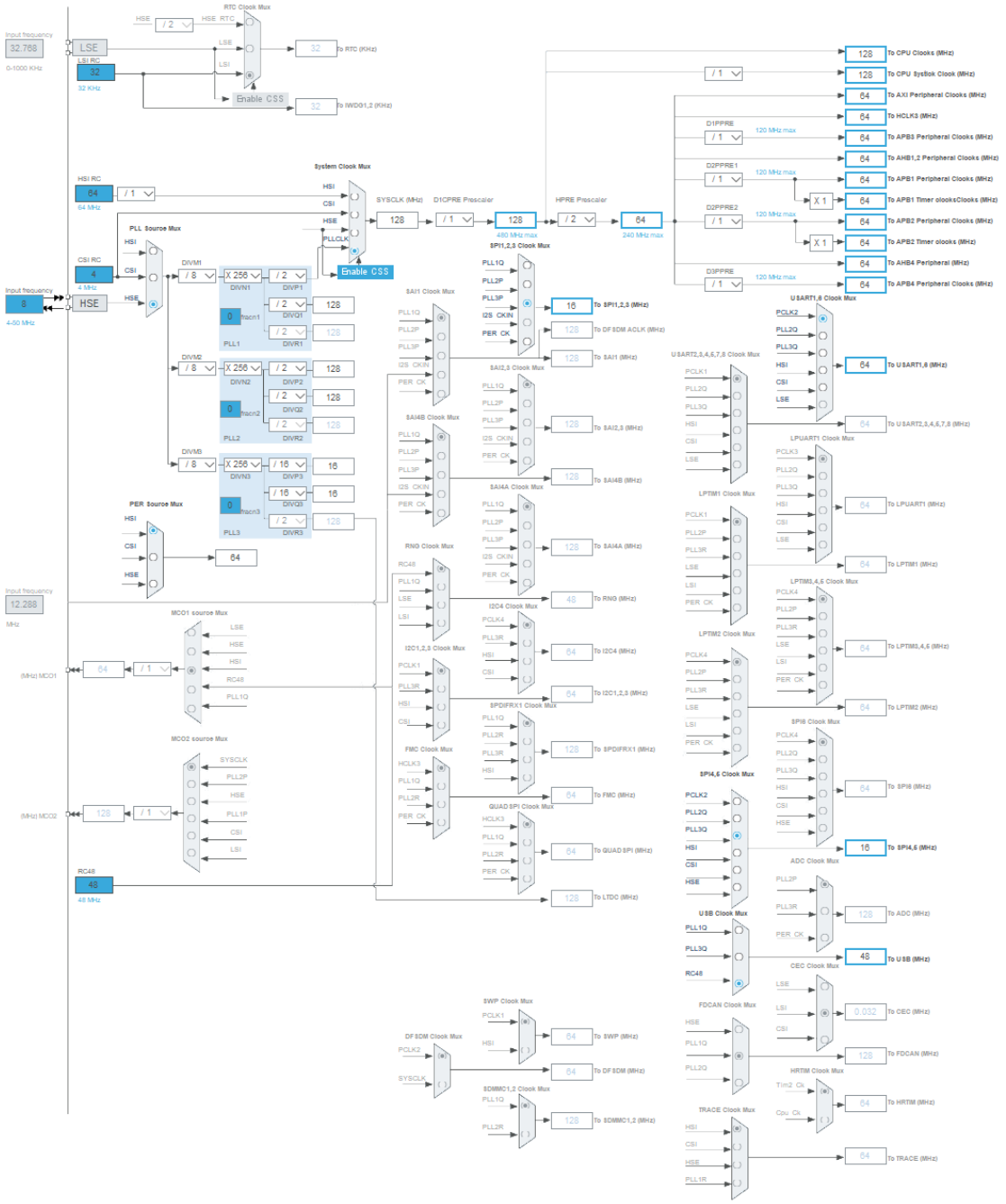
5.2.1 Projekterstellung

Mit der IDE werden schon bei der Projekterstellung die Takt- und Peripheriekonfigurationen vorgenommen. Als erstes werden die Oszillatoren ausgewählt. Der Mikrocontroller verfügt über einen internen Phasenschieber-Oszillator mit RC-Gliedern [10, Seite 338]. Dieser ist zu ungenau für dieses Projekt. Auf dem Entwicklerboard ist auch ein Quarz mit 8 MHz verbaut [11, Seite 25]. Dieser wird als externer Oszillator genutzt. Für hohe Genauigkeitsanforderungen empfiehlt der Hersteller einen externen Quarz vom Typ NX2016SA-25MHz-EXS00A-CS11321 zu verwenden. Für die USB-Schnittstelle ist ein eigener Quarz mit einer Frequenz von 48 MHz vorhanden. In einem Diagramm werden Multiplikatoren und Divisoren eingestellt, um die Systeme mit den gewünschten Takten zu versorgen (siehe Abb. 5.5). Der Systemtakt wird auf 128 MHz eingestellt, die Takte für die Peripherien und Timer werden auf 64 Mhz eingestellt. Die SPI-Controller werden mit 16 MHz betrieben. In einer zweiten Ansicht werden die Pins des Mikroprozessors belegt (siehe Abb. 5.4). Timer, SPI, USB, sowie die benötigten GPIOs werden hier aktiviert.



Quelle: Screenshot der IDE STM32CubeIDE

Abbildung 5.4: Einstellung der Pinbelegung bei der Projekterstellung

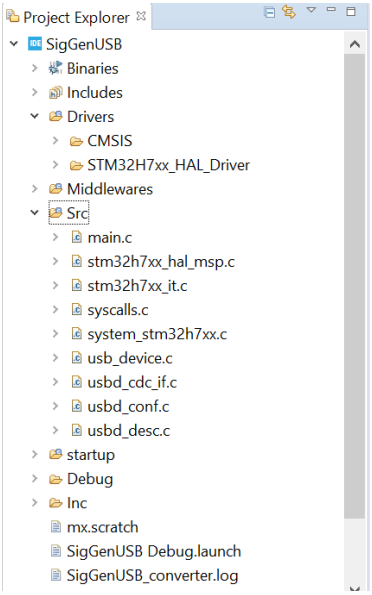


Quelle: Screenshot der IDE STM32CubeIDE

Abbildung 5.5: Taktkonfiguration bei der Projekteinstellung

Das Programm erstellt daraufhin das Projekt. Es ist bereits eine Ordnerstruktur mit Treibern der gewählten Peripherien vorhanden (siehe Abb. 5.6). In dem Ordner Inc befinden

sich Makro-Funktionen und Konstanten für eine verbesserte Lesbarkeit der Treiberfunktionen. In der Datei stm32h7xx_it.c sind Funktionen, die per Interrupt aufgerufen werden. Dort kann beispielsweise das Verhalten bei dem Auftreten eines Hardwarefehlers programmiert werden. Des Weiteren sind mehrere Dateien für die USB-Kommunikation erstellt worden. In der Datei usbd_desc.c kann die Beschreibung des USB-Gerätes inklusive der VID und der PID verändert werden. In der Datei usb_device.c befindet sich die Initialisierungsfunktion der Schnittstelle. Die Datei usbd_cdc_if.c wurde erstellt, da die USB-Klasse CDC gewählt wurde. Hier kann das Verhalten bei dem Senden und dem Empfangen von Nachrichten angepasst werden.



Quelle: Screenshot der IDE STM32CubeIDE

Abbildung 5.6: Ordnerstruktur erstellt von der IDE. Das eigene Programm wird in der Datei main.c geschrieben

Die Konfigurationen, die bei der Projekterstellung gewählt wurden, werden durch automatisch erstellte Initialfunktionen in main.c ausgeführt und können noch verändert werden. Diese sind mit de Prefix MX in ihrer Bezeichnung gekennzeichnet (siehe Abb. 5.7)

```
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_SPI1_Init(void);
static void MX_SPI3_Init(void);
static void MX_SPI4_Init(void);
static void MX_SPI5_Init(void);
static void MX_TIM1_Init(void);
```

Abbildung 5.7: Automatisch erstellte Konfigurationsfunktionen der IDE. Sie führen die Einstellung bei der Projekterstellung aus.

5.2.2 Datenstruktur

Die für die Systemsteuerung wichtigen Variablen werden global erstellt. Sie werden in drei Strukturen gespeichert.

- Daten der Netzprofile
- Parameter für die Ausgabe
- Variablen für den Empfang von Daten

Die Daten der Netzprofile werden in einem globalen Array aus 512 Elementen gespeichert.

```
struct wave wave_matrix[512];
```

Abbildung 5.8: Globales Array für die Daten der Netzprofile, bestehend aus Elementen vom Typ wave

Um viele Indizes bei Schleifen zu vermeiden, besteht jedes Element aus einer Struktur (siehe Abb. 5.9). Die Struktur repräsentiert einen Datenpunkt jeder Spannungs- und Stromkurve von zwei Netzprofilen.

```
struct wave
{
    uint16_t out_11;    // Profile 1, U_L1
    uint16_t out_12;    // Profile 1, U_L2
    uint16_t out_13;    // Profile 1, U_L3
    uint16_t out_14;    // Profile 1, U_E
    uint16_t out_15;    // Profile 1, I_L1
    uint16_t out_16;    // Profile 1, I_L2
    uint16_t out_17;    // Profile 1, I_L3
    uint16_t out_18;    // Profile 1, I_E
    uint16_t out_21;    // Profile 2, U_L1
    uint16_t out_22;    // Profile 2, U_L2
    uint16_t out_23;    // Profile 2, U_L3
    uint16_t out_24;    // Profile 2, U_E
    uint16_t out_25;    // Profile 2, I_L1
    uint16_t out_26;    // Profile 2, I_L2
    uint16_t out_27;    // Profile 2, I_L3
    uint16_t out_28;    // Profile 2, I_E
};
```

Abbildung 5.9: Struktur für die Schwingungsdaten. Ein Element der Struktur repräsentiert einen Datenpunkt von zwei Netzprofilen.

Alle relevanten Variablen für die Schwingungsausgabe sind in einer Struktur enthalten (siehe Abb. 5.10). Parameter, wie z.B. Frequenz oder die aktuelle Kurvenposition sind hier gespeichert.

```
struct global_control_handler
{
    uint16_t frequency;          // 100-2000 frequency
    uint16_t wave_source;        // 0 = profile 1, 1 = profile 2
    uint16_t i;                  // 0-511 array index
    uint8_t enable_output;       // 0 = disable, 1= enable | DAC output
    uint8_t Command;             // 0 = no command received, 1 = command received
    uint16_t minF;               // minimum frequency
    uint16_t maxF;               // maximum frequency
    float gain;                  // 0.000-9.999 gain factor
};
```

Abbildung 5.10: Struktur für die Kontrollvariablen. Die Ausgabe richtet sich nach den hier gesetzten Parametern.

Auch die Variablen für eine Netzdatei sind in einer Struktur zusammengefasst (siehe Abb. ??). Mit über 42 kByte belegt sie den größten Speicher. Über USB empfangene Inhalte werden hier im Format ASCII gespeichert. Das Array receive_array wurde groß genug gewählt, um eine komplette Datei zu speichern, bevor die Inhalte in das Array wave_matrix übernommen werden müssen.

```
struct global_file_handler
{
    uint8_t receive_array[42000];       // Input buffer for files
    uint16_t file_index;                // Index for receive_array
    uint8_t Received_Data_Buffer[12];   // Input buffer for commands
    uint32_t Length;                    // Length of current Package
    uint16_t DataLength;                // Data length of received file
    uint16_t DataLength_ofActiveFile;   // Data length of active file
    uint8_t OnGoing_FileTransfer;       // 0 = no file transfer, 1 = file transfer
    int HeaderLength;                   // Header Length of received file
    int HeaderLength_ofActiveFile;      // Header Length of active file
    uint64_t tick;                      // ticks since last package reveived
    uint64_t timeout;                   // ticks to file transfer timeout
};
```

Abbildung 5.11: Struktur der dateirelevanten Variablen

5.2.3 Initialisierung

In der main.c werden die Takt- und Peripheriekonfigurationen der Projekterstellung ausgeführt. Der Zählwert des periodischen Timers wird so gesetzt, dass sich eine Frequenz

von 50 Hz ergibt. Der Wert wird mit dem Timertakt von 64 MHz als Gleitkommazahl berechnet (siehe Abb. 5.12). Das Ergebnis wird gerundet und als int Wert übernommen.

```
// calulate counting value
float period = TIM1_CLK / 512.0 / 50.0;
if (period - (int)period >= 0.5)
{
    period++; // round up
}
```

Abbildung 5.12: Zählwertberechnung des periodischen Timers bei der Initialisierung

Die Ausgangsspannung wird auf 0 V gesetzt, indem der Nullwert 2048 an die DAC gesendet wird. Um die Ausgabe der Schwingungsdaten zu deaktivieren, wird die Variable enable_output auf Null gesetzt. Das Array der Netzdaten wird mit zwei Netzprofilen befüllt. Das erste enthält die Daten einer Sinusfunktion, das zweite die Daten einer Sägezahnfunktion.

5.2.4 Hauptschleife

Nach dem Start befindet sich das Programm in einer dauerhaften Schleife. Der Ablauf folgt dem zuvor erstellten Ablaufdiagramm. Abbildung 5.13 zeigt den Programmausschnitt.

```
while (1)
{
    /****************************** Control DAC output ***********************************/
    if (G_hControl.enable_output)
    {
        //------------------ Transmit DAC 1-4
        select_DAC_1_to_4();            // CSS low
        write_SPI_DAC_1_to_4();         // SPI data transmit
        wait_SPI_transmit();            // wait for SPI EOT flags
        deselect_DAC_1_to_4();          // CSS high

        //------------------ Transmit DAC 5-8
        select_DAC_5_to_8();            // CSS low
        write_SPI_DAC_5_to_8();         // SPI data transmit
        wait_SPI_transmit();            // wait for SPI EOT flags
        deselect_DAC_5_to_8();          // CSS high

        //------------------ update DAC Output
        while (!(htim1.Instance->SR&TIM_SR_UIF));                   // wait for timer
        LDAC_GPIO_Port->BSRR = (uint32_t)LDAC_Pin << GPIO_NUMBER;   // LDAC Pin Low
        htim1.Instance->SR &= ~TIM_SR_UIF;                          // reset flag
        if ( G_hControl.i < 512 ) G_hControl.i++;                   // increment wave index
        LDAC_GPIO_Port->BSRR = LDAC_Pin;                            // LDAC Pin High
        if ( G_hControl.i >= 512) G_hControl.i = 0;                 // reset wave index
    }

    /*********************** check for file transfer timeout ******************************/
    else if(G_hFile.OnGoing_FileTransfer) timeout_check();

    /************** react to USB commands - received in usb_rXcallback() ******************/
    if (G_hControl.Command) USB_command_handler();
}
```

Abbildung 5.13: Hauptschleife für die Datenausgabe

In zwei Blöcken werden die acht DAC beschrieben. Für eine verbesserte Übersicht sind die einzelnen Anweisungen in Funktionen zusammengefasst. Die Ansteuerung DAC erfolgt grundlegend nach dem Muster des Vorversuches in Abschnitt 4.2. Zusätzlich wurde der Verstärkungsfaktor G_hControl.gain bei dem Beschreiben der SPI-Controller eingefügt, dadurch werden mehr Takte für einen Schleifendurchlauf benötigt. Die Ausgabefrequenz wurde deshalb auf 200 Hz begrenzt. Das Programm wartet anschließend auf das Timer-Flag und zieht den Pin $\overline{\text{LDAC}}$ auf GND-Potential. Das Potential muss mindestens 100 ns anhalten [5, Seite 6]. Deshalb wird der Index der Schwingungsdaten inkrementiert, bevor das High-Potential wieder an den Pin $\overline{\text{LDAC}}$ angelegt wird.

Am Ende eines Zyklus wird über eine einzelne if-Schleife kontrolliert, ob ein Befehl über USB erhalten wurde. Eingehende Befehle werden in dem globalen Array Received_Data_Buffer gespeichert. In der Funktion USB_command_handler() wird der Inhalt des Arrays mit der Befehlsliste verglichen.

5.2.5 Kommunikation über USB

Bei der Projekterstellung wurde die USB-Klasse CDC gewählt. Das ist eine USB-Standardklasse, die auch dem PC-Treiber bekannt ist. Die Kommunikationsabwicklung wird vom Treiber übernommen. Am PC wird das Gerät als serielles USB-Gerät erkannt. Abbildung 5.14 zeigt den Eintrag im Geräte-Manager bei erfolgreicher Erkennung. Für die Software verhält sich die Schnittstelle nun wie eine einfache serielle Schnittstelle.

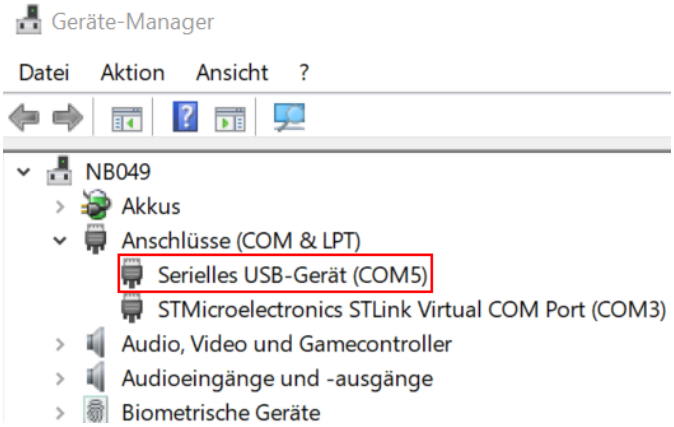


Abbildung 5.14: Erkennung des Funktionsgenerators im Geräte-Manager

Bei einem ersten Test wurde das Gerät nicht erkannt. Daraufhin wurde die zweite USB-Schnittstelle auf dem Entwicklerboard benutzt. Hier wurde das Gerät nur unzuverlässig erkannt. Mittels einer Internetrecherche wurde eine Lösung gefunden [17]. Es ist ein bekannter Fehler der IDE. Nach dem Befolgen einer kurzen Anleitung, funktioniert die Geräteerkennung zuverlässig, jedoch nur bei dem zweiten USB-Port. Es werden daher zwei USB-Kabel genutzt, eins zur Energieversorgung und eins zur Kommunikation.

5.2.6 Befehle und Dateiformat

Von einem PC werden Befehle und Dateien gesendet. Die möglichen Befehle sind in Tabelle 5.1 aufgelistet.

Befehl	**Beschreibung**
S	Start/Stop
F=X	Frequenzänderung mit X $\in \mathbb{N}$ und X $\in$ [100 2000]
W=X	Änderung des Netzprofils mit X $\in \mathbb{N}$ und X $\in$ [1 2]
LSIN	Lade reinen Sinus für jeden Ausgang
GAIN=X	Verstärkungsänderung mit X $\in \mathbb{R}$ und X $\in$ [0.000 9.999]
STATUS	Geräteindentifikation
HELP	Sendet Liste möglicher Befehle

Tabelle 5.1: Liste der möglichen Befehle über USB

Das Dateiformat wurde mit einem einfachen Aufbau definiert. Dateien beginnen mit einem Header. Der Header besteht aus vier Zeilen (siehe Abb. 5.15). Die zweite und dritte Zeile enthalten die Header- und Datenlänge. Dadurch wird erkannt, wann eine Datei vollständig übertragen ist. Darauf folgen die Daten von zwei Netzprofilen. Sie werden tabellarisch mit 16 Spalten und 512 Zeilen angegeben. Jede Spalte enthält den Kurvenverlauf eines DAC. Die acht Spalten für das erste Netzprofil und weitere acht für das zweite Netzprofil. Um auch eine einfache Dateierstellung mit dem Programm Excel zu ermöglichen, werden die Werte mit einem Semikolon getrennt. Der Wertebereich entspricht der DAC-Auflösung von 12 Bit.

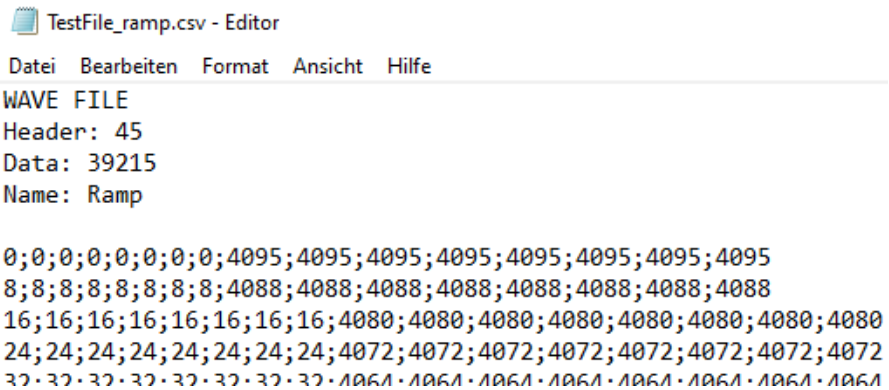


Abbildung 5.15: Beispieldatei für das festgelegte Dateiformat

5.2.7 Das Senden und das Empfangen

Der Treiber stellt eine Funktion zur Verfügung, um eine Nachricht über USB zu senden (siehe Abb. 5.16). Diese Funktion wird bei den Anweisungen STATUS und HELP, sowie bei allen Fehlermeldungen genutzt.

```
uint8_t CDC_Transmit_FS(uint8_t* Buf, uint16_t Len)
```

Abbildung 5.16: Treiberfunktion für das Senden von Nachrichten über USB. Sie ist enthalten in der Datei usbd_cdc_if.c.

Wird ein Datenpaket empfangen, wird die Treiberfunktion in Abb. 5.17 per Interrupt ausgeführt. Diese beendet den Erhalt des Datenpakets. Ein weiteres kann anschließend empfangen werden.

```
static int8_t CDC_Receive_FS(uint8_t* Buf, uint32_t *Len)
{
  /* USER CODE BEGIN 6 */
  usb_rXcallback(Buf,Len);

  USBD_CDC_SetRxBuffer(&hUsbDeviceFS, &Buf[0]);
  USBD_CDC_ReceivePacket(&hUsbDeviceFS);
  return (USBD_OK);
  /* USER CODE END 6 */
}
```

Abbildung 5.17: Treiberfunktion für das Empfangen von Nachrichten über USB. Enthalten in der Datei usbd_cdc_if.c. Die Paketdaten sind über den Pointer Buf zugänglich. Der Pointer Len gibt die Datenlänge an.

Der Funktionsaufruf $\mathrm{usb\_rXcallback(Buf, Len)}$ wurde einfügt. Dies ist eine eigene Funktion. Sie Interpretiert den Dateninhalt und bestimmt die weitere Bearbeitung. Die Funktion arbeitet nach dem Ablaufdiagramm in Abb. 5.18.

