

Ostbayerische Technische Hochschule Regensburg

Fakultät Elektro- und Informationstechnik



## VMCB-Projektdokumentation

Funktionsgenerator Version 2

mit Bedienungsanleitung



**Projektmitglieder:** Manuel Dentgen

Tobias Frauenschläger

Thomas Taugenbeck

**Betreuer:** Prof. Dr.-Ing. Hans Meier

Prof. Dr. rer. nat. Roland Mandl

**Ausstellungsdatum:** 12. April 2019

# Inhaltsverzeichnis

<b>Inhaltsverzeichnis</b>	<b>2</b>
<b>Abbildungsverzeichnis</b>	<b>4</b>
<b>1 Einleitung</b>	<b>5</b>
<b>2 Konzept</b>	<b>6</b>
2.1 Systemanforderungen . . . . .	6
2.2 Hardware-Design . . . . .	7
2.3 Software-Architektur . . . . .	8
<b>3 Hardware</b>	<b>9</b>
3.1 Gehäuse und Anschlüsse . . . . .	9
3.2 Netzteil . . . . .	11
3.3 Frontpanel . . . . .	13
3.4 Hauptplatine . . . . .	14
3.4.1 Logikteil . . . . .	14
3.4.2 Analoge Signalerzeugung . . . . .	15
<b>4 Software</b>	<b>21</b>
4.1 Hardware-Abstraktion in Form eines Schichtenmodells . . . . .	21
4.2 Entwickelte Komponenten der Components-Schicht . . . . .	25
4.2.1 Benutzereingabe durch Taster am Frontpanel . . . . .	25
4.2.2 Benutzereingabe durch Dreh-Encoder am Frontpanel . . . . .	26
4.2.3 Visualisierung mittels Grafik-Display . . . . .	28
4.2.4 Speichern der Benutzereinstellungen . . . . .	30
4.2.5 Einstellen der Eingangsfrequenz des Signalgenerator ICs . . . . .	30
4.2.6 Erzeugen zusätzlich benötigter Spannungen . . . . .	31
4.2.7 Erzeugen des Ausgangssignals . . . . .	31
4.2.8 Bündelung der analogen Signalerzeugung . . . . .	32
4.3 Entwicklung des Betriebssystems innerhalb der Application-Schicht . . . . .	33
4.3.1 State Machine . . . . .	33
4.3.2 MenuBase . . . . .	34
4.3.3 BootscreenMenu . . . . .	35
4.3.4 MainMenu . . . . .	35
4.3.5 ChannelMenu . . . . .	36
4.3.6 Submenus . . . . .	38

4.3.7 MenuController . . . . .	40
<b>5 Fazit</b>	<b>42</b>
<b>Literaturverzeichnis</b>	<b>50</b>

## Abbildungsverzeichnis

1	Blockschaltbild des Gesamtsystems . . . . .	8
2	Frontansicht des Funktionsgenerators als 3D-Modell . . . . .	9
3	Rückansicht des Funktionsgenerators als 3D-Modell . . . . .	10
4	Draufsicht des Funktionsgenerators als 3D-Modell . . . . .	10
5	Netzteil der Firma MEAN WELL . . . . .	11
6	Netzteilplatine als 3D-Modell . . . . .	11
7	Blockschaltbild der Netzteilplatine . . . . .	12
8	Platine des Frontpanel als 3D-Modell . . . . .	13
9	Blockschaltbild der Hauptplatine . . . . .	16
10	Hauptplatine als 3D-Modell . . . . .	20
11	Struktur des Schichtenmodells . . . . .	22
12	Prinzip der Umwandlung zwischen Submenü-Parameter und Encoder	27
13	Encoder im <i>changePosition</i> Modus . . . . .	27
14	Encoder im <i>changeValue</i> Modus . . . . .	27
15	Zusammenhang der Nutzdatenbytes mit Pixelposition im Display . .	28
16	Aufbau der State Machine mit Übergangsbedingungen . . . . .	34
17	Allgemeine Übersicht des Hauptmenüs . . . . .	36
18	Allgemeine Übersicht des Menüs für Channel 1 . . . . .	38
19	Submenü mit dem Encoder im <i>changeDigit</i> -Modus . . . . .	39
20	Submenü mit dem Encoder im <i>changeValue</i> -Modus . . . . .	40
21	Veränderter Wert nach Drücken des <i>:10-Buttons</i> . . . . .	41
22	Der fertige Funktionsgenerator . . . . .	42
23	Rechtecksignal mit