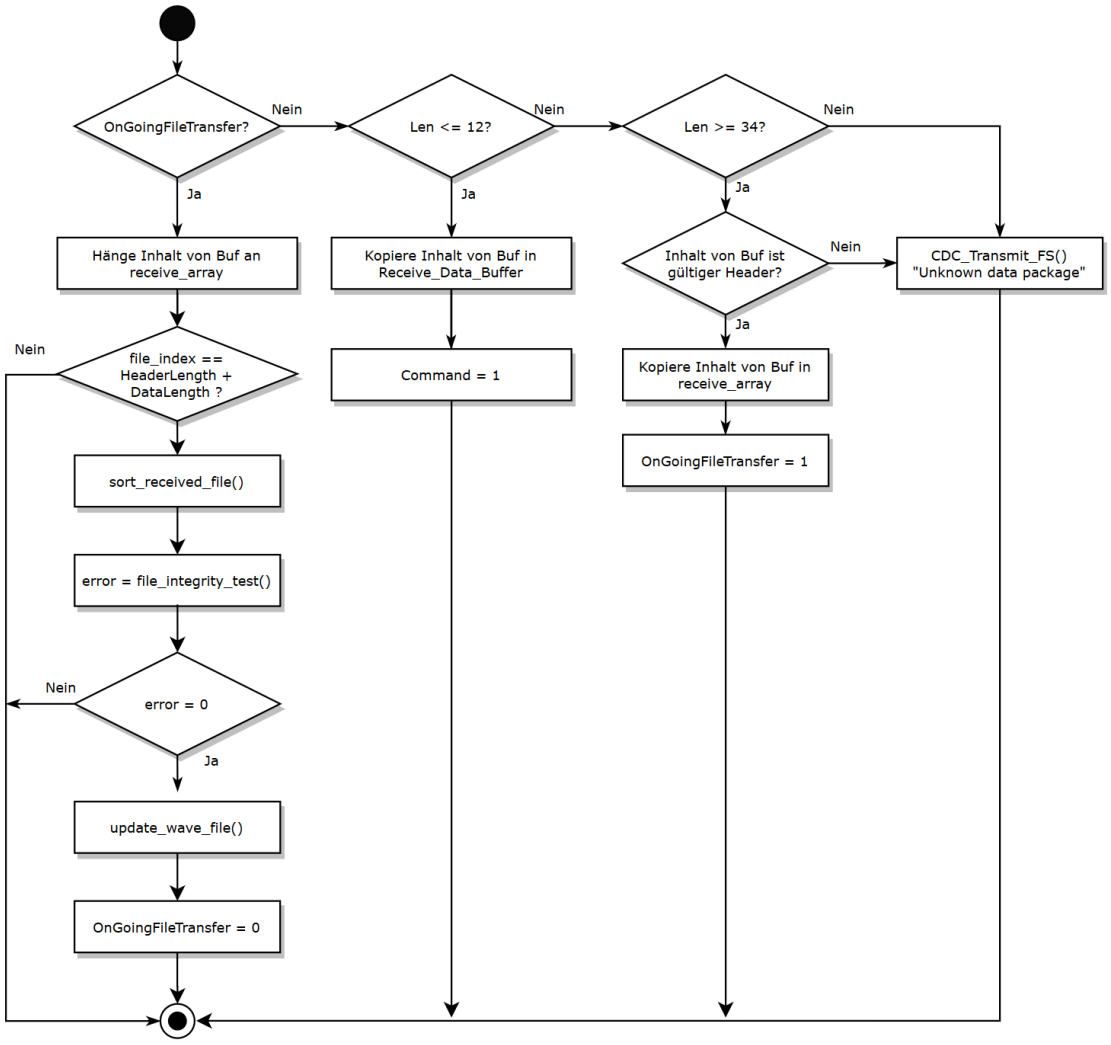


Abbildung 5.18: Ablaufdiagramm für die Bearbeitung eines eingehenden Datenpakets über USB

Anhand der Datenlänge wird entschieden, ob es sich um einen Befehl, eine Datei oder ein ungültiges Paket handelt. Ein Befehl wird in dem Array $\mathrm{Received\_Data\_Buffer[12]}$ gespeichert und die Variable $\mathrm{Command}$ wird auf eins gesetzt. In der Hauptschleife wird der Befehl dann durch den Funktionsaufruf $\mathrm{USB\_command\_handler()}$ ausgeführt. Das erste Paket einer Datei muss mindestens den Dateiheader beinhalten. Wird ein gültiger

Dateiheader erkannt, wird die globale Variable OnGoing_FileTransfer gesetzt und damit der Dateiempfang gestartet. Daraufhin wird jedes weitere Paket als Teil der Datei in dem globalen Array receive_array[42000] zwischengespeichert. Durch den globalen Index file_index werden die Paketinhalte nicht überschrieben, sondern zu einer kompletten Datei aneinandergereiht. Ist die im Header angegebene Dateilänge erreicht, werden die Zeilenumbruchzeichen mit der Funktion sort_received_file() durch ein einzelnes Semikolon ersetzt. Danach wird eine Integritätsprüfung mit der Funktion file_integrety_test() durchgeführt. Dabei wird die Anzahl und der Wertebereich der Daten geprüft. Wird die Prüfung bestanden, werden die Daten in das globale Array wave_matrix[512] übernommen. Anschließend wird die Variable OngoingFileTransfer zurückgesetzt. In die Hauptschleife wurde der Funktionsaufruf timeout_check() eingefügt. Die Funktion beendet eine laufende Dateiübertragung, wenn eine Zeitüberschreitung zwischen eingehenden Datenpaketen vorliegt.

6 Systemtests

In diesem Kapitel werden die durchgeführten Tests beschrieben. Mit den Tests sollen Design- und Implementierungsfehler erkannt werden. Auch soll gezeigt werden, dass das System funktionsfähig ist und die Spezifikation erfüllt. Die Testbedingungen werden definiert und die Ergebnisse dokumentiert. Anschließend wird das Resultat kritisch beurteilt. Zunächst wird die Funktion getestet und bewertet, danach die Qualität der Ausgangsspannung.

6.1 Funktionstests

Der entwickelte Teil des Funktionsgenerators besteht aus der Platine und den Softwarekomponenten. Um die Funktion zu prüfen, wird für jeden Test ein Testplan erstellt und durchgeführt. Als erstes wird die Funktionsfähigkeit der Platine geprüft. Danach die Verbindung zu einem PC und das Senden und Empfangen von Befehlen und Dateien.

6.1.1 Platinenfunktion

Der komplette Aufbau wird in Betrieb genommen. Die Ausgangsspannung sollte 0 V betragen. Mithilfe des Debuggers wird die Variable $\mathrm{enable\_output}$ gesetzt und damit die Ausgabe aktiviert. Nun sollte auf allen acht Kanälen eine Sinuskurve mit maximaler Amplitude und einer Frequenz von 50 Hz schwingen. Anschließend wird die Variable $\mathrm{enable\_output}$ zurückgesetzt. Die Ausgangsspannung aller Kanäle sollte wieder 0 V betragen.

Für den Test wurde das PC-Oszilloskop verwendet (siehe Abb 4.4). Der Test wird schrittweise durchgeführt. Nach jedem Testschritt werden die Verläufe der Ausgangsspannungen nacheinander aufgenommen. Die Ergebnisse werden in der Tabelle 6.1 dokumentiert. Die Abbildung 6.1 zeigt den Testverlauf bei Kanal 1.

Pos	Testschritt	Aktion	Testergebnis
1.	Inbetriebnahme	0 V an allen Ausgängen	Erfüllt
2.	Ausgabe aktivieren	Schwingung wird ausgegeben	Erfüllt
3.	Ausgabe deaktivieren	0 V an allen Ausgängen	Erfüllt

Tabelle 6.1: Funktionstest der Output-Kanäle der Platinenschaltung

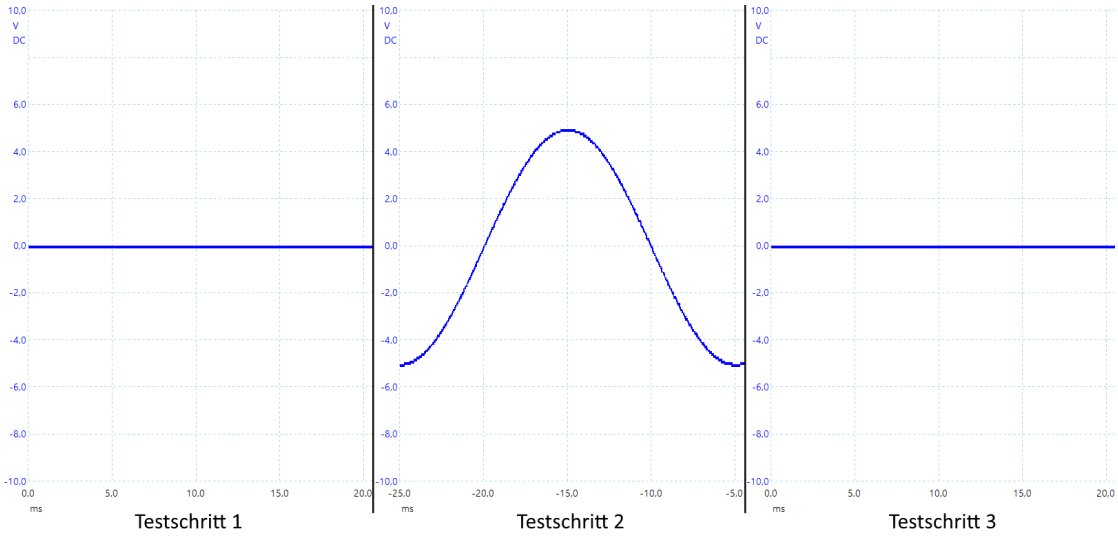


Abbildung 6.1: Spannungsverlauf von Kanal 1 beim Funktionstest der Platinenschaltung

Alle Kanäle funktionieren erwartungsgemäß. Die Sinusschwingungen haben eine Amplitude von 5 V. Im Gegensatz zu der Simulation ist keine Begrenzung erkennbar (siehe Abb. 4.2).

6.1.2 Verbindung über USB

Nach der Initialisierung sollte die USB-Schnittstelle aktiviert sein. Durch wiederholtes Einstecken und Abziehen des Steckers, wird die Plug&Play-Funktion getestet. Zur Verifizierung dient der Geräte-Manager des Betriebssystems Windows 10. Danach wird mit einem beliebigen Terminalprogramm auf den USB-Port zugegriffen. Bei diesem Test wird das Terminal HTerm 0.8.4 verwendet.

Die Ergebnisse werden in der Tabelle 6.2 dokumentiert.

Pos	Testschritt	Aktion	Testergebnis
1.	Inbetriebnahme	Verbindung getrennt	Erfüllt
2.	USB-Kabel einstecken	Verbindung hergestellt	Erfüllt
3.	USB-Kabel abziehen	Verbindung getrennt	Erfüllt
4.	USB-Kabel einstecken	Verbindung hergestellt	Erfüllt
5.	Verbinde Terminal	Zugriff gewährt	Erfüllt

Tabelle 6.2: Funktionstest der USB-Schnittstelle

Das Gerät wird sehr zuverlässig erkannt. Es konnte kein Fehler festgestellt werden. Während der Implementierungsphase wurde ein Verbindungsfehler bereits erkannt und behoben (siehe Abbschnitt 5.2.5).

6.1.3 Annahme von Befehlen

Als nächstes wird die korrekte Bearbeitung von Befehlen getestet. Schrittweise wird die Befehlsliste in Tabelle 5.1 durchgegangen. Diese werden ebenfalls über das Terminalprogramm HTerm 0.8.4 gesendet. Nach jedem Befehl wird eine Statusmeldung auf dem Terminal erwartet. Zusätzlich wird das Spannungsverhalten an den Ausgängen mit dem PC-Oszilloskop aufgenommen.

Um den Befehl LSIN (Lade reinen Sinus für jeden Ausgang) zu testen, wird mithilfe des Debuggers ein Wert des Netzprofils geändert. Nach dem Befehl LSIN sollte dieser Wert wieder korrigiert worden sein.

Anschließend wurden verschiedene fehlerhafte Befehle gesendet. Dabei wurden unter anderem auch unzulässige Werte für die Frequenz, das Netzprofil und der Verstärkung gewählt.

Die Ergebnisse werden in der Tabelle 6.3 dokumentiert.

Pos	Testschritt	Aktion	Testergebnis
1.	Sende S	Output aktiviert	Erfüllt
2.	Sende F=100	Ausgabe mit 10 Hz	Erfüllt
3.	Sende F=2500	Ausgabe mit 250 Hz	Erfüllt
4.	Sende W=2	Output Rampenfunktion	Erfüllt
5.	Netzprofil beschädigt	-	Erfüllt
6.	Sende LSIN	Netzprofil zurückgesetzt	Erfüllt
7.	Sende GAIN=0.000	Output 0 V	Erfüllt
8.	Sende GAIN=0.500	Output mit Faktor 0,5	Erfüllt
9.	Sende GAIN=1.000	Output mit Faktor 1	Erfüllt
10.	Sende STATUS	Sende Geräteidentifikation	Erfüllt
11.	Sende HELP	Sende Befehlsliste	Nicht erfüllt
12.	Sende S	Output deaktiviert	Erfüllt
13.	Senden fehlerhafter Befehle	-	Erfüllt

Tabelle 6.3: Funktionstest von dem Empfang von Befehlen über USB

Die Befehle werden erfolgreich angenommen und ausgeführt. Ausgenommen von dem Befehl HELP. Die gesendete Antwort befand sich auf einem veralteten Entwicklungsstand. Alle fehlerhaften Befehle wurde erkannt und mit einer Fehlermeldung an den PC abgelehnt.

6.1.4 Annahme von Dateien

Die Dateien zum Testen werden mit dem Programm Excel erstellt. Es werden Kurvenverläufe mit Sinus-, Sägezahn-, Dreieck- und Rechteckschwingungen generiert. Außerdem werden auch fehlerhafte Dateien erstellt. Folgende Fehlerfälle werden getestet:

Fehler 1: Zu viele Datenpunkte

Fehler 2: Zu wenig Datenpunkte

Fehler 3: Wertebereich überschritten

Fehler 4: Negative Werte

Fehler 5: Falsche Datenlänge im Header angegeben

Fehler 6: Falsche Headerlänge im Header angegeben

Fehler 7: Falsche Trennungszeichen zwischen Werten

Fehler 8: Kein Trennungszeichen zwischen Werten

Die erfolgreiche Annahme einer Datei sollte mit einer Nachricht bestätigt werden. Zusätzlich wird mit dem PC-Oszilloskop kontrolliert, ob der Kurvenverlauf der Spannung dem Dateiinhalten entspricht.

Die Ergebnisse werden in der Tabelle 6.4 dokumentiert. Abbildung 6.2 zeigt die Kurvenverläufe der fehlerfreien Dateien.

Pos	Testschritt	Aktion	Testergebnis
1.	Sende Datei mit Sägezahnverlauf	Datei akzeptiert	Erfüllt
2.	Sende Datei mit Dreiecksverlauf	Datei akzeptiert	Erfüllt
3.	Sende Datei mit Rechteckverlauf	Datei akzeptiert	Erfüllt
4.	Sende Datei mit Sinusverlauf	Datei akzeptiert	Erfüllt
5.	Sende Datei mit Fehler 1	Datei nicht akzeptiert	Nicht erfüllt
6.	Sende Datei mit Fehler 2	Datei nicht akzeptiert	Erfüllt
7.	Sende Datei mit Fehler 3	Datei nicht akzeptiert	Erfüllt
8.	Sende Datei mit Fehler 4	Datei nicht akzeptiert	Erfüllt
9.	Sende Datei mit Fehler 5	Datei nicht akzeptiert	Nicht erfüllt
10.	Sende Datei mit Fehler 6	Datei nicht akzeptiert	Nicht erfüllt
11.	Sende Datei mit Fehler 7	Datei nicht akzeptiert	Erfüllt
12.	Sende Datei mit Fehler 8	Datei nicht akzeptiert	Erfüllt

Tabelle 6.4: Funktionstest: Empfangen von Dateien über USB

Die richtigen Dateien werden alle akzeptiert. Die Ausgabe auf allen Kanälen entspricht den Dateiinhalten (siehe Abb. 6.2). Fehlerhafte Längenangaben im Header oder zusätzliche Daten nach einer vollständigen Datei werden nicht zuverlässig erkannt.

Das Programm wurde angepasst und der Test mehrfach mit verschiedenen Dateien wiederholt. Alle Fehler wurden danach erkannt.

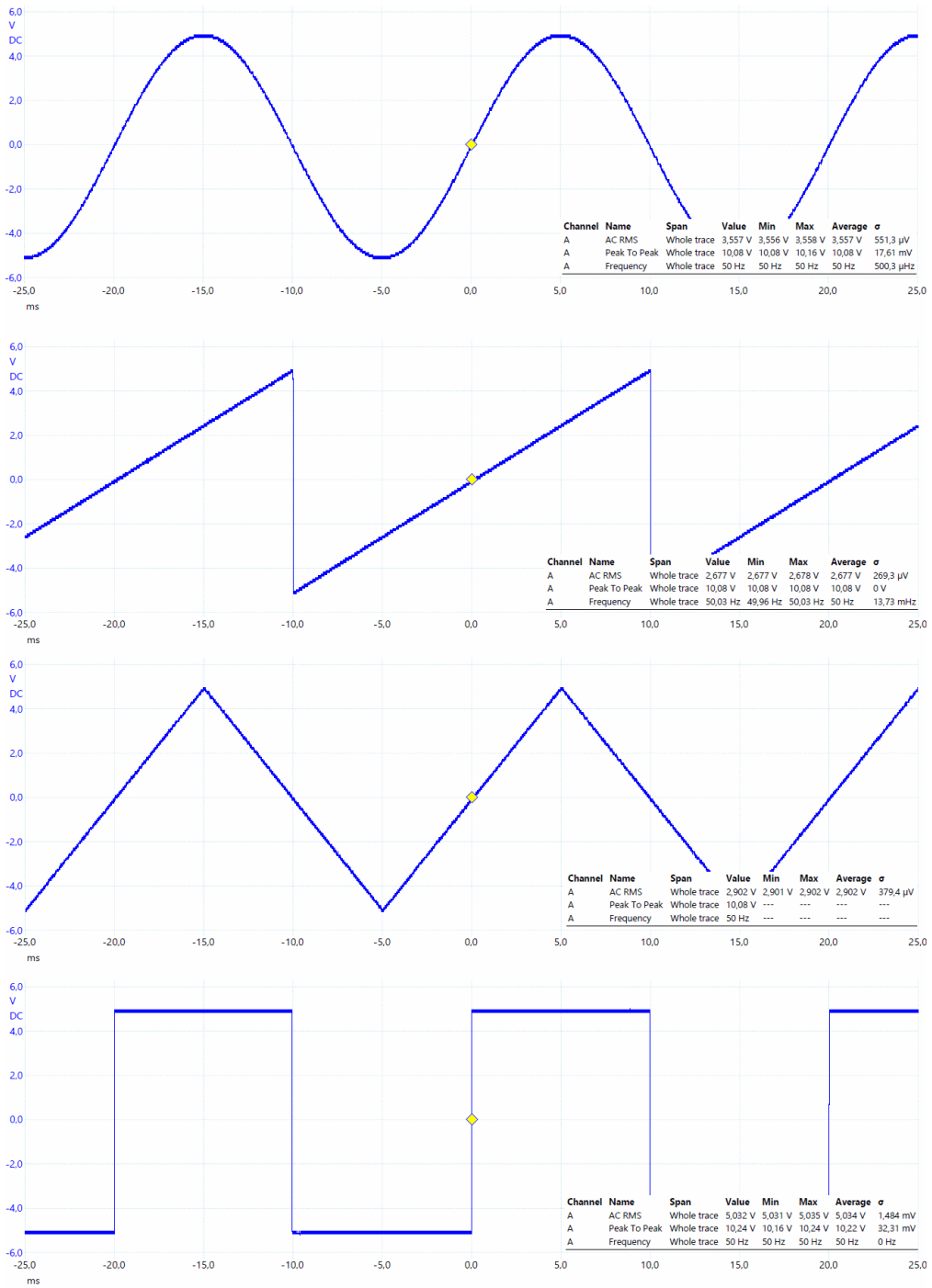


Abbildung 6.2: Spannungsverläufe von verschiedenen Kurvenformen. 1. Sinus, 2. Sägezahn, 3. Dreieck, 4. Rechteck

6.2 Genauigkeit

Die Qualität eines Funktionsgenerators ist um so höher, je genauer eine Kurvenform ausgegeben werden kann. In diesem Abschnitt wird getestet, wie genau der Spannungsbereich von -5 V bis 5 V und die eingestellte Frequenz erreicht wird.

Folgende Tests werden hierfür durchgeführt:

1) Messung des Gleichanteils

2) Messung des Spannungsbereichs

2) Messung der Frequenzgenauigkeit

Da der Funktionsgenerator zur Netzsimulation genutzt werden soll, werden die Messungen an einer Sinusschwingung durchgeführt. Auch soll geprüft werden, wie groß die Differenz zwischen den Ausgangskanälen ist.

6.2.1 Messung des Gleichanteils

Da der Offset mit einem Spannungsteiler auslegt wurde, wird die Ausgangsspannung nicht ideal um 0 V schwingen. Zum einen wurde die Schaltung in Abschnitt 3.3.1 so ausgelegt, dass ein Gleichanteil von etwa -1 mV vor der Verstärkung entsteht. Zum anderen ist die Spannung am Teiler direkt abhängig von der 5 V Versorgungsspannung des Entwicklerboards. Weiterhin wurden Widerstände mit einer Toleranz von 1 % verwendet. Dies kann zu verschiedenen Ergebnissen zwischen den Ausgabekanälen führen.

Als erstes wird die Spannungsversorgung des Entwicklerboards an verschiedenen Geräten und Kabeln gemessen. Bei Computer 1 wird kein weiteres USB-Gerät angeschlossen, bei Computer 2 werden dagegen sieben weitere USB-Geräte angeschlossen. Bei Kabel 2 hat der Stecker keinen festen Halt. Durch den Spannungsregler auf dem Entwicklerboard sollte die Versorgungsspannung nur geringfügig variieren. Tabelle 6.5 zeigt die Ergebnisse.

Pos	Gerät	Kabel 1	Kabel 2	Kabel 3
1.	Computer 1	5,039 V	4.961 V	4.971 V
2.	Computer 2	4.889 V	4.880 V	4.891 V
2.	USB-Netzgerät	5.113 V	5.041 V	5.108 V

Tabelle 6.5: Messung der 5 V-Versorgungsspannung mit verschiedenen Geräten und Kabeln

Je nach der verwendeten Hardware kommt es zu erheblichen Spannungsschwankungen. Das Entwicklerboard gibt die USB-Spannung direkt an den 5 V-Ausgang weiter. Es handelt sich hierbei um einen Konzeptionsfehler. Es wurde angenommen, dass die Spannung auf dem Board geregelt wird. Der 5 V-Spannungsregler wird nur bei einer externen Spannungsversorgung von 7 V bis 12 V benutzt. Die USB-Schnittstelle hat laut Spezifikation eine Toleranz von $\pm$ 0.25 V [11, Seite 22].

Je geringer die Spannung, desto weniger wird der Gleichanteil durch den Spannungsteiler reduziert. Damit erhöht sich der Maximalwert der Kurvenform der Spannung. Sollte der Höchstwert größer als die Versorgungsspannung sein, wird die Ausgangsspannung begrenzt. Im Weiteren wird Computer 1 mit Kabel 1 für die Energieversorgung verwendet.

Als nächstes werden die Ausgangsspannungen der Kanäle auf 0 V gesetzt und in Tabelle 6.6 dokumentiert. Auch die Differenzen zwischen den Kanälen werden in der zweiten Spalte dargestellt.

Kanal	Messwert	Differenz zu Kanal 1
1	-47,10 mV	-
2	-50,94 mV	-3,84 mV
3	-55,05 mV	-7,95 mV
4	-53,18 mV	-6,08 mV
5	-50,11 mV	-3,01 mV
6	-39,09 mV	+8,01 mV
7	-49,06 mV	-1,99 mV
8	-41,58 mV	+5,52 mV

Tabelle 6.6: Messung der Spannungen an den acht Kanälen mit Sollwert gleich 0 V

Es bleibt ein Gleichanteil von durchschnittlich -48,3 mV. Diese Spannung entsteht durch die leicht erhöhte Versorgungsspannung von 5,039 V (siehe Tabelle 6.5). Die Reduktion durch den Spannungsteiler und die Verstärkung der OP-Schaltung heben sich nahezu auf (siehe Abschnitt 3.3). Deshalb wirkt sich der Übertrag von 39 mV im Verhältnis 1:1 auf den Gleichanteil aus. Die Differenzen zwischen den Kanälen, werden von der Herstellertoleranz der Widerstände verursacht.

Da der Funktionsgenerator ausschließlich für Wechselspannung gebraucht wird, ist der Gleichanteil nicht problematisch. Der nachgeschaltete Verstärker wird den Gleichanteil mit einer Eingangskapazität herausfiltern.

6.2.2 Messung des Spannungsbereichs

In diesem Abschnitt wird der Spannungsbereich der Ausgänge gemessen. Daraus soll die erreichte Genauigkeit der Spannungswerte bestimmt werden. Die Spitze-Spitze-Spannung soll 10 V betragen. Die Ausgangsspannung wird auf den Minimal- und Maximalwert eingestellt und der Gleichspannungsmittelwert gemessen.

Die Messung wird wieder mit dem PC-Oszilloskop durchgeführt. Es werden wieder verschiedene Spannungsquellen verwendet, um den Einfluss auf den Spannungsbereich zu bewerten.

In Tabelle 6.7 sind die Messergebnisse aufgelistet.

Kanal	Minimum	Maximum	Spitze-Spitze
	Computer 1 (U_{USB} = 5,039 V)		
1	-5,118 V	4,955 V	10,073 V
2	-5,127 V	4,933 V	10,060 V
3	-5,124 V	4,931 V	10,055 V
4	-5,131 V	4,945 V	10,076 V
5	-5,121 V	4,949 V	10,070 V
6	-5,122 V	4,949 V	10,071 V
7	-5,122 V	4,936 V	10,058 V
8	-5,120 V	4,941 V	10,061 V
	Computer 2 (U_{USB} = 4,882 V)		
1	-4,968 V	4,910 V	9,878 V
2	-4,988 V	4,909 V	9,897 V
3	-4,982 V	4,912 V	9,897 V
4	-5,002 V	4,913 V	9,915 V
5	-4,974 V	4,908 V	9,882 V
6	-4,976 V	4,912 V	9,888 V
7	-4,975 V	4,911 V	9,886 V
8	-4,970 V	4,909 V	9,879 V
	USB-Netzgerät (U_{USB} = 5,134 V)		
1	-5,197 V	4,872 V	10,069 V
2	-5,204 V	4,851 V	10,055 V
3	-5,202 V	4,855 V	10,057 V
4	-5,209 V	4,867 V	10,076 V
5	-5,199 V	4,871 V	10,070 V
6	-5,198 V	4,863 V	10,061 V
7	-5,200 V	4,861 V	10,061 V
8	-5,198 V	4,860 V	10,058 V

Tabelle 6.7: Messung des Spannungsbereichs an verschiedenen Geräten

Der Betrieb mit Computer 2 ist nicht möglich. Die Ausgangsschwingungen werden von der Versorgungsspannung begrenzt. Die Versorgungsspannung muss zwangsläufig über 5 V liegen. Nur so ist der fehlerfreie Betrieb möglich.

Weiterhin lässt sich aus der Tabelle ablesen, dass eine erhöhte Versorgungsspannung keinen Einfluss auf die Spitze-Spitze-Spannungen hat. Das war auch zu erwarten, da Differenzenverstärker verwendet wurden. Alle Kanäle haben nur geringfügige Abweichungen. Der höchste Wert beträgt 10,076 V. Gefordert waren 10 V. Damit weicht die Spannung 0,76 % von der Vorgabe ab.

Der erreichte Scheitelfaktor wird mit der Gleichung (2.1) bestimmt. Der Effektivwert und die Amplitude werden dafür bei einer Sinusschwingung gemessen. Mit dem AC-Filter des PC-Oszilloskops wurde ein RMS-Wert von 3,557 V gemessen. Die Amplitude der Spannung wurde mit 5,039 V gemessen.

$$Scheitelfaktor = \frac{\hat{u}}{RMS}$$
$$Scheitelfaktor = \frac{5,039\,V}{3,557\,V}$$
$$Scheitelfaktor = \underline{1,417}$$

Der Scheitelfaktor wird mit einem Wert von 1,417 berechnet. Idealer Weise wird ein Wert von 1,414 ($\sqrt{2}$) erreicht.

6.2.3 Messung der Frequenzgenauigkeit

Durch die Software ist die Frequenz in Schritten von 0,1 Hz einstellbar. Wichtig ist, dass die Frequenz konstant gehalten wird. Laut den Anforderungen ist eine Standardabweichung von unter 1 mHz zu erreichen. Um die Frequenzgenauigkeit zu bestimmen, wird eine Sinusschwingung mit 50 Hz gemessen. Die Standardabweichung wird von dem PC-Oszilloskop angegeben. In dem Datenblatt des PC-Oszilloskops ist eine Genauigkeit der Zeitbasis von $\pm$ 50 ppm angegeben. Die Abbildung 6.3 zeigt das Ergebnis.

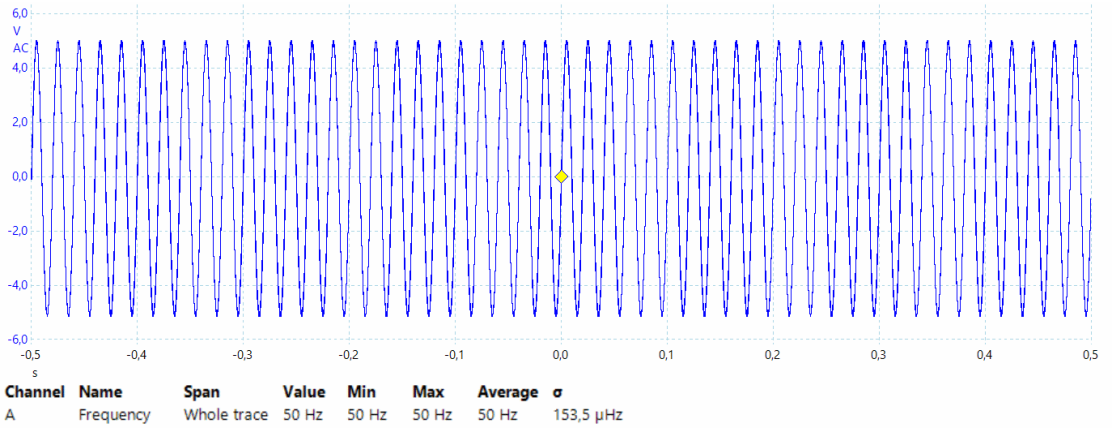


Abbildung 6.3: Messung der Frequenz einer Sinusschwingung mit 50 Hz. Über 50 Perioden gemessen

Es wurde eine Standardabweichung von 153,5 μHz gemessen, damit ist die Anforderung erfüllt. Die Frequenz ist stabil genug. Durch einen externen Oszillator mit höherer Genauigkeit kann dieser Wert weiter verbessert werden. Für den Zweck des Funktionsgenerators ist es jedoch nicht nötig.

6.3 Spektrale Reinheit

Um die spektrale Reinheit zu bewerten, wird das Frequenzspektrum aufgenommen. Die Schwingungen werden durch je 512 Einzelwerte gebildet. Dadurch kommt es zu einem stufenartigen Spannungsverlauf. Der steile Anstieg zwischen den Stufen sorgt für ungewollte Frequenzanteile. Außerdem verfügen die Ausgabekanäle über keinen Filter und keine Glättung. Von den verwendeten Bauteilen verursacht der DC/DC-Wandler das größte Rauschen. Da nur Frequenzen bis 3 kHz für den Verstärker relevant sein sollen, wird das Spektrum nur bis 5 kHz betrachtet.

Die Messung wird mit dem PC-Oszilloskop durchgeführt. Es wird ein Lastwiderstand von 10 kΩ angeschlossen. Der Wert SNR und der Wert THD von dem PC-Oszilloskop berechnet und angezeigt.

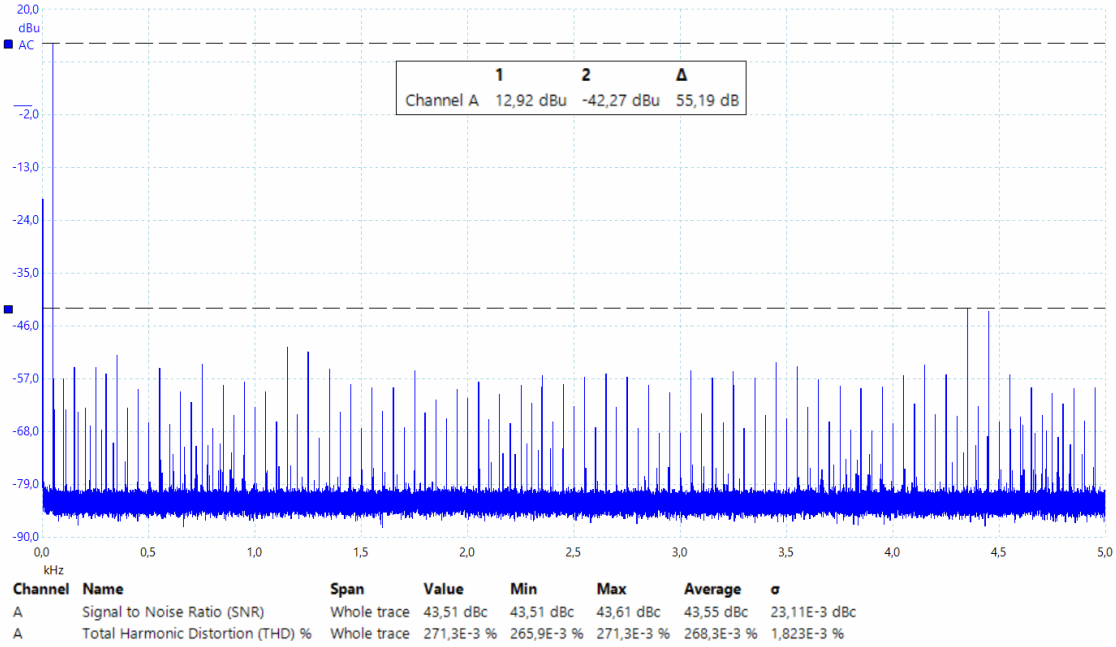


Abbildung 6.4: Frequenzspektrum bei einer Sinusschwingung mit 50 Hz, Fensterfunktion: Hanning, mit 131072 Spektrum-Bins

Die Amplitude der Grundschwingung liegt bei 12,92 dB. Neben der Grundschwingung zeigt das Spektrum die größere Amplitude bei 4,35 kHz und bei 4,45 kHz. Diese Amplituden haben einen Wert von etwa -42,27 dB. Damit ergibt sich ein Amplitudenabstand von 55,19 dB. Weitere signifikante Frequenzanteile sind nicht zu erkennen. Der Wert des SNR wurde mit 43,51 dB berechnet. Der Wert des THD liegt bei 0,271 %. In der Anforderung ist ein Wert von unter 1 % zu erreichen. Damit ist der Anforderung an die spektrale Reinheit erfüllt.

Wird ein Filter mit einer Grenzfrequenz von 3 kHz eingebaut, können die genannten Frequenzen herausgefiltert werden. In der Abbildung 6.5 ist das Spektrum wiederholt, mit den Linealen auf der nächsthöheren Amplitude unter dieser Grenzfrequenz angezeigt.

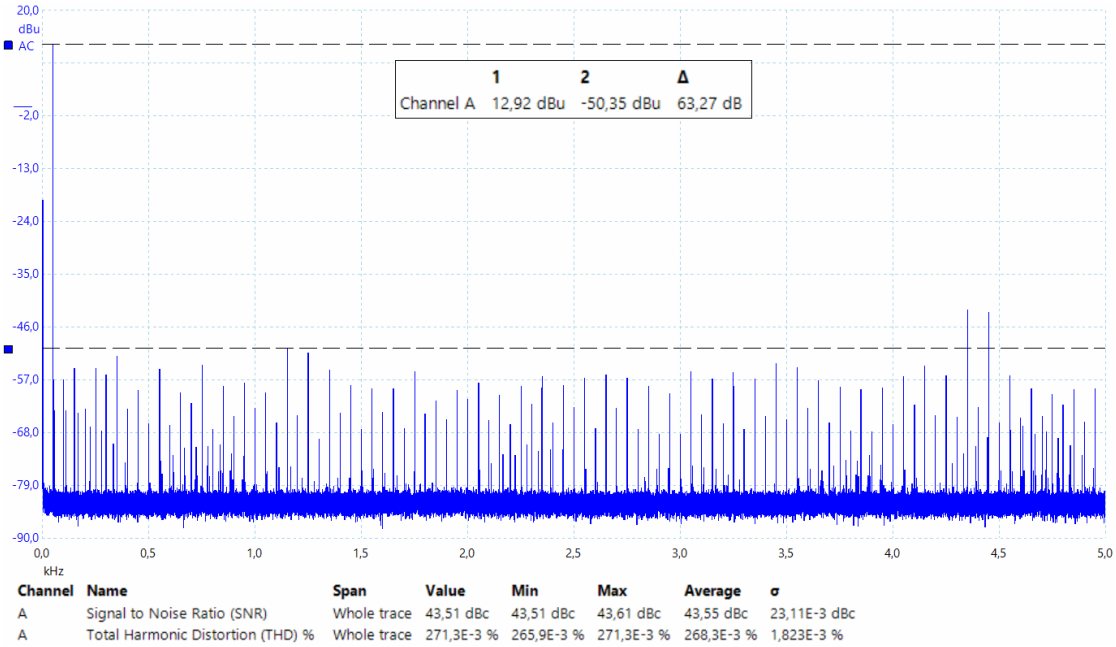


Abbildung 6.5: Frequenzspektrum bei einer Sinusschwingung mit 50 Hz, Fensterfunktion: Hanning, mit 131072 Spektrum-Bins, Das Lineal auf höchster Amplitude unter einer Frequenz von 3 kHz

Der Amplitudenabstand wird dann mit 63,27 dB gemessen. Für eine präzise Analyse muss die Messung mit einem Messgerät mit hoher Genauigkeit wiederholt werden.

7 Bewertung und Ausblick

In diesem Kapitel werden offene und abgeschlossene Punkte der Entwicklung genannt. Die Firma Stucke Elektrotechnik GmbH entwickelt und produziert Netzschutz-Relais. Für die Prüfung der Schutzmechanismen wird ein Netz-Simulator verwendet, dessen Testabläufe von dem Hersteller programmiert werden müssen. Für viele Tests in der Soft- und Hardwareentwicklung wird ein einfacher, flexibler und kostengünstiger Funktionsgenerator benötigt. Die Entwicklung einer Platine wurde in Auftrag gegeben, die mithilfe eines Entwicklerboards, eine beliebige Kurvenform ausgeben kann.

7.1 Fazit

Es wurde ein Teil eines Funktionsgenerators entwickelt, an den die Entwicklung eines Verstärkers anschließen kann. Das Testen an einem Netzschutz-Relais ist in dieser Phase noch nicht möglich gewesen. Mit den durchgeführten Tests wurde jedoch gezeigt, dass das Gerät innerhalb der gesetzten Spezifikationen arbeitet.

Es konnte eine Platine erstellt werden, die die Daten von dem Entwicklerboard in acht analoge Spannungen umsetzt. Durch ungelöste Probleme mit dem Treiber der USB-Schnittstelle, werden zwei USB-Kabel benötigt. Weiterhin wurde zu spät erkannt, dass die 5 V-Spannung des Entwicklerboards die unbearbeitete USB-Spannung ist. Damit ist die Funktion der Platine abhängig von der USB-Spannung. Das Ziel für die Energieversorgung wurde damit nicht erreicht.

Über eine Datei können die Kurvenverläufe innerhalb der Grenzwerte modelliert werden. Dadurch können Verzerrungen und Verschiebungen hinzugefügt werden. Während des Betriebes können Parameter eingestellt werden.

Die Kommunikation über USB bietet eine einfache Schnittstelle zu jedem PC. Dateien und Befehle können so versendet werden. Das Steuerprogramm hat eine zuverlässige Erkennung von Eingabefehlern bei Dateien und Befehlen.

7.2 Offene Punkte

Die Art der Spannungsversorgung muss dringend verändert werden. Es bietet sich an, die Spannungsversorgung des Verstärkers zu nutzen, um das Entwicklerboard mit 7 V bis 12 V zu versorgen. Damit würde auch ein USB-Kabel weniger benötigt werden.

Auch wenn die Funktion erfüllt ist, können weitere Verbesserungen gemacht werden. Besonders bei der Entwicklung des Verstärkers und der PC-Anwendung, können Ideen zur Weiterentwicklung entstehen. Bisher ergaben sich die folgenden Vorschläge:

- Die Versorgung des Entwicklerboards sollte über den Verstärker erfolgen. Dafür muss die Spannung von 230 V heruntertransformiert und gleichgerichtet werden. Damit entfällt ein USB-Kabel.
- Es muss ein neuer Aufbau erstellt werden mit einem korrigiertem Platinenlayout. Dabei müssen Widerstände mit einer Toleranz unter 0,25 % verwendet werden.
- Die Messung der spektralen Reinheit muss mit einem Messgerät mit hoher Genauigkeit wiederholt werden.
- Die Netzschutz-Relais verfügen über einen Datenlogger. Die Log-Datei könnte genutzt werden, um den Netzzustand vor einem Ereignis zu rekonstruieren und über den Funktionsgenerator auszugeben.
- Über einen digitalen Ausgang könnte dem Verstärker gemeldet werden, dass die Ausgabe gestoppt ist. Damit können die Ausgänge des Verstärkers hochohmig geschaltet werden.
- Es können zusätzliche GPIOs als digitale Eingänge genutzt werden. Am zukünftigen Verstärker könnte dann ein kleines Bedienfeld eingebaut werden. Parameter wie Start/Stopp, wären dann auch ohne PC zugänglich.
- Der Flash-Speicher kann genutzt werden, um mehrere Netzprofile dauerhaft zu speichern.

Die Anforderungen konnten erfüllt werden. Mit dem einfachen Aufbau einer Datei eignet sich der Funktionsgenerator besonders gut für nicht standardmäßige Tests bei der Entwicklung neuer Funktionen und Geräte. Mit dem fertiggestellten Funktionsgenerator wird sich die Flexibilität bei der Entwicklung erhöhen.

Literaturverzeichnis

[1] *Conrad Electronic SE*. URL www.conrad.de. – Aufgerufen am: 06.02.2020

[2] *Datenblatt: Buchsenleiste 2.54mm*. EVE GmbH. – URL https://asset.conrad.com/media10/add/160267/c1/-/en/001492298DS01/datenblatt-1492298-econ-connect-buchsenleiste-standard-anzahl-reihen-2-polzahl-je-reihe-8-blg2x8-1-st.pdf. – Aufgerufen am: 06.02.2020

[3] *Datenblatt: OMICRON CMC-256plus*. OMICRON. – URL https://www.omicronenergy.com/de/produkte/cmc-256plus/. – Aufgerufen am: 06.02.2020

[4] *Mouser Electronics*. URL www.mouser.de. – Aufgerufen am: 06.02.2020

[5] *Datenblatt: MCP4921/4922*. Microchip Technology Inc, 2004. – URL https://www.mouser.de/datasheet/2/268/21897a-70809.pdf. – Aufgerufen am: 06.02.2020

[6] *PICOSOPE 3206B*. Pico Technology, 2011. – URL http://download.zeitech.de/Documents/PicoScope3000_Datasheet.pdf. – Aufgerufen am: 06.02.2020

[7] *Datenblatt: TLV2372IDR*. Texas Insruments, 2016. – URL http://www.ti.com/lit/ds/symlink/tlv2373.pdf. – Aufgerufen am: 06.02.2020

[8] *Datenblatt: RFMM-0505S*. RECOM, 2018. – URL https://www.mouser.de/datasheet/2/468/RFMM-1711257.pdf. – Aufgerufen am: 06.02.2020

[9] *Datenblatt: STM32H743ZI*. STMicroelectronics, 2019. – URL https://www.st.com/en/microcontrollers-microprocessors/stm32h743zi.html. – Aufgerufen am: 06.02.2020

[10] *STM32H7 Reference manuel RM0433*. STMicroelectronics, 2019. – URL https://www.st.com/content/ccc/resource/technical/document/reference_manual/group0/c9/a3/76/fa/55/46/45/fa/DM00314099/files/DM00314099.pdf/jcr:content/translations/en.DM00314099.pdf. – Aufgerufen am: 06.02.2020

[11] *STM32H7 User manuel UM2407*. STMicroelectronics, 2019. – URL https://www.st.com/content/ccc/resource/technical/document/user_manual/group1/95/1a/9a/89/87/6a/45/70/DM00499160/files/DM00499160.pdf/jcr:content/translations/en.DM00499160.pdf. – Aufgerufen am: 06.02.2020

[12] *Elektronik Kompendium - Subtrahiererschaltung*. www.elektronik-kompedium.de, 2020. – URL http://www.elektronik-kompendium.de/sites/sit/0210153.htm. – Aufgerufen am: 01.04.2020

[13] ADALBERT PRECHTL, Christian T.: *Vorlesungen über die Grundlagen der Elektrotechnik 2.Auflage Band 2*. SpringerWien NewYork, 2008

[14] BERNSTEIN, Herbert: *Elektrotechnik/Elektronik für Maschinenbauer 2.Auflage*. Springer Vieweg, 2012

[15] ERWIN BÖHMER, Wolfgang O.: *Elemente der angewandten Elektronik*. Vieweg+Teubner, 2010

[16] HORST STEFFEN, Hansjüren B.: *Elektrotechnik Grundlagen 6.Auflage*. Teubner, 2007

[17] MICHAEL: *Stack overflow What is issue with STM32 Virtual Com Port? I can not open it*. URL https://stackoverflow.com/questions/56490843/what-is-issue-with-stm32-virtual-com-port-i-can-not-open-it. – Aufgerufen am: 06.02.2020

[18] NERRETER, Wolfgang: *Grundlagen der Elektrotechnik 2.Auflage*. Hanser, 2011

[19] PASCHOTTA, Dr. R.: *Drehstrom*. www.energie-lexikon.info, 2020. – URL https://www.energie-lexikon.info/drehstrom.html. – Aufgerufen am: 06.02.2020

[20] RICHARD MARENBACH, Christian T.: *Elektrische Energietechnik 2.Auflage*. Springer Vieweg, 2013

Abbildungsverzeichnis

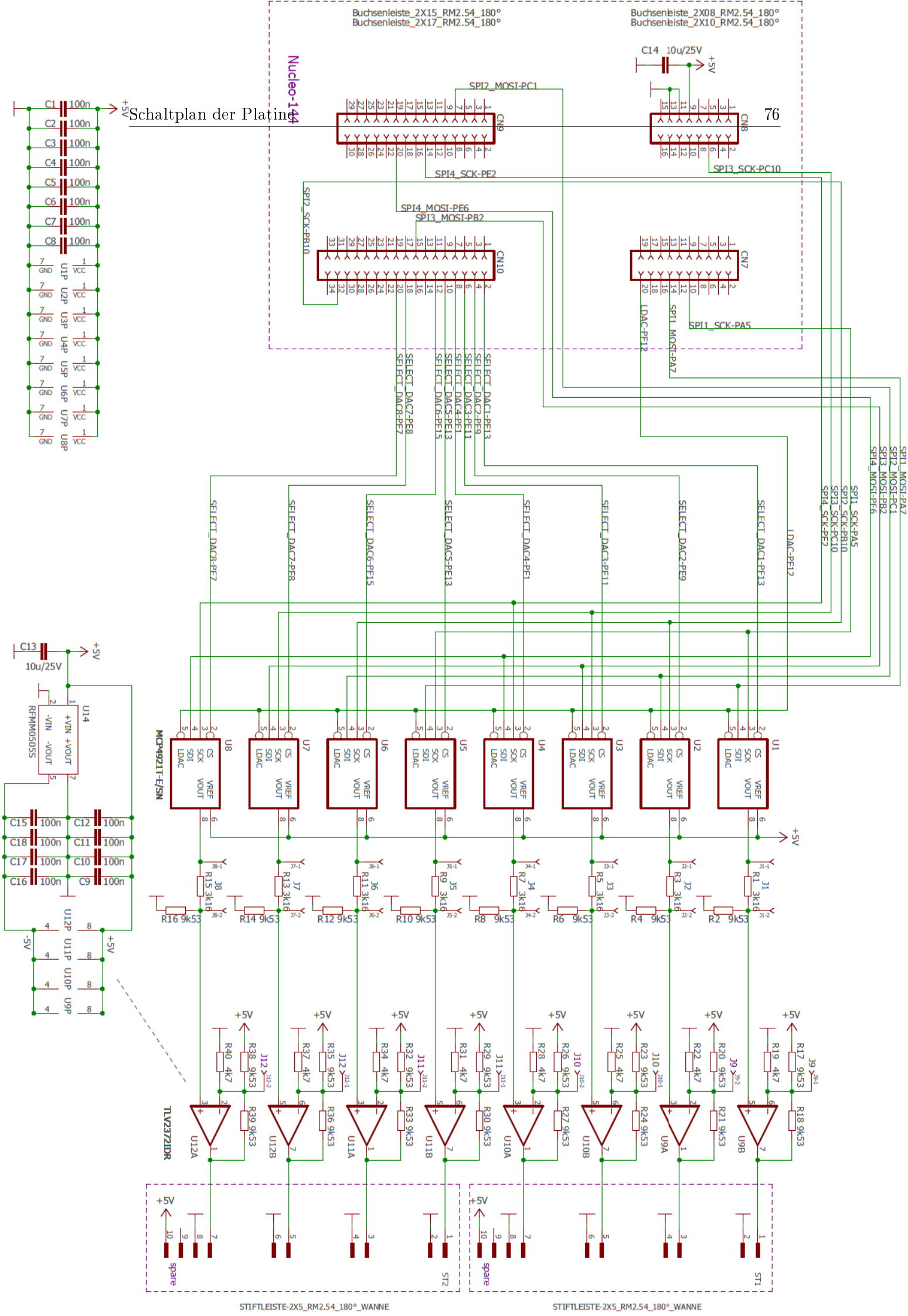
Tabellenverzeichnis

Abkürzungsverzeichnis

DAC	Digital-Analog-Wandler
LSB	Least significant bit
OP	Operationsverstärker
ASCII	Amerikanischer Standard-Code für den Informationsaustausch
USB	Universal serial bus
CPU	Central processing unit
SPI	Serial pephidral interface
STM	STMicroelectronics
IDE	Integrated Development Environment
GPIO	General purpose input output
SMD	Surface mounted device
SOIC	Small outline integrated circuit
GND	Ground potential
MOSFET	Metal-Oxid-Halbleiter-Feldeffekttransistor
EMV	Elektromagnetische Verträglichkeit
FIFO	First in first out
EOT	End of transmission
PC	Personal computer
SR	Slewrate
RMS	Root mean square
DC	Direct current
AC	Alternating current
V	Volt
A	Ampere
Ω	Ohm
Hz	Hertz

A Schaltplan der Platine

In diesem Anhang befindet sich der Schaltplan des erstellten Platinenlayouts.

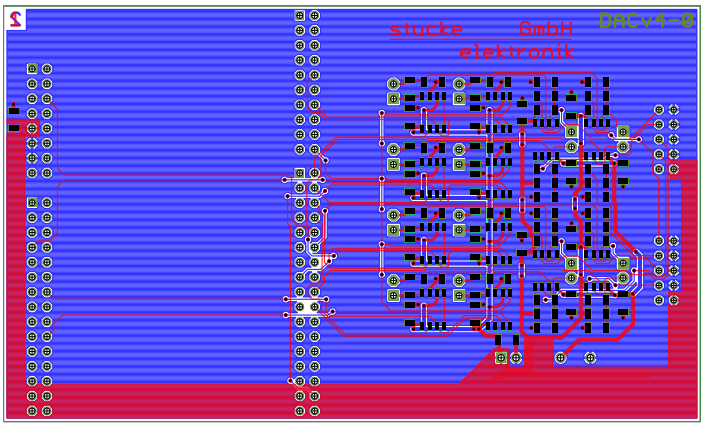

Buchsenleiste_2X15_RM2.54_180°
Buchsenleiste_2X17_RM2.54_180°
Buchsenleiste_2X08_RM2.54_180°
Buchsenleiste_2X10_RM2.54_180°
Nucleo-144
CN9
CN8
CN10
CN7
C14 10u/25V
+5V
SPI2_MOSI-PC1
SPI4_SCK-PE2
SPI3_SCK-PC10
SPI4_MOSI-PE6
SPI3_MOSI-PB2
SPI2_SCK-PB10
SPI1_SCK-PA5
SPI1_MOSI-PA7
LDAC-PE12
SELECT_DAC1-PE13
SELECT_DAC2-PE9
SELECT_DAC3-PE11
SELECT_DAC4-PE1
SELECT_DAC5-PE13
SELECT_DAC6-PE15
SELECT_DAC7-PE8
SELECT_DAC8-PE7
C1 100n
C2 100n
C3 100n
C4 100n
C5 100n
C6 100n
C7 100n
C8 100n
U1P U2P U3P U4P U5P U6P U7P U8P
GND VCC
C13 10u/25V
U14
RFM0505S
-VIN +VIN -VOUT +VOUT
C15 100n
C18 100n
C17 100n
C16 100n
C12 100n
C11 100n
C10 100n
C9 100n
U12P U11P U10P U9P
-5V
MCP4921T-E/SN
U8 U7 U6 U5 U4 U3 U2 U1
CS SCK SDI LDAC VREF VOUT
R15 3k16 J8
R13 3k16 J7
R11 3k16 J6
R9 3k16 J5
R7 3k16 J4
R5 3k16 J3
R3 3k16 J2
R1 3k16 J1
R16 9k53
R14 9k53
R12 9k53
R10 9k53
R8 9k53
R6 9k53
R4 9k53
R2 9k53
R40 4k7
R38 9k53
R37 4k7
R35 9k53
R34 4k7
R32 9k53
R31 4k7
R29 9k53
R28 4k7
R26 9k53
R25 4k7
R23 9k53
R22 4k7
R20 9k53
R19 4k7
R17 9k53
R39 9k53
R36 9k53
R33 9k53
R30 9k53
R27 9k53
R24 9k53
R21 9k53
R18 9k53
TLV2372IDR
U12A U12B U11A U11B U10A U10B U9A U9B
spare
ST2
ST1
STIFTLEISTE-2X5_RM2.54_180°_WANNE
STIFTLEISTE-2X5_RM2.54_180°_WANNE

B Platinenlayout

In diesem Anhang befindet sich das hergestellte Platinenlayout von der Firma BETA Layout. Der Herstellungsprozess wird schrittweise dargestellt.

Note: For best viewing and the Layer functionality, please use Adobe® Acrobat® Reader® X or higher.

Beta LAYOUT



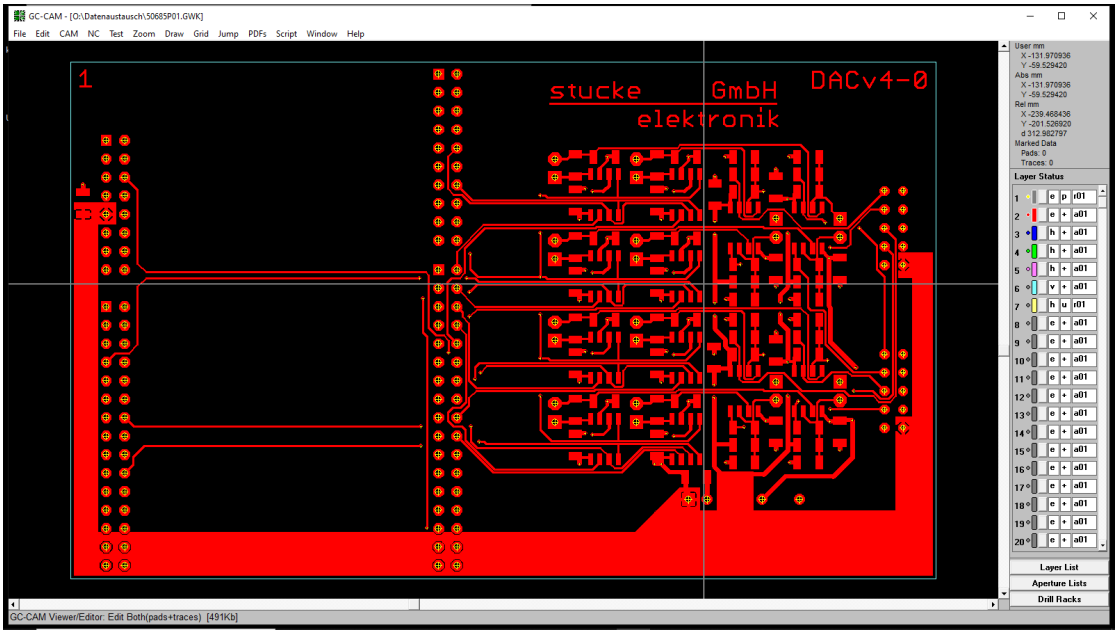


Abbildung B.1: Oberseite des erstellten Platinenlayouts

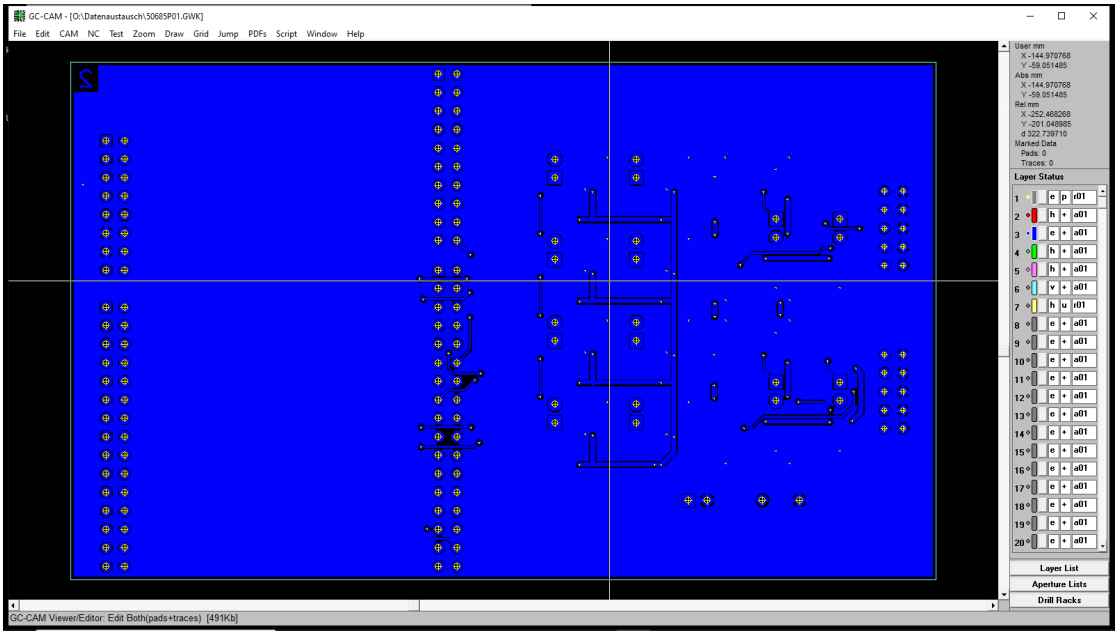


Abbildung B.2: Unterseite des erstellten Platinenlayouts

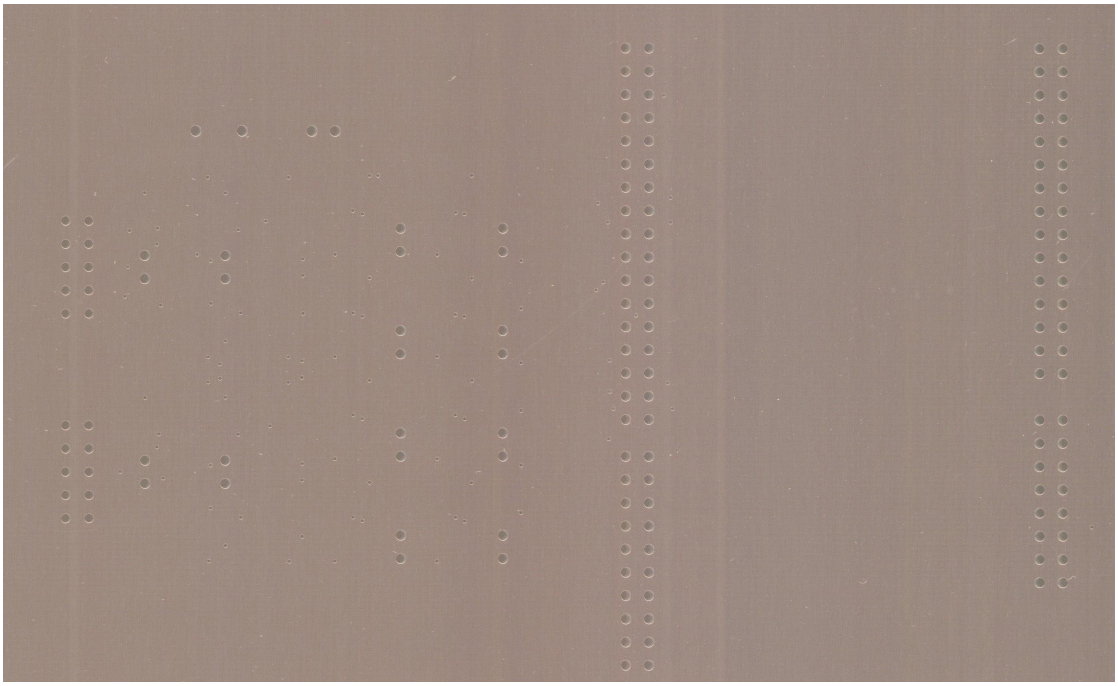


Abbildung B.3: Herstellung der Platine, Schritt 1: Bohren und Durchkontaktieren

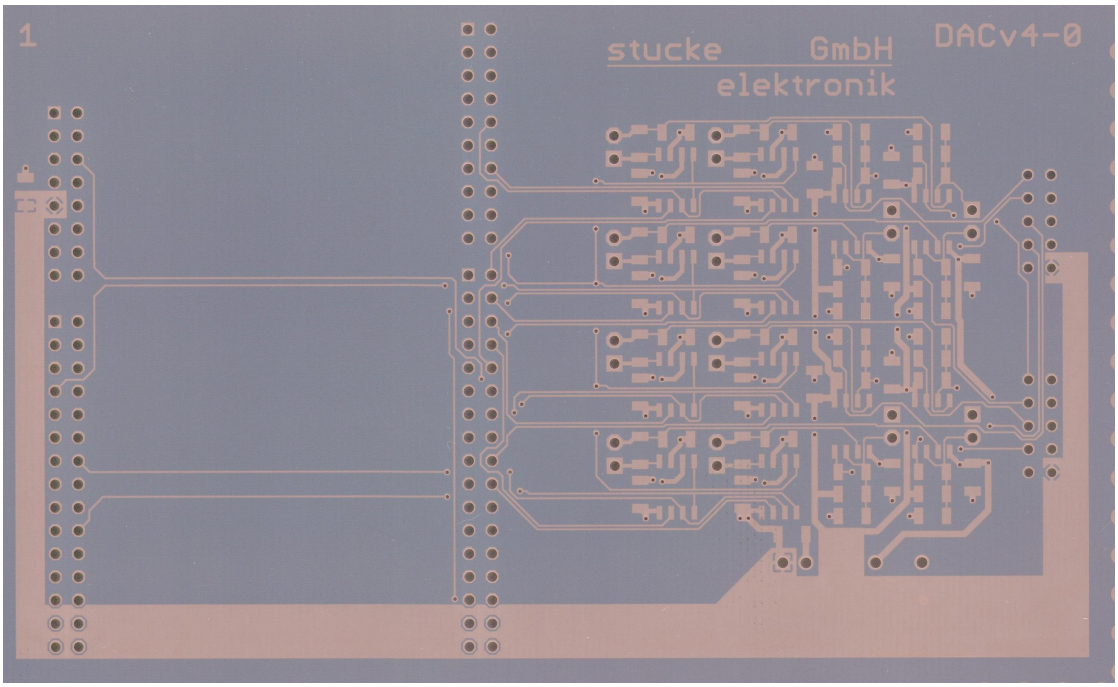


Abbildung B.4: Herstellung der Platine, Schritt 2: Belichten Oberseite

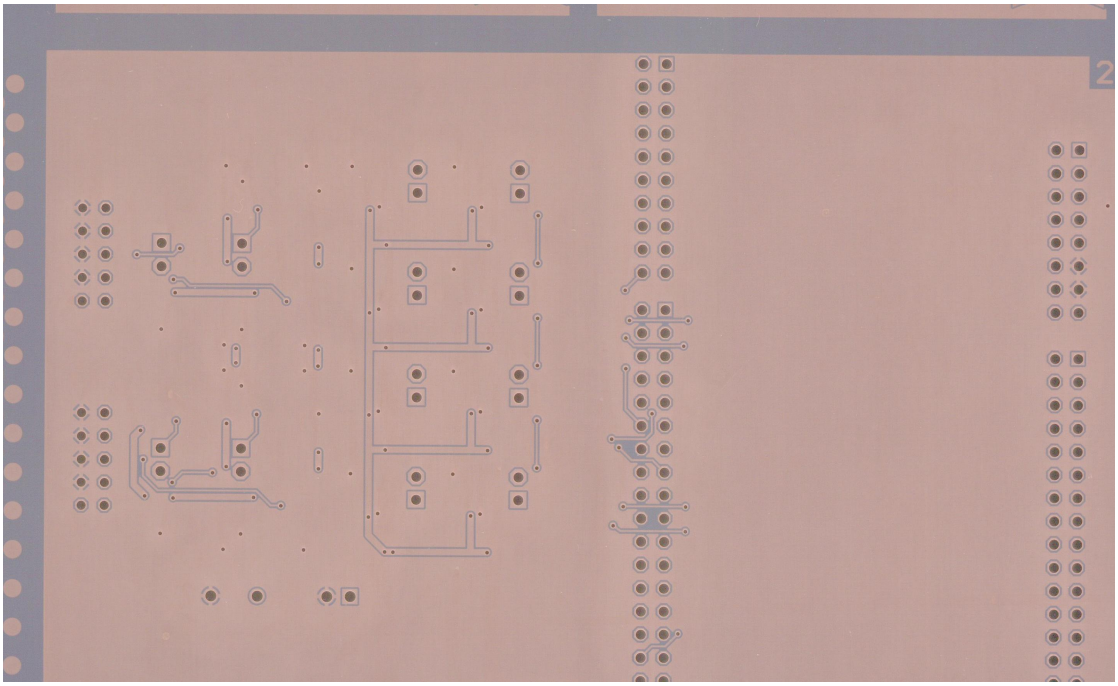


Abbildung B.5: Herstellung der Platine, Schritt 3: Belichten Unterseite

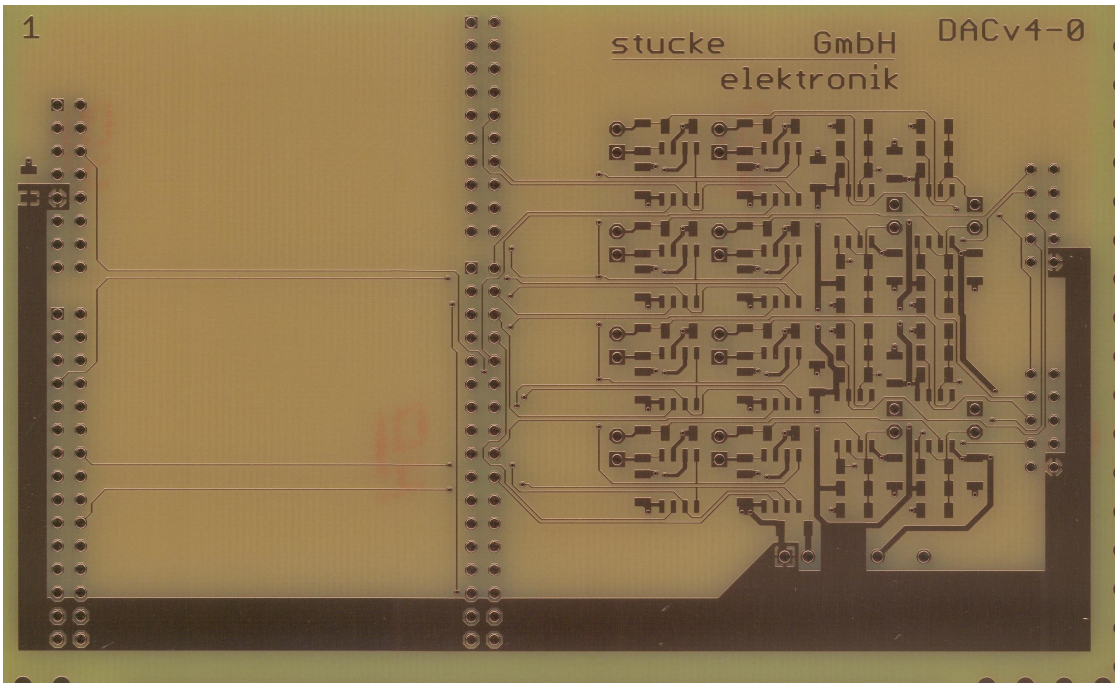


Abbildung B.6: Herstellung der Platine, Schritt 5: Ätzen der Oberseite

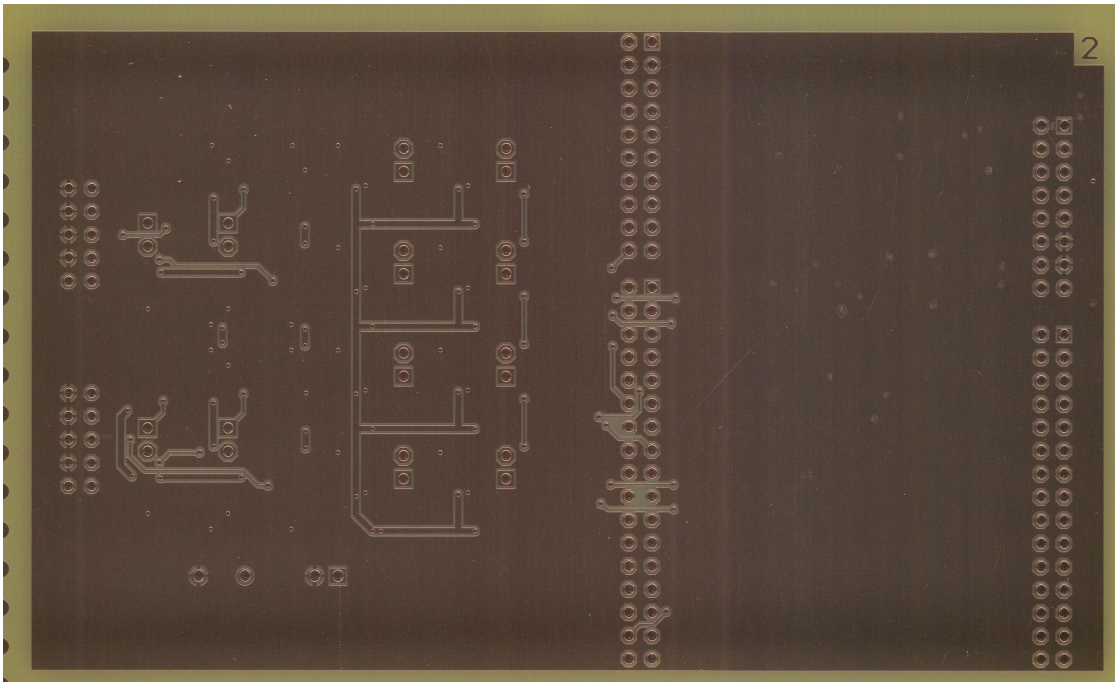


Abbildung B.7: Herstellung der Platine, Schritt 5: Ätzen der Unterseite

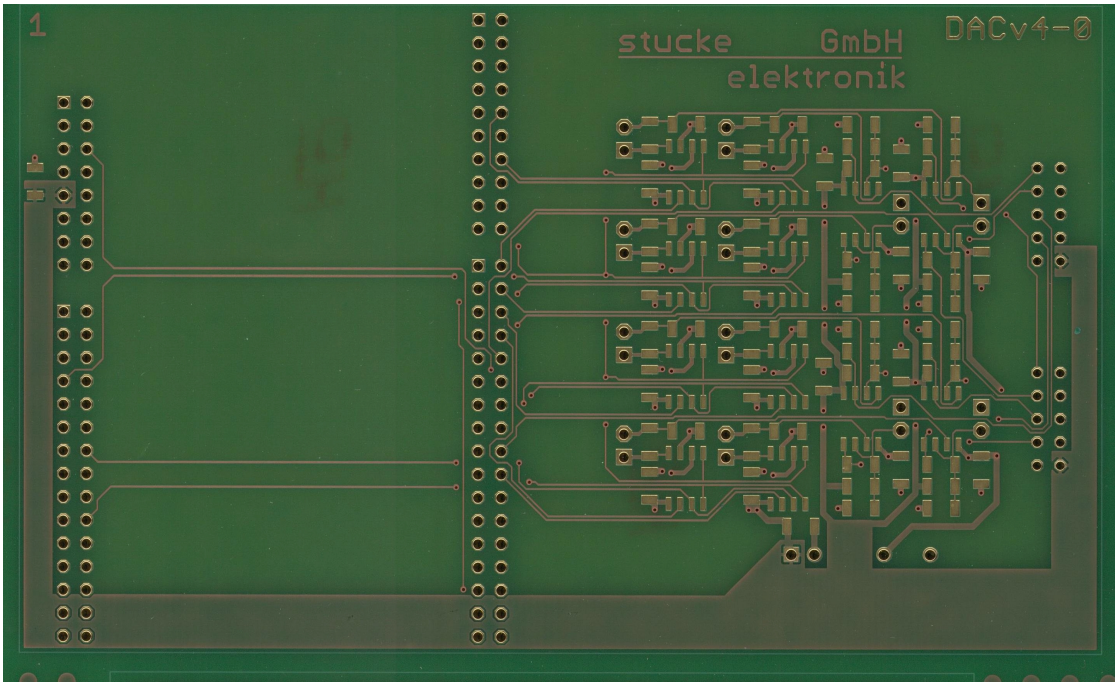


Abbildung B.8: Herstellung der Platine, Schritt 6: Lotstopp auf die Oberseite

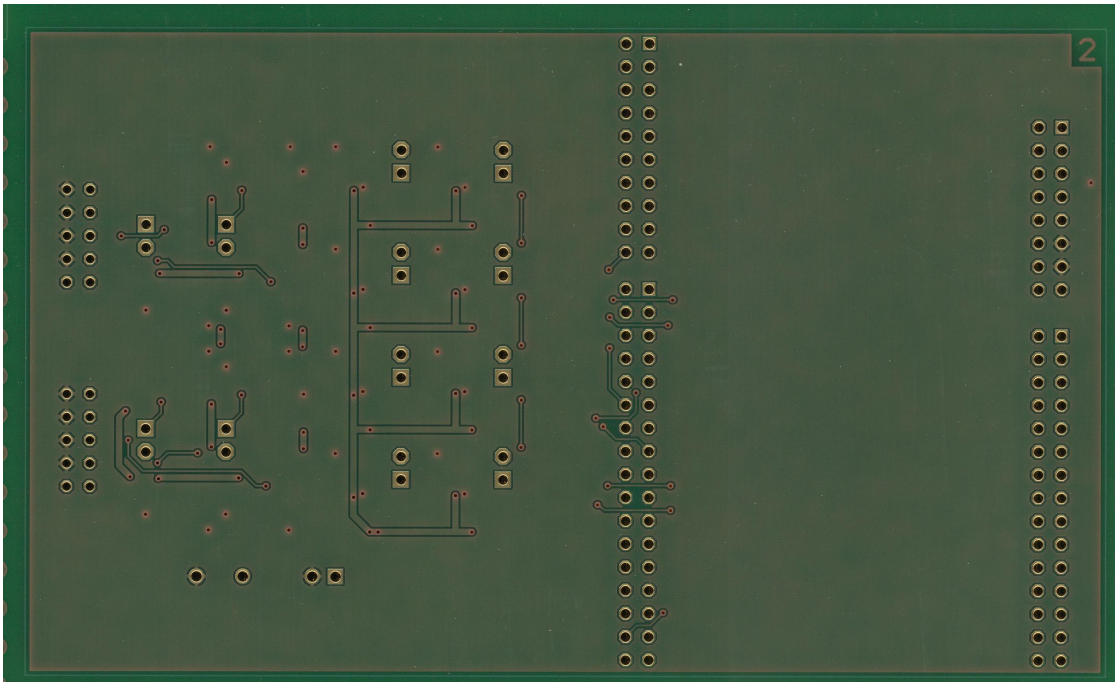


Abbildung B.9: Herstellung der Platine, Schritt 7: Lötstopp auf die Unterseite

C Codes Listings

In diesem Anhang befindet sich der C-Code des Mikrocontrollers. Die Projektverzechnisse, sowie weiterer Code befindet sich auf CD.

C.1 MAIN.c

main.c

```
1 /**
2
  ******************************************************************************
  ****
3   * @file           : main.c
4   * @brief          : Main program body
5
  ******************************************************************************
  ****
6   * @author         : Jonas Jegminat
7   * @date           : 12.05.2020
8
  ******************************************************************************
  ****
9   * @ Description   :
10  * Program controls the output of eight external DAC (MCP4921) over SPI.
11  * It is to be used with the developed DAC-board
12  * The second mini-USB connector is initialized as a serial USB-Device and
13  * can communicate with a Computer.
14  * @ USB commands  :
15  * S       = start/stop toggle
16  * F=X     = set frequency (minF <= X <= maxF)
17  * W=X     = set wave source (1 <= X <= 2)
18  * LSIN    = load standard sin wave
19  * GAIN=X  = set output gain (0 <= X <= 9.999)
20  * STATUS  = send name, Frimware version, unit number
21  * HELP    = sends list of available commands
22
  ******************************************************************************
  ****
23 **/
24
25 /************************************ Includes
  ****************************************/
26 #include "main.h"
27 #include <math.h>
28 #include <stdlib.h>
29 #include <stdio.h>
30 #include "usb_device.h"
31 #include "usbd_cdc.h"
32 #include "usbd_cdc_if.h"
33 #include "usbd_core.h"
34
35 /************************************ Private defines
  ************************************/
36 struct wave
37 {
38     uint16_t out_11;    // Profile 1, U_L1
39     uint16_t out_12;    // Profile 1, U_L2
40     uint16_t out_13;    // Profile 1, U_L3
41     uint16_t out_14;    // Profile 1, U_E
42     uint16_t out_15;    // Profile 1, I_L1
43     uint16_t out_16;    // Profile 1, I_L2
44     uint16_t out_17;    // Profile 1, I_L3
45     uint16_t out_18;    // Profile 1, I_E
46     uint16_t out_21;    // Profile 2, U_L1
47     uint16_t out_22;    // Profile 2, U_L2
48     uint16_t out_23;    // Profile 2, U_L3
49     uint16_t out_24;    // Profile 2, U_E
50     uint16_t out_25;    // Profile 2, I_L1
51     uint16_t out_26;    // Profile 2, I_L2
52     uint16_t out_27;    // Profile 2, I_L3
```

main.c

```
53      uint16_t out_28;    // Profile 2, I_E
54  };
55
56  struct global_control_handler
57  {
58      uint16_t frequency;         // 100-2000 frequency
59      uint16_t wave_source;       // 0 = profile 1, 1 = profile 2
60      uint16_t i;                 // 0-511 array index
61      uint8_t enable_output;      // 0 = disable, 1= enable | DAC output
62      uint8_t Command;            // 0 = no command received, 1 = command received
63      uint16_t minF;              // minimum frequency
64      uint16_t maxF;              // maximum frequency
65      float gain;                 // 0.000-9.999 gain factor
66  };
67
68  struct global_file_handler
69  {
70      uint8_t receive_array[42000];       // Input buffer for files
71      uint16_t file_index;                // Index for receive_array
72      uint8_t Received_Data_Buffer[12];   // Input buffer for commands
73      uint32_t Length;                    // Length of current Package
74      uint16_t DataLength;                // Data length of received file
75      uint16_t DataLength_ofActiveFile;   // Data length of active file
76      uint8_t OnGoing_FileTransfer;       // 0 = no file transfer, 1 = file transfer
77      int HeaderLength;                   // Header Length of received file
78      int HeaderLength_ofActiveFile;      // Header Length of active file
79      uint64_t tick;                      // ticks since last package reveived
80      uint64_t timeout;                   // ticks to file transfer timeout
81  };
82
83  /********************************** Private variables
    ***************************************/
84  SPI_HandleTypeDef hspi1;
85  SPI_HandleTypeDef hspi3;
86  SPI_HandleTypeDef hspi4;
87  SPI_HandleTypeDef hspi5;
88
89  TIM_HandleTypeDef htim1;
90
91  struct wave wave_matrix[512];
92  struct global_control_handler G_hControl;
93  struct global_file_handler G_hFile;
94
95  /****************************** Private function prototypes
    ***********************************/
96  void SystemClock_Config(void);
97  static void MX_GPIO_Init(void);
98  static void MX_SPI1_Init(void);
99  static void MX_SPI3_Init(void);
100 static void MX_SPI4_Init(void);
101 static void MX_SPI5_Init(void);
102 static void MX_TIM1_Init(void);
103
104 void select_DAC_1_to_4();
105 void deselect_DAC_1_to_4();
106 void select_DAC_5_to_8();
107 void deselect_DAC_5_to_8();
108 void write_SPI_DAC_1_to_4();
109 void write_SPI_DAC_5_to_8();
110 void wait_SPI_transmit();
111 void set_DAC_output_to_zero();
112
```

main.c

```
113 void create_sin_wave();
114 void recieve_wave_file();
115 void update_wave_file();
116 int file_integrety_test();
117 void sort_received_file();
118 void USB_command_handler();
119 void timeout_check();
120
121 /*******************************************************************************
    *******/
122 /*                          @brief  The application entry
    point                                */
123 /*******************************************************************************
    *******/
124 int main(void)
125 {
126     //------------------------- Enable I-Cache
127     SCB_EnableICache();
128     //------------------------- Enable D-Cache
129     SCB_EnableDCache();
130
131     //------------------------- Reset of all peripherals, Initializes the Flash interface
    and the Systick.
132     HAL_Init();
133     //------------------------- Configure the system clock
134     SystemClock_Config();
135
136     //------------------------- Initialize all configured peripherals
137     MX_GPIO_Init();
138     MX_SPI1_Init();
139     MX_SPI3_Init();
140     MX_SPI4_Init();
141     MX_SPI5_Init();
142     MX_TIM1_Init();
143     MX_USB_DEVICE_Init();
144
145     //------------------------- Global init values
146     G_hFile.HeaderLength = 0;
147     G_hFile.HeaderLength_ofActiveFile = 0;
148     G_hFile.DataLength = 0;
149     G_hFile.DataLength_ofActiveFile = 0;
150     G_hFile.Length = 0;                 // Package Length for USB commands
151     G_hFile.file_index = 0;             // file index for package reception
152     G_hFile.OnGoing_FileTransfer = 0;   // is set on file receive. is reset on success or
    failure
153     G_hFile.timeout = 1000000;          // cycle count until file reception is canceled
154
155     G_hControl.Command = 0;             // gets set on USB command receive
156     G_hControl.frequency = 500;         // current frequency
157     G_hControl.wave_source = 0;         //
158     G_hControl.i = 0;                   // wave index
159     G_hControl.enable_output=0;         // enable DAC
160     G_hControl.minF = 100;              // Frequency min 100 = 10 Hz
161     G_hControl.maxF = 2000;             // Frequency max 2000 = 200 Hz
162     G_hControl.gain = 1;                // gain 0.00 - 9.99
163
164
165     //------------------------- initialize default wave form
166     create_sin_wave();
167
168     //------------------------- Reset DAC output
169     set_DAC_output_to_zero();
```

main.c

```
170
171     /********************************** Infinite loop
    ************************************/
172     while (1)
173     {
174         /****************************** Control DAC output
    ********************************/
175         if (G_hControl.enable_output)
176         {
177             //------------------ Transmit DAC 1-4
178             select_DAC_1_to_4();            // CSS low
179             write_SPI_DAC_1_to_4();         // SPI data transmit
180             wait_SPI_transmit();            // wait for SPI EOT flags
181             deselect_DAC_1_to_4();          // CSS high
182
183             //------------------ Transmit DAC 5-8
184             select_DAC_5_to_8();            // CSS low
185             write_SPI_DAC_5_to_8();         // SPI data transmit
186             wait_SPI_transmit();            // wait for SPI EOT flags
187             deselect_DAC_5_to_8();          // CSS high
188
189             //------------------ update DAC Output
190             while (!(htim1.Instance->SR&TIM_SR_UIF));                   // wait for timer
191             LDAC_GPIO_Port->BSRR = (uint32_t)LDAC_Pin << GPIO_NUMBER;   // LDAC Pin Low
192             htim1.Instance->SR &= ~TIM_SR_UIF;                          // reset flag
193             if ( G_hControl.i < 512 ) G_hControl.i++;                   // increment wave
    index
194             LDAC_GPIO_Port->BSRR = LDAC_Pin;                            // LDAC Pin High
195             if ( G_hControl.i >= 512) G_hControl.i = 0;                 // reset wave
    index
196         }
197
198         /************************ check for file transfer timeout
    ******************************/
199         else if(G_hFile.OnGoing_FileTransfer) timeout_check();
200
201         /************** react to USB commands - received in usb_rXcallback()
    ********************/
202         if (G_hControl.Command) USB_command_handler();
203     }
204 }
205
206 /*****************************************************************************
    *******/
207 /*              @brief  function interprets a received control
    command                             */
208 /*****************************************************************************
    *******/
209 void USB_command_handler()
210 {
211     float period;                       // Timer period calculation, float to round
    correctly
212     uint16_t f = 500;                   // DAC output frequency (minF <= XXX <= maxF)
213
214 /*************************** Start/Stop command ***********************************/
215     if((G_hFile.Length == 1)
216             && ((G_hFile.Received_Data_Buffer[0] == 'S')
217             || (G_hFile.Received_Data_Buffer[0] == 's')))
218     {
219         if(!G_hFile.OnGoing_FileTransfer) // command not accepted during file reception
220         {
221             //------------- toggle Start/Stop
```

main.c

```
222             G_hControl.enable_output++;
223             if (G_hControl.enable_output>1)
224             {
225                 G_hControl.enable_output = 0;
226                 set_DAC_output_to_zero();
227                 LED1_GPIO_Port->BSRR = (uint32_t)LED1_Pin << GPIO_NUMBER;   // LDAC Pin
    Low
228                 CDC_Transmit_FS((uint8_t*) "Output stopped\n", 15);
229             }
230             LED1_GPIO_Port->BSRR = LDAC_Pin;                            // LDAC Pin High
231             CDC_Transmit_FS((uint8_t*) "Output started\n", 15);
232         }
233         else
234         {
235             //------------- error message never reached, all data interpreted as file
    while OnGoing_FileTransfer
236             CDC_Transmit_FS((uint8_t*) "Can not start output while ongoing file
    transfer\n", 49);
237         }
238     }
239     else
240     {
241
242 /************************ Frequency change command **********************************/
243         if((G_hFile.Received_Data_Buffer[0] == 'F') || (G_hFile.Received_Data_Buffer[0] ==
    'f'))
244         {
245             if (G_hFile.Length == 5 || G_hFile.Length == 6)
246             {
247                 if(G_hFile.Length == 5)
248                 {
249                     //-------------- ASCII to int
250                     f = (G_hFile.Received_Data_Buffer[2] - 48) * 100
251                             + (G_hFile.Received_Data_Buffer[3] - 48) * 10
252                             + (G_hFile.Received_Data_Buffer[4] - 48);
253                 }
254                 else
255                 {
256                     //-------------- ASCII to int
257                     f = (G_hFile.Received_Data_Buffer[2] - 48) * 1000
258                             + (G_hFile.Received_Data_Buffer[3] - 48) * 100
259                             + (G_hFile.Received_Data_Buffer[4] - 48) * 10
260                             + (G_hFile.Received_Data_Buffer[5] - 48);
261                 }
262                 //-------------- Set new frequency
263                 if(f>=G_hControl.minF && f<=G_hControl.maxF)
264                 {
265                     if(G_hControl.frequency == f)
266                     {
267                         CDC_Transmit_FS((uint8_t*)"Frequency already selected\n",27);
268                     }
269                     else
270                     {
271                         //-------------- Translate new frequency to timer period
272                         G_hControl.frequency = f;
273                         period = ((TIM1_CLK/(512.0*G_hControl.frequency/10)));      //
    calculate new cnt
274                         if ((period-(uint16_t)period)>=0.5) period++;   // round cnt
275                         htim1.Instance->ARR = (unsigned)period-1;
276                         CDC_Transmit_FS((uint8_t*)"Frequency change accepted\n",26);
277                     }
278                 }
```

main.c

```
279                 else
280                 {
281                     CDC_Transmit_FS((uint8_t*)"Frequency out of range (100-2000)\n",34);
282                 }
283             }
284             else
285             {
286                 CDC_Transmit_FS((uint8_t*)"Frequency out of range (100-2000)\n",34);
287             }
288         }
289         else
290         {
291 
292 /********************** Wave change command (w=1 or w=2)
    ************************************/
293             if(G_hFile.Received_Data_Buffer[0] == 'W' || G_hFile.Received_Data_Buffer[0]
    == 'w')
294             {
295                 if((G_hFile.Length == 3) && (((G_hFile.Received_Data_Buffer[2]-48) == 1)
    || ((G_hFile.Received_Data_Buffer[2]-48) == 2)))
296                 {
297                     if(G_hControl.wave_source == (G_hFile.Received_Data_Buffer[2]-48-1))
298                     {
299                         CDC_Transmit_FS((uint8_t*)"Wave already selected\n",22);
300                     }
301                     else
302                     {
303                         //-------------- Assign new wave source
304                         G_hControl.wave_source = G_hFile.Received_Data_Buffer[2]-48-1;
305                         CDC_Transmit_FS((uint8_t*)"Wave change accepted\n",21);
306                     }
307                 }
308                 else
309                 {
310                     CDC_Transmit_FS((uint8_t*)"Invalid Wave command (w=1 or w=2)\n",34);
311                 }
312             }
313             else
314             {
315 /********************** adds a gain to the output (gain)
    ****************************************/
316                 if((G_hFile.Length == 10)
317                         && (((G_hFile.Received_Data_Buffer[0] == 'G') ||
    (G_hFile.Received_Data_Buffer[0] == 'g'))
318                         && ((G_hFile.Received_Data_Buffer[1] == 'A') ||
    (G_hFile.Received_Data_Buffer[1] == 'a'))
319                         && ((G_hFile.Received_Data_Buffer[2] == 'I') ||
    (G_hFile.Received_Data_Buffer[2] == 'i'))
320                         && ((G_hFile.Received_Data_Buffer[3] == 'N') ||
    (G_hFile.Received_Data_Buffer[3] == 'n'))
321                         && ((G_hFile.Received_Data_Buffer[4] == '=') ||
    (G_hFile.Received_Data_Buffer[4] == '='))
322                         && ((G_hFile.Received_Data_Buffer[6] == '.') ||
    (G_hFile.Received_Data_Buffer[6] == '.'))))
323                 {
324                     float gain = 0;
325                     unsigned err = 0;
326                     //-------------- ASCII to int
327                     gain = (G_hFile.Received_Data_Buffer[5] - 48) *1000
328                             + (G_hFile.Received_Data_Buffer[7] - 48) *100
329                             + (G_hFile.Received_Data_Buffer[8] - 48) *10
330                             + (G_hFile.Received_Data_Buffer[9] - 48);
```

main.c

```
331                     gain /= 1000.0;
332                     if(gain >= 0 && gain < 10)
333                     {
334                         for(unsigned i = 0; i < 511; i++) // check if gain*wave exceeds limits
335                         {
336                             if (2048+(int)(gain*(wave_matrix[i].out_11-2048)) > 4095
337                                                                 || 2048+(int)(gain*(wave_matrix[i].out_11-2048)) < 0) err++;
338                             if (2048+(int)(gain*(wave_matrix[i].out_12-2048)) > 4095
339                                                                 || 2048+(int)(gain*(wave_matrix[i].out_12-2048)) < 0) err++;
340                             if (2048+(int)(gain*(wave_matrix[i].out_13-2048)) > 4095
341                                                                 || 2048+(int)(gain*(wave_matrix[i].out_13-2048)) < 0) err++;
342                             if (2048+(int)(gain*(wave_matrix[i].out_14-2048)) > 4095
343                                                                 || 2048+(int)(gain*(wave_matrix[i].out_14-2048)) < 0) err++;
344
345                             if (2048+(int)(gain*(wave_matrix[i].out_15-2048)) > 4095
346                                                                 || 2048+(int)(gain*(wave_matrix[i].out_15-2048)) < 0) err++;
347                             if (2048+(int)(gain*(wave_matrix[i].out_16-2048)) > 4095
348                                                                 || 2048+(int)(gain*(wave_matrix[i].out_16-2048)) < 0) err++;
349                             if (2048+(int)(gain*(wave_matrix[i].out_17-2048)) > 4095
350                                                                 || 2048+(int)(gain*(wave_matrix[i].out_17-2048)) < 0) err++;
351                             if (2048+(int)(gain*(wave_matrix[i].out_18-2048)) > 4095
352                                                                 || 2048+(int)(gain*(wave_matrix[i].out_18-2048)) < 0) err++;
353
354                             if (2048+(int)(gain*(wave_matrix[i].out_21-2048)) > 4095
355                                                                 || 2048+(int)(gain*(wave_matrix[i].out_21-2048)) < 0) err++;
356                             if (2048+(int)(gain*(wave_matrix[i].out_22-2048)) > 4095
357                                                                 || 2048+(int)(gain*(wave_matrix[i].out_22-2048)) < 0) err++;
358                             if (2048+(int)(gain*(wave_matrix[i].out_23-2048)) > 4095
359                                                                 || 2048+(int)(gain*(wave_matrix[i].out_23-2048)) < 0) err++;
360                             if (2048+(int)(gain*(wave_matrix[i].out_24-2048)) > 4095
361                                                                 || 2048+(int)(gain*(wave_matrix[i].out_24-2048)) < 0) err++;
362
363                             if (2048+(int)(gain*(wave_matrix[i].out_25-2048)) > 4095
364                                                                 || 2048+(int)(gain*(wave_matrix[i].out_25-2048)) < 0) err++;
365                             if (2048+(int)(gain*(wave_matrix[i].out_26-2048)) > 4095
366                                                                 || 2048+(int)(gain*(wave_matrix[i].out_26-2048)) < 0) err++;
367                             if (2048+(int)(gain*(wave_matrix[i].out_27-2048)) > 4095
368                                                                 || 2048+(int)(gain*(wave_matrix[i].out_27-2048)) < 0) err++;
369                             if (2048+(int)(gain*(wave_matrix[i].out_28-2048)) > 4095
370                                                                 || 2048+(int)(gain*(wave_matrix[i].out_28-2048)) < 0) err++;
371                         }
372                         if (err>0)
373                         {
374                             CDC_Transmit_FS((uint8_t*) "Gain to high for current setup\n", 31);
```

main.c

```
375                         }
376                         else
377                         {
378                             G_hControl.gain = gain;
379                         }
380                     }
381                     else
382                     {
383                         CDC_Transmit_FS((uint8_t*) "Gain out of range (0 10)\n", 25);
384                     }
385                     //-------------- Set new frequency
386                     if(f>=G_hControl.minF && f<=G_hControl.maxF)
387                     CDC_Transmit_FS((uint8_t*) "New Gain set\n", 13);
388                 }
389                 else
390                 {
391 /********************** send identity (status) ***************************************/
392 
393                     if((G_hFile.Length == 6 || G_hFile.Length == 7 || G_hFile.Length == 8)
394                             && (((G_hFile.Received_Data_Buffer[0] == 'S') ||
    (G_hFile.Received_Data_Buffer[0] == 's'))
395                             && ((G_hFile.Received_Data_Buffer[1] == 'T') ||
    (G_hFile.Received_Data_Buffer[1] == 't'))
396                             && ((G_hFile.Received_Data_Buffer[2] == 'A') ||
    (G_hFile.Received_Data_Buffer[2] == 'a'))
397                             && ((G_hFile.Received_Data_Buffer[3] == 'T') ||
    (G_hFile.Received_Data_Buffer[3] == 't'))
398                             && ((G_hFile.Received_Data_Buffer[4] == 'U') ||
    (G_hFile.Received_Data_Buffer[4] == 'u'))
399                             && ((G_hFile.Received_Data_Buffer[5] == 'S') ||
    (G_hFile.Received_Data_Buffer[5] == 's'))))
400                     {
401                         CDC_Transmit_FS((uint8_t*) "SymaCron\nFirmware v1.0\nUnit 1\n",
    30);
402                     }
403                     else
404                     {
405 
406 /********************** load standard sin wave (lsin)
    ***************************************/
407                         if(G_hFile.Length == 4 && (((G_hFile.Received_Data_Buffer[0] ==
    'L') || (G_hFile.Received_Data_Buffer[0] == 'l'))
408                                 && ((G_hFile.Received_Data_Buffer[1] == 'S') ||
    (G_hFile.Received_Data_Buffer[1] == 's'))
409                                 && ((G_hFile.Received_Data_Buffer[2] == 'I') ||
    (G_hFile.Received_Data_Buffer[2] == 'i'))
410                                 && ((G_hFile.Received_Data_Buffer[3] == 'N') ||
    (G_hFile.Received_Data_Buffer[3] == 'n'))))
411                         {
412                             create_sin_wave();
413                             G_hFile.DataLength_ofActiveFile = 0;
414                             G_hFile.HeaderLength_ofActiveFile = 0;
415                             CDC_Transmit_FS((uint8_t*) "standard wave setup loaded\n",
    27);
416                         }
417                         else
418                         {
419 
420 /********************** send possible commands (help)
    ***************************************/
421                             if(G_hFile.Length == 4 && (((G_hFile.Received_Data_Buffer[0]
    == 'H') || (G_hFile.Received_Data_Buffer[0] == 'h'))
```

main.c

```
422                                     && ((G_hFile.Received_Data_Buffer[1] == 'E') ||
    (G_hFile.Received_Data_Buffer[1] == 'e'))
423                                     && ((G_hFile.Received_Data_Buffer[2] == 'L') ||
    (G_hFile.Received_Data_Buffer[2] == 'l'))
424                                     && ((G_hFile.Received_Data_Buffer[3] == 'P') ||
    (G_hFile.Received_Data_Buffer[3] == 'p'))))
425                                 {
426                                     CDC_Transmit_FS((uint8_t*) "Available commands\n\ns:
    start/stop\nw=x:    select wave      [x=1 or x=2]\nf=x:    select frequency
    [100>=x>=2000]\ngain=x:    select gain [0.000>=x>=9.999]\nlsin:   load standard sine
    wave\nstatus: Device Identification\nhelp:   list of commands\n", 245);
427                                 }
428                                 else
429                                 {
430                                     // -------------- Command not recognized
431                                     CDC_Transmit_FS((uint8_t*) "Invalid command\n", 16);
432                                 }
433                             }
434                         }
435                     }
436                 }
437             }
438     }
439
440 /********************************** Reset ***************************************/
441     // Reset Received_Data_Buffer
442     G_hFile.Length = 0;
443     for (uint16_t index = 0; index < 5; index++)
444     {
445         G_hFile.Received_Data_Buffer[index] = 0;
446     }
447     //-------------- Reset command
448     G_hControl.Command = 0;
449 }
450
451 /*****************************************************************************
    *******/
452 /*              @brief  function checks for ongoing file transfer
    timeout                         */
453 /*****************************************************************************
    *******/
454 void timeout_check()
455 {
456     G_hFile.tick++;
457     if(G_hFile.tick > G_hFile.timeout)
458     {
459         CDC_Transmit_FS((uint8_t*) "ERROR: TIMEOUT\n", 15);
460         G_hFile.OnGoing_FileTransfer = 0;
461         G_hFile.file_index = 0;
462         G_hFile.HeaderLength = 0;
463         G_hFile.DataLength = 0;
464     }
465 }
466
467 /*****************************************************************************
    *******/
468 /*                          @brief  USB data receive call
    back                                    */
469 /*****************************************************************************
    *******/
470 void usb_rXcallback(uint8_t* Buf, uint32_t *Len)
471 {
```

main.c

```
472     int error = 0;
473 
474     /************************** handle ongoing data packages.
    *********************************/
475     if(G_hFile.OnGoing_FileTransfer)
476     {
477         //-------------- reset timeout tick with new package
478         G_hFile.tick = 0;
479 
480         //-------------- Copy data into received file storage
481         for(uint16_t index = 0; index < *Len; index++)
482         {
483             *(G_hFile.receive_array + G_hFile.file_index) = *(Buf + index);
484             G_hFile.file_index++;
485         }
486 
487         //-------------- Last package received
488         if(G_hFile.file_index == G_hFile.DataLength + G_hFile.HeaderLength)
489         {
490             //-------------- transform received file into desired format
491             sort_received_file();
492 
493             //-------------- Test received file for errors
494             error = file_integrety_test();
495 
496             //-------------- received file ok
497             if(error == 0)
498             {
499                 //-------------- assign data to wave_matrix
500                 update_wave_file();
501 
502                 //-------------- Check for error
503                 if(G_hFile.OnGoing_FileTransfer) //is set to zero on error in
    update_wave_file()
504                 {
505                     G_hFile.file_index = 0;
506                     G_hFile.OnGoing_FileTransfer = 0;
507                     G_hFile.HeaderLength_ofActiveFile = G_hFile.HeaderLength;
508                     G_hFile.DataLength_ofActiveFile = G_hFile.DataLength;
509                     G_hFile.HeaderLength = 0;
510                     G_hFile.DataLength = 0;
511 
512                     CDC_Transmit_FS((uint8_t*)"File accepted\n", 14);
513                 }
514                 else
515                 {
516                     //-------------- load standard sine
517                     create_sin_wave();
518                     CDC_Transmit_FS((uint8_t*)"File corrupted.\n    Value out of range or
    not identifiable.\n    Standard sine loaded\n", 86);
519                     G_hFile.OnGoing_FileTransfer = 0;
520                     G_hFile.file_index = 0;
521                     G_hFile.HeaderLength = 0;
522                     G_hFile.DataLength = 0;
523                 }
524             }
525             //-------------- received file corrupted
526             else
527             {
528                 if(error == -1) CDC_Transmit_FS((uint8_t*)"File corrupted.\n    Value out
    of range or not identifiable.\n", 60);
529                 else
```

main.c

```
530                 {
531                     if(error == -2) CDC_Transmit_FS((uint8_t*)"File corrupted.\n    Header
    format false or data length out of range.\n", 69);
532                     else
533                     {
534                         if(error == -3) CDC_Transmit_FS((uint8_t*)"File corrupted.\n
    Header format false or header length out of range.\n", 72);
535                         else
536                         {
537                             if(error == -4) CDC_Transmit_FS((uint8_t*)"File corrupted.\n
    Header format false.\n", 41);
538                             else
539                             {
540                                 if(error == -5) CDC_Transmit_FS((uint8_t*)"File
    corrupted.\n    Incorrect number of Datapoints.\n",52);
541                                 else
542                                 {
543                                     CDC_Transmit_FS((uint8_t*)"File corrupted.\n
    Unknown error occurred\n", 42);
544                                 }
545                             }
546                         }
547                     }
548                 }
549                 G_hFile.OnGoing_FileTransfer = 0;
550                 G_hFile.file_index = 0;
551                 G_hFile.HeaderLength = 0;
552                 G_hFile.DataLength = 0;
553             }
554         }
555         else
556         {
557             if(G_hFile.file_index > G_hFile.DataLength + G_hFile.HeaderLength)
558             {
559                 G_hFile.OnGoing_FileTransfer = 0;
560                 G_hFile.file_index = 0;
561                 G_hFile.HeaderLength = 0;
562                 G_hFile.DataLength = 0;
563                 CDC_Transmit_FS((uint8_t*)"File corrupted.\n    Unexpected File length\n",
    43);
564             }
565         }
566     }
567     else
568     {
569     /**************** If a command is received, the data gets processed in main
    ********************/
570         if((*Len > 0) && (*Len <= 12))
571         {
572             //-------------- Copy package to Received_Data_Buffer
573             for(uint16_t index = 0; index < 12; index++)
574             {
575                 G_hFile.Received_Data_Buffer[index] = Buf[index];
576             }
577             G_hFile.Length = *Len;
578             G_hControl.Command = 1;
579         }
580         else
581         {
582     /************************check and handle file reception
    start*****************************/
583             if((*Len >= 34) //-------------- first file package has to exceed a length of
```

main.c

```
   34 to be recognized
584                 && ( Buf[0] == 'W' )
585                 && ( Buf[1] == 'A' )
586                 && ( Buf[2] == 'V' )
587                 && ( Buf[3] == 'E' )
588                 && ( Buf[4] == ' ' )
589                 && ( Buf[5] == 'F' )
590                 && ( Buf[6] == 'I' )
591                 && ( Buf[7] == 'L' )
592                 && ( Buf[8] == 'E' ))
593             {
594                 //-------------- Read Header Length
595                 uint16_t i = 9;
596                 while(Buf[i] == '\n' || Buf[i] == '\r' || Buf[i] == ';') i++; //skip to
   next line
597
598                 if(
599                     Buf[i] == 'H'
600                     && Buf[i+1] == 'e'
601                     && Buf[i+2] == 'a'
602                     && Buf[i+3] == 'd'
603                     && Buf[i+4] == 'e'
604                     && Buf[i+5] == 'r'
605                     && Buf[i+6] == ':'
606                     && Buf[i+7] == ' ')
607                 {
608                     char hlen[2];
609                     char flen[5];
610
611                     hlen[0] = *(Buf + i + 8);
612                     hlen[1] = *(Buf + i + 9);
613                     G_hFile.HeaderLength = (hlen[0]-48)*10+(hlen[1]-48);
614
615                     //------------ Read data Length
616                     i=i+10;
617                     while(Buf[i] == '\n' || Buf[i] == '\r' || Buf[i] == ';') i++; //skip
   to next line
618
619                     if(
620                         Buf[i] == 'D'
621                         && Buf[i+1] == 'a'
622                         && Buf[i+2] == 't'
623                         && Buf[i+3] == 'a'
624                         && Buf[i+4] == ':'
625                         && Buf[i+5] == ' ')
626                     {
627                         flen[0] = Buf[i+6];
628                         flen[1] = Buf[i+7];
629                         flen[2] = Buf[i+8];
630                         flen[3] = Buf[i+9];
631                         flen[4] = Buf[i+10];
632
633                         G_hFile.DataLength = (flen[0]-48)*10000
634                                 + (flen[1]-48)*1000
635                                 + (flen[2]-48)*100
636                                 + (flen[3]-48)*10
637                                 + (flen[4]-48)*1;
638
639                         //------------- Start reading File
640                         if(G_hFile.DataLength > 512*16 && G_hFile.DataLength <
   42000-G_hFile.HeaderLength)
641                         {
```

main.c

```
642                             G_hFile.file_index = 0;
643
644                             // loop over package
645                             while(G_hFile.file_index < *Len)
646                             {
647                                 //store data to ram
648                                 *(G_hFile.receive_array + G_hFile.file_index) = *(Buf +
    G_hFile.file_index);
649                                 G_hFile.file_index++;
650                             }
651
652                             //----------- enable receiving mode stop output
653                             CDC_Transmit_FS((uint8_t*)"Receiving file\n", 15);
654                             set_DAC_output_to_zero();
655                             G_hFile.OnGoing_FileTransfer = 1;
656                             G_hControl.enable_output = 0;
657                             G_hFile.tick = 0;
658                         }
659                         else
660                         {
661                             CDC_Transmit_FS((uint8_t*)"ERROR: Insufficient data length\n",
    32);
662                             G_hFile.file_index=0;
663
664                         }
665                     }
666                     else
667                     {
668                         CDC_Transmit_FS((uint8_t*)"ERROR: Unable to identify file
    length\n", 38);
669                         G_hFile.file_index=0;
670                     }
671                 }
672                 else
673                 {
674                     CDC_Transmit_FS((uint8_t*)"ERROR: Unable to identify header length\n",
    40);
675                     G_hFile.file_index=0;
676                 }
677             }
678             else
679             {
680                 CDC_Transmit_FS((uint8_t*)"Unknown data package received\n", 30);
681                 G_hFile.file_index=0;
682             }
683         }
684     }
685 }
686
687 /*******************************************************************************
    *******/
688 /*                  @brief  changes newline to ";" in received array
                                */
689 /*******************************************************************************
    *******/
690 void sort_received_file()
691 {
692     //----------- loop through received file
693     for(uint16_t index = 0; index < G_hFile.DataLength + G_hFile.HeaderLength; index++)
694     {
695         //----------- if nextline change to ';'
696         if( G_hFile.receive_array[index] =='\n' || G_hFile.receive_array[index] =='\r')
```

main.c

```
697         {
698             //----------- if ';' before nextline delete
699             if( G_hFile.receive_array[index-1] == ';')
700             {
701                 for(uint16_t index2 = index; index2 < 41998; index2++)
702                 {
703                     G_hFile.receive_array[index2] = G_hFile.receive_array[index2+1];
704                 }
705                 index--;
706 
707                 //----------- decrement file length when deleted
708                 if(index > G_hFile.HeaderLength-1)
709                 {
710                     G_hFile.DataLength--;
711                 }
712                 else
713                 {
714                     G_hFile.HeaderLength--;
715                 }
716             }
717             else
718             {
719                 G_hFile.receive_array[index] = ';';
720             }
721         }
722     }
723 }
724 
725 /*******************************************************************************************
    *******/
726 /*                          @brief search format errors in
    file                                    */
727 /*******************************************************************************************
    *******/
728 int file_integrety_test()
729 {
730     char hlen[2];
731     char flen[5];
732     uint16_t hnumber = 0;
733     uint16_t fnumber = 0;
734     uint8_t valueLength = 0;
735     uint16_t valueCount = 0;
736 
737     //----------- Validate header integrity
738     if(( G_hFile.receive_array[0] == 'W' )
739     && ( G_hFile.receive_array[1] == 'A' )
740     && ( G_hFile.receive_array[2] == 'V' )
741     && ( G_hFile.receive_array[3] == 'E' )
742     && ( G_hFile.receive_array[4] == ' ' )
743     && ( G_hFile.receive_array[5] == 'F' )
744     && ( G_hFile.receive_array[6] == 'I' )
745     && ( G_hFile.receive_array[7] == 'L' )
746     && ( G_hFile.receive_array[8] == 'E' )
747     && ( G_hFile.receive_array[9] == ';' )
748     && ( G_hFile.receive_array[10] == 'H' )
749     && ( G_hFile.receive_array[11] == 'e' )
750     && ( G_hFile.receive_array[12] == 'a' )
751     && ( G_hFile.receive_array[13] == 'd' )
752     && ( G_hFile.receive_array[14] == 'e' )
753     && ( G_hFile.receive_array[15] == 'r' )
754     && ( G_hFile.receive_array[16] == ':' )
755     && ( G_hFile.receive_array[17] == ' ' ))
```

main.c

```
756     {
757         //----------- store header length
758         hlen[0] = G_hFile.receive_array[18];
759         hlen[1] = G_hFile.receive_array[19];
760         hnumber = (hlen[0]-48)*10 + (hlen[1]-48)*1;
761 
762         //----------- validate header integrity
763         if (hnumber > 40 && hnumber < 100
764         && ( G_hFile.receive_array[20] == ';' )
765         && ( G_hFile.receive_array[21] == 'D' )
766         && ( G_hFile.receive_array[22] == 'a' )
767         && ( G_hFile.receive_array[23] == 't' )
768         && ( G_hFile.receive_array[24] == 'a' )
769         && ( G_hFile.receive_array[25] == ':' )
770         && ( G_hFile.receive_array[26] == ' ' ))
771         {
772             //----------- store data length
773             flen[0] = G_hFile.receive_array[27];
774             flen[1] = G_hFile.receive_array[28];
775             flen[2] = G_hFile.receive_array[29];
776             flen[3] = G_hFile.receive_array[30];
777             flen[4] = G_hFile.receive_array[31];
778 
779             fnumber = (flen[0]-48)*10000
780                     + (flen[1]-48)*1000
781                     + (flen[2]-48)*100
782                     + (flen[3]-48)*10
783                     + (flen[4]-48)*1;
784 
785             //-----------  validate header integrity
786             if (fnumber > 16383 && fnumber < 40961
787             && ( G_hFile.receive_array[32] == ';' )
788             && ( G_hFile.receive_array[33] == 'N' )
789             && ( G_hFile.receive_array[34] == 'a' )
790             && ( G_hFile.receive_array[35] == 'm' )
791             && ( G_hFile.receive_array[36] == 'e' )
792             && ( G_hFile.receive_array[37] == ':' )
793             && ( G_hFile.receive_array[38] == ' ' ))
794             {
795                 for(uint16_t index = G_hFile.HeaderLength; index <
    G_hFile.HeaderLength+G_hFile.DataLength; index += valueLength + 1)
796                 {
797                     //----------- get value length
798                     valueLength = 0;
799                     while(G_hFile.receive_array[index + valueLength] != ';') valueLength+
    +;
800 
801                     if(valueLength>4 || valueLength<1)
802                     {
803                         //----------- ERROR: value out of range or not identifiable
804                         return -1;
805                     }
806                     else
807                     {
808                         valueCount++;
809                     }
810                 }
811                 if (valueCount!=8192)
812                 {
813                     //----------- ERROR: not enough values read
814                     return -5;
815                 }
```

main.c

```
816             }
817             else
818             {
819                 //----------- ERROR: header format or data length
820                 return -2;
821             }
822         }
823         else
824         {
825             //----------- ERROR: header format or header length
826             return -3;
827         }
828     }
829     else
830     {
831         //----------- ERROR: header format
832         return -4;
833     }
834     //----------- pass
835     return 0;
836 }
837 /*****************************************************************************************
    *******/
838 /*                      @brief translate received file to
    Wave                                    */
839 /*****************************************************************************************
    *******/
840 void update_wave_file()
841 {
842     //----------- Data index to data start
843     G_hFile.file_index = G_hFile.HeaderLength;
844     G_hControl.gain = 1;
845     uint8_t cvalue[4] = {0};
846     uint16_t ivalue = 0;
847     uint8_t valueLength = 0;
848     uint16_t row = 0;
849     uint16_t column = 0;
850 
851     //----------- Loop through rows and columns of wave_matrix
852     for(row = 0; row < 512 ; row++)
853     {
854         for(column = 0; column < 16; column++)
855         {
856             valueLength = 0;
857             cvalue[0] = '0';
858             cvalue[1] = '0';
859             cvalue[2] = '0';
860             cvalue[3] = '0';
861 
862             //----------- get value length
863             while(G_hFile.receive_array[G_hFile.file_index + valueLength] != ';')
864             {
865                 valueLength++;
866                 if(valueLength>4 || valueLength<1)
867                 {
868                     //----------- ERROR check valuelength produces error after this
    function. standard sine will be loaded
869                     G_hFile.OnGoing_FileTransfer=0;
870                     return;
871                 }
872             }
873 
```

main.c

```
874             //----------- evaluate string data point
875             while(G_hFile.receive_array[G_hFile.file_index] != ';')
876             {
877                 cvalue[4-valueLength] = G_hFile.receive_array[G_hFile.file_index];
878                 valueLength--;
879                 G_hFile.file_index++;
880             }
881 
882             //-----------  assign int value
883             ivalue = (cvalue[0]-48)*1000
884                     +(cvalue[1]-48)*100
885                     +(cvalue[2]-48)*10
886                     +(cvalue[3]-48)*1;
887 
888             //----------- ERROR check ivalue produces error after this function. standard
    sine will be loaded
889             if(ivalue>=4096 || ivalue<0)
890             {
891                 G_hFile.OnGoing_FileTransfer=0;
892                 return;
893             }
894 
895             //----------- Assign value to matrix with DAC control bits
896             switch (column)
897             {
898             //----------- first wave data
899             case 0:
900               wave_matrix[row].out_11 = ivalue;
901               break;
902             case 1:
903               wave_matrix[row].out_12 = ivalue;
904               break;
905             case 2:
906               wave_matrix[row].out_13 = ivalue;
907               break;
908             case 3:
909               wave_matrix[row].out_14 = ivalue;
910               break;
911             case 4:
912               wave_matrix[row].out_15 = ivalue;
913               break;
914             case 5:
915               wave_matrix[row].out_16 = ivalue;
916               break;
917             case 6:
918               wave_matrix[row].out_17 = ivalue;
919               break;
920             case 7:
921               wave_matrix[row].out_18 = ivalue;
922               break;
923 
924             //----------- second wave data
925             case 8:
926               wave_matrix[row].out_21 = ivalue;
927               break;
928             case 9:
929               wave_matrix[row].out_22 = ivalue;
930               break;
931             case 10:
932               wave_matrix[row].out_23 = ivalue;
933               break;
934             case 11:
```

main.c

```
935               wave_matrix[row].out_24 = ivalue;
936               break;
937             case 12:
938               wave_matrix[row].out_25 = ivalue;
939               break;
940             case 13:
941               wave_matrix[row].out_26 = ivalue;
942               break;
943             case 14:
944               wave_matrix[row].out_27 = ivalue;
945               break;
946             case 15:
947               wave_matrix[row].out_28 = ivalue;
948               break;
949             }
950
951             G_hFile.file_index++;
952         }
953     }
954 }
955
956 /*************************************************************************************
    *******/
957 /*                          @brief  Sends new data to
    DACs                                        */
958 /*************************************************************************************
    *******/
959 void write_SPI_DAC_1_to_4()
960
961 {
962     if(G_hControl.wave_source == 0)
963     {
964         hspi1.Instance->TXDR = DAC_CONTROL_BITS+2048+(int)(G_hControl.gain*
    (wave_matrix[G_hControl.i].out_11-2048));
965         hspi1.Instance->CR1 |= SPI_CR1_CSTART;      // master start
966
967         hspi5.Instance->TXDR = DAC_CONTROL_BITS+2048+(int)(G_hControl.gain*
    (wave_matrix[G_hControl.i].out_12-2048));
968         hspi5.Instance->CR1 |= SPI_CR1_CSTART;      // master start
969
970         hspi3.Instance->TXDR = DAC_CONTROL_BITS+2048+(int)(G_hControl.gain*
    (wave_matrix[G_hControl.i].out_13-2048));
971         hspi3.Instance->CR1 |= SPI_CR1_CSTART;      // master start
972
973         hspi4.Instance->TXDR = DAC_CONTROL_BITS+2048+(int)(G_hControl.gain*
    (wave_matrix[G_hControl.i].out_14-2048));
974         hspi4.Instance->CR1 |= SPI_CR1_CSTART;      // master start
975     }
976     else
977     {
978         hspi1.Instance->TXDR = DAC_CONTROL_BITS+2048+(int)(G_hControl.gain*
    (wave_matrix[G_hControl.i].out_21-2048));
979         hspi1.Instance->CR1 |= SPI_CR1_CSTART;      // master start
980
981         hspi5.Instance->TXDR = DAC_CONTROL_BITS+2048+(int)(G_hControl.gain*
    (wave_matrix[G_hControl.i].out_22-2048));
982         hspi5.Instance->CR1 |= SPI_CR1_CSTART;      // master start
983
984         hspi3.Instance->TXDR = DAC_CONTROL_BITS+2048+(int)(G_hControl.gain*
    (wave_matrix[G_hControl.i].out_23-2048));
985         hspi3.Instance->CR1 |= SPI_CR1_CSTART;      // master start
986
```

main.c

```
 987         hspi4.Instance->TXDR = DAC_CONTROL_BITS+2048+(int)(G_hControl.gain*
     (wave_matrix[G_hControl.i].out_24-2048));
 988         hspi4.Instance->CR1 |= SPI_CR1_CSTART;      // master start
 989     }
 990 
 991 }
 992 
 993 /*************************************************************************************
     *******/
 994 /*                              @brief  Sends new data to
     DACs                                    */
 995 /*************************************************************************************
     *******/
 996 void write_SPI_DAC_5_to_8()
 997 {
 998     if(G_hControl.wave_source == 0)
 999     {
1000         hspi1.Instance->TXDR = DAC_CONTROL_BITS+2048+(int)(G_hControl.gain*
     (wave_matrix[G_hControl.i].out_15-2048));
1001         hspi1.Instance->CR1 |= SPI_CR1_CSTART;      // master start
1002 
1003         hspi5.Instance->TXDR = DAC_CONTROL_BITS+2048+(int)(G_hControl.gain*
     (wave_matrix[G_hControl.i].out_16-2048));
1004         hspi5.Instance->CR1 |= SPI_CR1_CSTART;      // master start
1005 
1006         hspi3.Instance->TXDR = DAC_CONTROL_BITS+2048+(int)(G_hControl.gain*
     (wave_matrix[G_hControl.i].out_17-2048));
1007         hspi3.Instance->CR1 |= SPI_CR1_CSTART;      // master start
1008 
1009         hspi4.Instance->TXDR = DAC_CONTROL_BITS+2048+(int)(G_hControl.gain*
     (wave_matrix[G_hControl.i].out_18-2048));
1010         hspi4.Instance->CR1 |= SPI_CR1_CSTART;      // master start
1011     }
1012     else
1013     {
1014         hspi1.Instance->TXDR = DAC_CONTROL_BITS+2048+(int)(G_hControl.gain*
     (wave_matrix[G_hControl.i].out_25-2048));
1015         hspi1.Instance->CR1 |= SPI_CR1_CSTART;      // master start
1016 
1017         hspi5.Instance->TXDR = DAC_CONTROL_BITS+2048+(int)(G_hControl.gain*
     (wave_matrix[G_hControl.i].out_26-2048));
1018         hspi5.Instance->CR1 |= SPI_CR1_CSTART;      // master start
1019 
1020         hspi3.Instance->TXDR = DAC_CONTROL_BITS+2048+(int)(G_hControl.gain*
     (wave_matrix[G_hControl.i].out_27-2048));
1021         hspi3.Instance->CR1 |= SPI_CR1_CSTART;      // master start
1022 
1023         hspi4.Instance->TXDR = DAC_CONTROL_BITS+2048+(int)(G_hControl.gain*
     (wave_matrix[G_hControl.i].out_28-2048));
1024         hspi4.Instance->CR1 |= SPI_CR1_CSTART;      // master start
1025     }
1026 
1027 }
1028 
1029 /*************************************************************************************
     *******/
1030 /*                              @brief  selects
     DACs                                   */
1031 /*************************************************************************************
     *******/
1032 void select_DAC_1_to_4()
1033 {
```

main.c

```
1034     DAC1_GPIO_Port->BSRR = (uint32_t)DAC1_Pin << GPIO_NUMBER;   // dac_1 css pin Low
1035     DAC2_GPIO_Port->BSRR = (uint32_t)DAC2_Pin << GPIO_NUMBER;   // dac_2 css pin Low
1036     DAC3_GPIO_Port->BSRR = (uint32_t)DAC3_Pin << GPIO_NUMBER;   // dac_3 css pin Low
1037     DAC4_GPIO_Port->BSRR = (uint32_t)DAC4_Pin << GPIO_NUMBER;   // dac_4 css pin Low
1038 }
1039
1040 /*************************************************************************************
     *******/
1041 /*                                  @brief  deselects
     DACs                                        */
1042 /*************************************************************************************
     *******/
1043 void deselect_DAC_1_to_4()
1044 {
1045     DAC1_GPIO_Port->BSRR = DAC1_Pin; // dac_1 css pin High
1046     DAC2_GPIO_Port->BSRR = DAC2_Pin; // dac_2 css pin High
1047     DAC3_GPIO_Port->BSRR = DAC3_Pin; // dac_3 css pin High
1048     DAC4_GPIO_Port->BSRR = DAC4_Pin; // dac_4 css pin High
1049 }
1050
1051 /*************************************************************************************
     *******/
1052 /*                              @brief  selects
     DACs                                          */
1053 /*************************************************************************************
     *******/
1054 void select_DAC_5_to_8()
1055 {
1056     DAC5_GPIO_Port->BSRR = (uint32_t)DAC5_Pin << GPIO_NUMBER;   // dac_5 css pin Low
1057     DAC6_GPIO_Port->BSRR = (uint32_t)DAC6_Pin << GPIO_NUMBER;   // dac_6 css pin Low
1058     DAC7_GPIO_Port->BSRR = (uint32_t)DAC7_Pin << GPIO_NUMBER;   // dac_7 css pin Low
1059     DAC8_GPIO_Port->BSRR = (uint32_t)DAC8_Pin << GPIO_NUMBER;   // dac_8 css pin Low
1060 }
1061
1062 /*************************************************************************************
     *******/
1063 /*                                  @brief  deselects
     DACs                                        */
1064 /*************************************************************************************
     *******/
1065 void deselect_DAC_5_to_8()
1066 {
1067     DAC5_GPIO_Port->BSRR = DAC5_Pin; // dac_5 css pin High
1068     DAC6_GPIO_Port->BSRR = DAC6_Pin; // dac_6 css pin High
1069     DAC7_GPIO_Port->BSRR = DAC7_Pin; // dac_7 css pin High
1070     DAC8_GPIO_Port->BSRR = DAC8_Pin; // dac_8 css pin High
1071 }
1072
1073 /*************************************************************************************
     *******/
1074 /*              @brief  wait for EOT flag raise for all SPI and clear the
     flag                    */
1075 /*************************************************************************************
     *******/
1076 void wait_SPI_transmit()
1077 {
1078     while(!(hspi1.Instance->SR & SPI_FLAG_EOT));
1079     while(!(hspi5.Instance->SR & SPI_FLAG_EOT));
1080     while(!(hspi3.Instance->SR & SPI_FLAG_EOT));
1081     while(!(hspi4.Instance->SR & SPI_FLAG_EOT));
1082     //------------ flag reset
1083     hspi1.Instance->IFCR |= (0x7fc);
```

main.c

```
1084    hspi5.Instance->IFCR |= (0x7fc);
1085    hspi3.Instance->IFCR |= (0x7fc);
1086    hspi4.Instance->IFCR |= (0x7fc);
1087 }
1088
1089 /*****************************************************************************
    *******/
1090 /*              @brief  write default sine and faulty sine in
    wave_matrix                       */
1091 /*****************************************************************************
    *******/
1092 void create_sin_wave()
1093 {
1094       G_hControl.enable_output = 0;
1095
1096       unsigned k0 = 0;
1097       unsigned k1 = 171;
1098       unsigned k2 = 342;
1099       unsigned k3 = 75;
1100
1101       for (k0 = 0; k0 < 512; k0++)
1102       {
1103           //-------------- Sine
1104           wave_matrix[k0].out_11 = (unsigned) ( 2048.0 * ( 1 + sin( k0 / 512.0 * 2 * M_PI
    ));
1105           wave_matrix[k0].out_12 = (unsigned) ( 2048.0 * ( 1 + sin( k1 / 512.0 * 2 * M_PI
    ));
1106           wave_matrix[k0].out_13 = (unsigned) ( 2048.0 * ( 1 + sin( k2 / 512.0 * 2 * M_PI
    ));
1107           wave_matrix[k0].out_14 = (unsigned) ( 2048.0 * ( 1 + sin( k3 / 512.0 * 2 * M_PI
    ));
1108           wave_matrix[k0].out_15 = (unsigned) ( 2048.0 * ( 1 + sin( k0 / 512.0 * 2 * M_PI
    ));
1109           wave_matrix[k0].out_16 = (unsigned) ( 2048.0 * ( 1 + sin( k1 / 512.0 * 2 * M_PI
    ));
1110           wave_matrix[k0].out_17 = (unsigned) ( 2048.0 * ( 1 + sin( k2 / 512.0 * 2 * M_PI
    ));
1111           wave_matrix[k0].out_18 = (unsigned) ( 2048.0 * ( 1 + sin( k3 / 512.0 * 2 * M_PI
    ));
1112
1113           //-------------- ramp
1114           wave_matrix[k0].out_21 = (unsigned) ( k0 * 8 );
1115           wave_matrix[k0].out_22 = (unsigned) ( k1 * 8 );
1116           wave_matrix[k0].out_23 = (unsigned) ( k2 * 8 );
1117           wave_matrix[k0].out_24 = (unsigned) ( k3 * 8 );
1118           wave_matrix[k0].out_25 = (unsigned) ( k0 * 8 );
1119           wave_matrix[k0].out_26 = (unsigned) ( k1 * 8 );
1120           wave_matrix[k0].out_27 = (unsigned) ( k2 * 8 );
1121           wave_matrix[k0].out_28 = (unsigned) ( k3 * 8 );
1122
1123           k1++;
1124           k2++;
1125           k3++;
1126           if (k1>=512) k1 = 0;
1127           if (k2>=512) k2 = 0;
1128           if (k3>=512) k3 = 0;
1129       }
1130       for(k0=0;k0<512;k0++) //-- set maxvalue to 4095
1131       {
1132           if(wave_matrix[k0].out_11 > 4095) wave_matrix[k0].out_11 = 4095;
1133           if(wave_matrix[k0].out_12 > 4095) wave_matrix[k0].out_12 = 4095;
1134           if(wave_matrix[k0].out_13 > 4095) wave_matrix[k0].out_13 = 4095;
```

main.c

```
1135          if(wave_matrix[k0].out_14 > 4095) wave_matrix[k0].out_14 = 4095;
1136          if(wave_matrix[k0].out_15 > 4095) wave_matrix[k0].out_15 = 4095;
1137          if(wave_matrix[k0].out_16 > 4095) wave_matrix[k0].out_16 = 4095;
1138          if(wave_matrix[k0].out_17 > 4095) wave_matrix[k0].out_17 = 4095;
1139          if(wave_matrix[k0].out_18 > 4095) wave_matrix[k0].out_18 = 4095;
1140
1141          if(wave_matrix[k0].out_21 > 4095) wave_matrix[k0].out_21 = 4095;
1142          if(wave_matrix[k0].out_22 > 4095) wave_matrix[k0].out_22 = 4095;
1143          if(wave_matrix[k0].out_23 > 4095) wave_matrix[k0].out_23 = 4095;
1144          if(wave_matrix[k0].out_24 > 4095) wave_matrix[k0].out_24 = 4095;
1145          if(wave_matrix[k0].out_25 > 4095) wave_matrix[k0].out_25 = 4095;
1146          if(wave_matrix[k0].out_26 > 4095) wave_matrix[k0].out_26 = 4095;
1147          if(wave_matrix[k0].out_27 > 4095) wave_matrix[k0].out_27 = 4095;
1148          if(wave_matrix[k0].out_28 > 4095) wave_matrix[k0].out_28 = 4095;
1149      }
1150
1151      //-------------- reset file length to message: no file loaded
1152      G_hControl.gain = 1;
1153      G_hFile.DataLength_ofActiveFile = 0;
1154      G_hFile.HeaderLength_ofActiveFile = 0;
1155 }
1156
1157 /*************************************************************************************
     *******/
1158 /*                      @brief  write zero to DACs and update
     output                          */
1159 /*************************************************************************************
     *******/
1160 void set_DAC_output_to_zero()
1161 {
1162     //---- select all DACs
1163     select_DAC_1_to_4();
1164     select_DAC_5_to_8();
1165
1166     //-------------- transmit zero to all DACs
1167     hspi1.Instance->TXDR = DAC_CONTROL_BITS + 2048;
1168     hspi1.Instance->CR1 |= SPI_CR1_CSTART;      // master start
1169     hspi5.Instance->TXDR = DAC_CONTROL_BITS + 2048;
1170     hspi5.Instance->CR1 |= SPI_CR1_CSTART;      // master start
1171     hspi3.Instance->TXDR = DAC_CONTROL_BITS + 2048;
1172     hspi3.Instance->CR1 |= SPI_CR1_CSTART;      // master start
1173     hspi4.Instance->TXDR = DAC_CONTROL_BITS + 2048;
1174     hspi4.Instance->CR1 |= SPI_CR1_CSTART;      // master start
1175
1176     //-------------- wait for EOT
1177     wait_SPI_transmit();
1178
1179     //-------------- deselect all DACs
1180     deselect_DAC_1_to_4();
1181     deselect_DAC_5_to_8();
1182
1183     //-------------- wait time
1184     G_hControl.i = 0; // set back wave index
1185     while(G_hControl.i < 10) G_hControl.i++;
1186     G_hControl.i = 0; // set back wave index
1187
1188     //-------------- LDAC Low
1189     LDAC_GPIO_Port->BSRR = (uint32_t)LDAC_Pin << GPIO_NUMBER;
1190
1191     //-------------- wait time
1192     while(G_hControl.i < 10) G_hControl.i++;
1193     G_hControl.i = 0; // set back wave index
```

main.c

```
1194
1195    //-------------- LDAC High
1196    LDAC_GPIO_Port->BSRR = LDAC_Pin;
1197
1198 }
1199
1200
1201 /*******************************************************************************
    *******/
1202 /*                          @brief System Clock
    Configuration                                       */
1203 /*******************************************************************************
    *******/
1204 void SystemClock_Config(void)
1205 {
1206   RCC_OscInitTypeDef RCC_OscInitStruct = {0};
1207   RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};
1208   RCC_PeriphCLKInitTypeDef PeriphClkInitStruct = {0};
1209
1210   /** Supply configuration update enable
1211   */
1212   HAL_PWREx_ConfigSupply(PWR_LDO_SUPPLY);
1213   /** Configure the main internal regulator output voltage
1214   */
1215   __HAL_PWR_VOLTAGESCALING_CONFIG(PWR_REGULATOR_VOLTAGE_SCALE1);
1216
1217   while(!__HAL_PWR_GET_FLAG(PWR_FLAG_VOSRDY)) {}
1218   /** Macro to configure the PLL clock source
1219   */
1220   __HAL_RCC_PLL_PLLSOURCE_CONFIG(RCC_PLLSOURCE_HSE);
1221   /** Initializes the CPU, AHB and APB busses clocks
1222   */
1223   RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSI48|RCC_OSCILLATORTYPE_HSE;
1224   RCC_OscInitStruct.HSEState = RCC_HSE_ON;
1225   RCC_OscInitStruct.HSI48State = RCC_HSI48_ON;
1226   RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
1227   RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSE;
1228   RCC_OscInitStruct.PLL.PLLM = 8;
1229   RCC_OscInitStruct.PLL.PLLN = 256;
1230   RCC_OscInitStruct.PLL.PLLP = 2;
1231   RCC_OscInitStruct.PLL.PLLQ = 2;
1232   RCC_OscInitStruct.PLL.PLLR = 2;
1233   RCC_OscInitStruct.PLL.PLLRGE = RCC_PLL1VCIRANGE_0;
1234   RCC_OscInitStruct.PLL.PLLVCOSEL = RCC_PLL1VCOWIDE;
1235   RCC_OscInitStruct.PLL.PLLFRACN = 0;
1236   if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
1237   {
1238     Error_Handler();
1239   }
1240   /** Initializes the CPU, AHB and APB busses clocks
1241   */
1242   RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
1243                                |RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2
1244                                |RCC_CLOCKTYPE_D3PCLK1|RCC_CLOCKTYPE_D1PCLK1;
1245   RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
1246   RCC_ClkInitStruct.SYSCLKDivider = RCC_SYSCLK_DIV1;
1247   RCC_ClkInitStruct.AHBCLKDivider = RCC_HCLK_DIV2;
1248   RCC_ClkInitStruct.APB3CLKDivider = RCC_APB3_DIV1;
1249   RCC_ClkInitStruct.APB1CLKDivider = RCC_APB1_DIV1;
1250   RCC_ClkInitStruct.APB2CLKDivider = RCC_APB2_DIV1;
1251   RCC_ClkInitStruct.APB4CLKDivider = RCC_APB4_DIV1;
1252
```

main.c

```
1253   if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_0) != HAL_OK)
1254   {
1255     Error_Handler();
1256   }
1257   PeriphClkInitStruct.PeriphClockSelection = RCC_PERIPHCLK_USART1|RCC_PERIPHCLK_SPI5
1258                               |RCC_PERIPHCLK_SPI4|RCC_PERIPHCLK_SPI3
1259                               |RCC_PERIPHCLK_SPI1|RCC_PERIPHCLK_USB;
1260   PeriphClkInitStruct.PLL3.PLL3M = 8;
1261   PeriphClkInitStruct.PLL3.PLL3N = 256;
1262   PeriphClkInitStruct.PLL3.PLL3P = 8;
1263   PeriphClkInitStruct.PLL3.PLL3Q = 8;
1264   PeriphClkInitStruct.PLL3.PLL3R = 2;
1265   PeriphClkInitStruct.PLL3.PLL3RGE = RCC_PLL3VCIRANGE_0;
1266   PeriphClkInitStruct.PLL3.PLL3VCOSEL = RCC_PLL3VCOWIDE;
1267   PeriphClkInitStruct.PLL3.PLL3FRACN = 0;
1268   PeriphClkInitStruct.Spi123ClockSelection = RCC_SPI123CLKSOURCE_PLL3;
1269   PeriphClkInitStruct.Spi45ClockSelection = RCC_SPI45CLKSOURCE_PLL3;
1270   PeriphClkInitStruct.Usart16ClockSelection = RCC_USART16CLKSOURCE_D2PCLK2;
1271   PeriphClkInitStruct.UsbClockSelection = RCC_USBCLKSOURCE_HSI48;
1272   if (HAL_RCCEx_PeriphCLKConfig(&PeriphClkInitStruct) != HAL_OK)
1273   {
1274     Error_Handler();
1275   }
1276   /** Enable USB Voltage detector
1277   */
1278   HAL_PWREx_EnableUSBVoltageDetector();
1279 }
1280
1281 /*******************************************************************************
      *******/
1282 /*                              @brief SPI1 Initialization
      Function                                  */
1283 /*******************************************************************************
      *******/
1284 static void MX_SPI1_Init(void)
1285 {
1286   /* SPI1 parameter configuration*/
1287   hspi1.Instance = SPI1;
1288   hspi1.Init.Mode = SPI_MODE_MASTER;
1289   hspi1.Init.Direction = SPI_DIRECTION_2LINES_TXONLY;
1290   hspi1.Init.DataSize = SPI_DATASIZE_16BIT;
1291   hspi1.Init.CLKPolarity = SPI_POLARITY_LOW;
1292   hspi1.Init.CLKPhase = SPI_PHASE_1EDGE;
1293   hspi1.Init.NSS = SPI_NSS_SOFT;
1294   hspi1.Init.BaudRatePrescaler = SPI_BAUDRATEPRESCALER_2;
1295   hspi1.Init.FirstBit = SPI_FIRSTBIT_MSB;
1296   hspi1.Init.TIMode = SPI_TIMODE_DISABLE;
1297   hspi1.Init.CRCCalculation = SPI_CRCCALCULATION_DISABLE;
1298   hspi1.Init.CRCPolynomial = 0x0;
1299   hspi1.Init.NSSPMode = SPI_NSS_PULSE_DISABLE;
1300   hspi1.Init.NSSPolarity = SPI_NSS_POLARITY_LOW;
1301   hspi1.Init.FifoThreshold = SPI_FIFO_THRESHOLD_01DATA;
1302   hspi1.Init.TxCRCInitializationPattern = SPI_CRC_INITIALIZATION_ALL_ZERO_PATTERN;
1303   hspi1.Init.RxCRCInitializationPattern = SPI_CRC_INITIALIZATION_ALL_ZERO_PATTERN;
1304   hspi1.Init.MasterSSIdleness = SPI_MASTER_SS_IDLENESS_00CYCLE;
1305   hspi1.Init.MasterInterDataIdleness = SPI_MASTER_INTERDATA_IDLENESS_00CYCLE;
1306   hspi1.Init.MasterReceiverAutoSusp = SPI_MASTER_RX_AUTOSUSP_DISABLE;
1307   hspi1.Init.MasterKeepIOState = SPI_MASTER_KEEP_IO_STATE_DISABLE;
1308   hspi1.Init.IOSwap = SPI_IO_SWAP_DISABLE;
1309   if (HAL_SPI_Init(&hspi1) != HAL_OK)
1310   {
1311     Error_Handler();
```

main.c

```
1312   }
1313   /* USER CODE BEGIN SPI1_Init 2 */
1314   hspi1.Instance->CR2 = 0x1;
1315   hspi1.Instance->CR1 |= SPI_CR1_SPE;           //spi enable
1316 
1317   hspi1.Instance->TXDR = DAC_CONTROL_BITS+2047;
1318   hspi1.Instance->CR1 |= SPI_CR1_CSTART;        // master start
1319 
1320   while(!(hspi1.Instance->SR & SPI_FLAG_EOT));
1321   hspi1.Instance->IFCR |= (0x7fc);
1322   /* USER CODE END SPI1_Init 2 */
1323 }
1324 
1325 /**********************************************************************************
      *******/
1326 /*                             @brief SPI3 Initialization
      Function                                  */
1327 /**********************************************************************************
      *******/
1328 static void MX_SPI3_Init(void)
1329 {
1330   /* SPI3 parameter configuration*/
1331   hspi3.Instance = SPI3;
1332   hspi3.Init.Mode = SPI_MODE_MASTER;
1333   hspi3.Init.Direction = SPI_DIRECTION_2LINES_TXONLY;
1334   hspi3.Init.DataSize = SPI_DATASIZE_16BIT;
1335   hspi3.Init.CLKPolarity = SPI_POLARITY_LOW;
1336   hspi3.Init.CLKPhase = SPI_PHASE_1EDGE;
1337   hspi3.Init.NSS = SPI_NSS_SOFT;
1338   hspi3.Init.BaudRatePrescaler = SPI_BAUDRATEPRESCALER_2;
1339   hspi3.Init.FirstBit = SPI_FIRSTBIT_MSB;
1340   hspi3.Init.TIMode = SPI_TIMODE_DISABLE;
1341   hspi3.Init.CRCCalculation = SPI_CRCCALCULATION_DISABLE;
1342   hspi3.Init.CRCPolynomial = 0x0;
1343   hspi3.Init.NSSPMode = SPI_NSS_PULSE_DISABLE;
1344   hspi3.Init.NSSPolarity = SPI_NSS_POLARITY_LOW;
1345   hspi3.Init.FifoThreshold = SPI_FIFO_THRESHOLD_01DATA;
1346   hspi3.Init.TxCRCInitializationPattern = SPI_CRC_INITIALIZATION_ALL_ZERO_PATTERN;
1347   hspi3.Init.RxCRCInitializationPattern = SPI_CRC_INITIALIZATION_ALL_ZERO_PATTERN;
1348   hspi3.Init.MasterSSIdleness = SPI_MASTER_SS_IDLENESS_00CYCLE;
1349   hspi3.Init.MasterInterDataIdleness = SPI_MASTER_INTERDATA_IDLENESS_00CYCLE;
1350   hspi3.Init.MasterReceiverAutoSusp = SPI_MASTER_RX_AUTOSUSP_DISABLE;
1351   hspi3.Init.MasterKeepIOState = SPI_MASTER_KEEP_IO_STATE_DISABLE;
1352   hspi3.Init.IOSwap = SPI_IO_SWAP_DISABLE;
1353   if (HAL_SPI_Init(&hspi3) != HAL_OK)
1354   {
1355     Error_Handler();
1356   }
1357   /* USER CODE BEGIN SPI3_Init 2 */
1358   hspi3.Instance->CR2 = 0x1;
1359   hspi3.Instance->CR1 |= SPI_CR1_SPE;           //spi enable
1360 
1361   hspi3.Instance->TXDR = DAC_CONTROL_BITS+2047;
1362   hspi3.Instance->CR1 |= SPI_CR1_CSTART;        // master start
1363 
1364   while(!(hspi3.Instance->SR & SPI_FLAG_EOT));
1365   hspi3.Instance->IFCR |= (0x7fc);
1366   /* USER CODE END SPI3_Init 2 */
1367 }
1368 
1369 /**********************************************************************************
      *******/
```

main.c

```
1370 /*                              @brief SPI4 Initialization
     Function                                 */
1371 /*****************************************************************************
     *******/
1372 static void MX_SPI4_Init(void)
1373 {
1374   /* SPI4 parameter configuration*/
1375   hspi4.Instance = SPI4;
1376   hspi4.Init.Mode = SPI_MODE_MASTER;
1377   hspi4.Init.Direction = SPI_DIRECTION_2LINES_TXONLY;
1378   hspi4.Init.DataSize = SPI_DATASIZE_16BIT;
1379   hspi4.Init.CLKPolarity = SPI_POLARITY_LOW;
1380   hspi4.Init.CLKPhase = SPI_PHASE_1EDGE;
1381   hspi4.Init.NSS = SPI_NSS_SOFT;
1382   hspi4.Init.BaudRatePrescaler = SPI_BAUDRATEPRESCALER_2;
1383   hspi4.Init.FirstBit = SPI_FIRSTBIT_MSB;
1384   hspi4.Init.TIMode = SPI_TIMODE_DISABLE;
1385   hspi4.Init.CRCCalculation = SPI_CRCCALCULATION_DISABLE;
1386   hspi4.Init.CRCPolynomial = 0x0;
1387   hspi4.Init.NSSPMode = SPI_NSS_PULSE_DISABLE;
1388   hspi4.Init.NSSPolarity = SPI_NSS_POLARITY_LOW;
1389   hspi4.Init.FifoThreshold = SPI_FIFO_THRESHOLD_01DATA;
1390   hspi4.Init.TxCRCInitializationPattern = SPI_CRC_INITIALIZATION_ALL_ZERO_PATTERN;
1391   hspi4.Init.RxCRCInitializationPattern = SPI_CRC_INITIALIZATION_ALL_ZERO_PATTERN;
1392   hspi4.Init.MasterSSIdleness = SPI_MASTER_SS_IDLENESS_00CYCLE;
1393   hspi4.Init.MasterInterDataIdleness = SPI_MASTER_INTERDATA_IDLENESS_00CYCLE;
1394   hspi4.Init.MasterReceiverAutoSusp = SPI_MASTER_RX_AUTOSUSP_DISABLE;
1395   hspi4.Init.MasterKeepIOState = SPI_MASTER_KEEP_IO_STATE_DISABLE;
1396   hspi4.Init.IOSwap = SPI_IO_SWAP_DISABLE;
1397   if (HAL_SPI_Init(&hspi4) != HAL_OK)
1398   {
1399     Error_Handler();
1400   }
1401   /* USER CODE BEGIN SPI4_Init 2 */
1402   hspi4.Instance->CR2 = 0x1;
1403   hspi4.Instance->CR1 |= SPI_CR1_SPE;           //spi enable
1404 
1405   hspi4.Instance->TXDR = DAC_CONTROL_BITS+2047;
1406   hspi4.Instance->CR1 |= SPI_CR1_CSTART;        // master start
1407 
1408   while(!(hspi4.Instance->SR & SPI_FLAG_EOT));
1409   hspi4.Instance->IFCR |= (0x7fc);
1410   /* USER CODE END SPI4_Init 2 */
1411 
1412 }
1413 
1414 /*****************************************************************************
     *******/
1415 /*                              @brief SPI5 Initialization
     Function                                 */
1416 /*****************************************************************************
     *******/
1417 static void MX_SPI5_Init(void)
1418 {
1419   /* SPI5 parameter configuration*/
1420   hspi5.Instance = SPI5;
1421   hspi5.Init.Mode = SPI_MODE_MASTER;
1422   hspi5.Init.Direction = SPI_DIRECTION_2LINES_TXONLY;
1423   hspi5.Init.DataSize = SPI_DATASIZE_16BIT;
1424   hspi5.Init.CLKPolarity = SPI_POLARITY_LOW;
1425   hspi5.Init.CLKPhase = SPI_PHASE_1EDGE;
1426   hspi5.Init.NSS = SPI_NSS_SOFT;
```

main.c

```
1427   hspi5.Init.BaudRatePrescaler = SPI_BAUDRATEPRESCALER_2;
1428   hspi5.Init.FirstBit = SPI_FIRSTBIT_MSB;
1429   hspi5.Init.TIMode = SPI_TIMODE_DISABLE;
1430   hspi5.Init.CRCCalculation = SPI_CRCCALCULATION_DISABLE;
1431   hspi5.Init.CRCPolynomial = 0x0;
1432   hspi5.Init.NSSPMode = SPI_NSS_PULSE_DISABLE;
1433   hspi5.Init.NSSPolarity = SPI_NSS_POLARITY_LOW;
1434   hspi5.Init.FifoThreshold = SPI_FIFO_THRESHOLD_01DATA;
1435   hspi5.Init.TxCRCInitializationPattern = SPI_CRC_INITIALIZATION_ALL_ZERO_PATTERN;
1436   hspi5.Init.RxCRCInitializationPattern = SPI_CRC_INITIALIZATION_ALL_ZERO_PATTERN;
1437   hspi5.Init.MasterSSIdleness = SPI_MASTER_SS_IDLENESS_00CYCLE;
1438   hspi5.Init.MasterInterDataIdleness = SPI_MASTER_INTERDATA_IDLENESS_00CYCLE;
1439   hspi5.Init.MasterReceiverAutoSusp = SPI_MASTER_RX_AUTOSUSP_DISABLE;
1440   hspi5.Init.MasterKeepIOState = SPI_MASTER_KEEP_IO_STATE_DISABLE;
1441   hspi5.Init.IOSwap = SPI_IO_SWAP_DISABLE;
1442   if (HAL_SPI_Init(&hspi5) != HAL_OK)
1443   {
1444     Error_Handler();
1445   }
1446   /* USER CODE BEGIN SPI5_Init 2 */
1447   hspi5.Instance->CR2 = 0x1;
1448   hspi5.Instance->CR1 |= SPI_CR1_SPE;           //spi enable
1449 
1450   hspi5.Instance->TXDR = DAC_CONTROL_BITS+2047;
1451   hspi5.Instance->CR1 |= SPI_CR1_CSTART;        // master start
1452 
1453   while(!(hspi5.Instance->SR & SPI_FLAG_EOT));
1454   hspi5.Instance->IFCR |= (0x7fc);
1455   /* USER CODE END SPI5_Init 2 */
1456 }
1457 
1458 /**********************************************************************************
    *******/
1459 /*                             @brief TIM1 Initialization
    Function                                  */
1460 /**********************************************************************************
    *******/
1461 static void MX_TIM1_Init(void)
1462 {
1463 
1464   /* USER CODE BEGIN TIM1_Init 0 */
1465 
1466   // calulate counting value
1467   float period = TIM1_CLK / 512.0 / 50.0;
1468   if (period - (int)period >= 0.5)
1469   {
1470       period++; // round up
1471   }
1472 
1473   /* USER CODE END TIM1_Init 0 */
1474 
1475   TIM_ClockConfigTypeDef sClockSourceConfig = {0};
1476   TIM_MasterConfigTypeDef sMasterConfig = {0};
1477 
1478   htim1.Instance = TIM1;
1479   htim1.Init.Prescaler = 1-1;
1480   htim1.Init.CounterMode = TIM_COUNTERMODE_UP;
1481   htim1.Init.Period = (uint16_t) period-1;
1482   htim1.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
1483   htim1.Init.RepetitionCounter = 0;
1484   htim1.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
1485   if (HAL_TIM_Base_Init(&htim1) != HAL_OK)
```

main.c

```
1486   {
1487     Error_Handler();
1488   }
1489   sClockSourceConfig.ClockSource = TIM_CLOCKSOURCE_INTERNAL;
1490   if (HAL_TIM_ConfigClockSource(&htim1, &sClockSourceConfig) != HAL_OK)
1491   {
1492     Error_Handler();
1493   }
1494   sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
1495   sMasterConfig.MasterOutputTrigger2 = TIM_TRGO2_RESET;
1496   sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
1497   if (HAL_TIMEx_MasterConfigSynchronization(&htim1, &sMasterConfig) != HAL_OK)
1498   {
1499     Error_Handler();
1500   }
1501   /* USER CODE BEGIN TIM1_Init 2 */
1502   htim1.Instance->CR1 |= TIM_CR1_ARPE;
1503   htim1.Instance->CR1 |= TIM_CR1_CEN;
1504   /* USER CODE END TIM1_Init 2 */
1505
1506 }
1507
1508 /*******************************************************************************
     *******/
1509 /*                              @brief GPIO Initialization
     Function                                    */
1510 /*******************************************************************************
     *******/
1511 static void MX_GPIO_Init(void)
1512 {
1513   GPIO_InitTypeDef GPIO_InitStruct = {0};
1514
1515   /* GPIO Ports Clock Enable */
1516   __HAL_RCC_GPIOE_CLK_ENABLE();
1517   __HAL_RCC_GPIOC_CLK_ENABLE();
1518   __HAL_RCC_GPIOF_CLK_ENABLE();
1519   __HAL_RCC_GPIOH_CLK_ENABLE();
1520   __HAL_RCC_GPIOA_CLK_ENABLE();
1521   __HAL_RCC_GPIOB_CLK_ENABLE();
1522   __HAL_RCC_GPIOG_CLK_ENABLE();
1523
1524   /*Configure GPIO pin Output Level */
1525   HAL_GPIO_WritePin(LED1_GPIO_Port, LED1_Pin, GPIO_PIN_RESET);
1526
1527   /*Configure GPIO pin Output Level */
1528   HAL_GPIO_WritePin(LDAC_GPIO_Port, LDAC_Pin, GPIO_PIN_SET);
1529
1530     /*Configure GPIO pin Output Level */
1531   HAL_GPIO_WritePin(GPIOE, DAC2_Pin|DAC3_Pin|DAC5_Pin|DAC7_Pin
1532                            |DAC8_Pin, GPIO_PIN_SET);
1533
1534   /*Configure GPIO pin Output Level */
1535   HAL_GPIO_WritePin(GPIOF, DAC1_Pin|DAC4_Pin|DAC6_Pin, GPIO_PIN_SET);
1536
1537   /*Configure GPIO pin : BButton_Pin */
1538   GPIO_InitStruct.Pin = BButton_Pin;
1539   GPIO_InitStruct.Mode = GPIO_MODE_IT_RISING;
1540   GPIO_InitStruct.Pull = GPIO_NOPULL;
1541   HAL_GPIO_Init(BButton_GPIO_Port, &GPIO_InitStruct);
1542
1543   /*Configure GPIO pin : LED1_Pin */
1544   GPIO_InitStruct.Pin = LED1_Pin;
```

main.c

```
1545   GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
1546   GPIO_InitStruct.Pull = GPIO_NOPULL;
1547   GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
1548   HAL_GPIO_Init(LED1_GPIO_Port, &GPIO_InitStruct);
1549
1550   /*Configure GPIO pin : LDAC_Pin DAC1_Pin DAC4_Pin DAC6_Pin */
1551   GPIO_InitStruct.Pin = LDAC_Pin|DAC1_Pin|DAC4_Pin|DAC6_Pin;
1552   GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
1553   GPIO_InitStruct.Pull = GPIO_NOPULL;
1554   GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_VERY_HIGH;
1555   HAL_GPIO_Init(GPIOF, &GPIO_InitStruct);
1556
1557   /*Configure GPIO pins : DAC2_Pin DAC3_Pin DAC5_Pin DAC7_Pin DAC8_Pin */
1558   GPIO_InitStruct.Pin = DAC2_Pin|DAC3_Pin|DAC5_Pin|DAC7_Pin|DAC8_Pin;
1559   GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
1560   GPIO_InitStruct.Pull = GPIO_NOPULL;
1561   GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_VERY_HIGH;
1562   HAL_GPIO_Init(GPIOE, &GPIO_InitStruct);
1563 }
1564
1565 /*******************************************************************************
     *******/
1566 /*                  @brief  This function is executed in case of error
     occurrence              */
1567 /*******************************************************************************
     *******/
1568 void Error_Handler(void)
1569 {
1570   /* USER CODE BEGIN Error_Handler_Debug */
1571   /* User can add his own implementation to report the HAL error return state */
1572
1573   /* USER CODE END Error_Handler_Debug */
1574 }
1575
1576 #ifdef  USE_FULL_ASSERT
1577 /**
1578   * @brief  Reports the name of the source file and the source line number
1579   *         where the assert_param error has occurred.
1580   * @param  file: pointer to the source file name
1581   * @param  line: assert_param error line source number
1582   * @retval None
1583   */
1584 void assert_failed(uint8_t *file, uint32_t line)
1585 {
1586   /* USER CODE BEGIN 6 */
1587   /* User can add his own implementation to report the file name and line number,
1588      tex: printf("Wrong parameters value: file %s on line %d\r\n", file, line) */
1589   /* USER CODE END 6 */
1590 }
1591 #endif /* USE_FULL_ASSERT */
1592
1593 /************************ (C) COPYRIGHT STMicroelectronics *****END OF FILE****/
1594
```

C.2 MAIN.h

main.h

```
1
2 /**
3   ******************************************************************************
4   * @file           : main.h
5   * @brief          : Header for main.c file.
6   *                   This file contains the common defines of the application.
7   ******************************************************************************
8   * @author         : Jonas Jegminat
9   * @date           : 12.05.2020
10  ******************************************************************************
11  */
12
13 /* Define to prevent recursive inclusion -------------------------------------*/
14 #ifndef __MAIN_H
15 #define __MAIN_H
16
17 #ifdef __cplusplus
18 extern "C" {
19 #endif
20
21 /* Includes ------------------------------------------------------------------*/
22 #include "stm32h7xx_hal.h"
23 #include <stdint.h>
24
25 void Error_Handler(void);
26
27 enum {
28     DAC_1,
29     DAC_2,
30     DAC_3,
31     DAC_4,
32     DAC_5,
33     DAC_6,
34     DAC_7,
35     DAC_8,
36 };
37
38 /* Private defines -----------------------------------------------------------*/
39 #define BButton_Pin GPIO_PIN_13
40 #define BButton_GPIO_Port GPIOC
41
42 #define LED1_Pin GPIO_PIN_0
43 #define LED1_GPIO_Port GPIOB
44 #define LED2_Pin GPIO_PIN_7
45 #define LED2_GPIO_Port GPIOB
46 #define LED3_Pin GPIO_PIN_14
47 #define LED3_GPIO_Port GPIOB
48
49 #define LDAC_Pin GPIO_PIN_12
50 #define LDAC_GPIO_Port GPIOF
51
52 #define DAC1_Pin GPIO_PIN_13
53 #define DAC1_GPIO_Port GPIOF
54 #define DAC2_Pin GPIO_PIN_9
55 #define DAC2_GPIO_Port GPIOE
56 #define DAC3_Pin GPIO_PIN_11
57 #define DAC3_GPIO_Port GPIOE
58 #define DAC4_Pin GPIO_PIN_14
59 #define DAC4_GPIO_Port GPIOF
60 #define DAC5_Pin GPIO_PIN_13
61 #define DAC5_GPIO_Port GPIOE
62 #define DAC6_Pin GPIO_PIN_15
```

main.h

```
63 #define DAC6_GPIO_Port GPIOF
64 #define DAC7_Pin GPIO_PIN_8
65 #define DAC7_GPIO_Port GPIOE
66 #define DAC8_Pin GPIO_PIN_7
67 #define DAC8_GPIO_Port GPIOE
68
69 #define USB_PowerSwitchOn_Pin GPIO_PIN_6
70 #define USB_PowerSwitchOn_GPIO_Port GPIOG
71 #define USB_OverCurrent_Pin GPIO_PIN_7
72 #define USB_OverCurrent_GPIO_Port GPIOG
73 #define USB_VBUS_Pin GPIO_PIN_9
74 #define USB_VBUS_GPIO_Port GPIOA
75 #define USB_DM_Pin GPIO_PIN_11
76 #define USB_DM_GPIO_Port GPIOA
77 #define USB_DP_Pin GPIO_PIN_12
78 #define USB_DP_GPIO_Port GPIOA
79
80 #define USART_TX_Pin GPIO_PIN_6
81 #define USART_TX_GPIO_Port GPIOB
82 #define USART_RX_Pin GPIO_PIN_7
83 #define USART_RX_GPIO_Port GPIOB
84
85 #define DAC_NoOfBits 16
86 #define DAC_CONTROL_BITS 28672
87 #define TIM1_CLK 64000000
88
89 #define GPIO_NUMBER           (16U)
90
91 #define SPI_SR_TXC_Pos              (12U)
92 #define SPI_SR_TXC_Msk              (0x1UL << SPI_SR_TXC_Pos)                  /*!<
   0x00001000 */
93 #define SPI_SR_TXC                  SPI_SR_TXC_Msk                             /*!<TxFIFO
   transmission complete */
94 #define SPI_FLAG_TXC                    SPI_SR_TXC     /* SPI status flag : TxFIFO
   transmission complete flag        */
95
96 #define TIM_CR1_CEN_Pos           (0U)
97 #define TIM_CR1_CEN_Msk           (0x1UL << TIM_CR1_CEN_Pos)                 /*!<
   0x00000001 */
98 #define TIM_CR1_CEN               TIM_CR1_CEN_Msk                            /*!<Counter
   enable */
99
100 #define TIM_SR_UIF_Pos            (0U)
101 #define TIM_SR_UIF_Msk            (0x1UL << TIM_SR_UIF_Pos)                  /*!<
   0x00000001 */
102 #define TIM_SR_UIF                TIM_SR_UIF_Msk                             /*!<Update
   interrupt Flag */
103
104 #define TIM_DIER_UIE_Pos          (0U)
105 #define TIM_DIER_UIE_Msk          (0x1UL << TIM_DIER_UIE_Pos)                /*!<
   0x00000001 */
106 #define TIM_DIER_UIE              TIM_DIER_UIE_Msk                           /*!<Update
   interrupt enable */
107 /* USER CODE END Private defines */
108
109 #ifdef __cplusplus
110 }
111 #endif
112
113 #endif /* __MAIN_H */
114
115 /************************ (C) COPYRIGHT STMicroelectronics *****END OF FILE****/
```

main.h

116

Erklärung zur selbstständigen Bearbeitung einer Abschlussarbeit

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich gemacht.

Ort	Datum	Unterschrift im Original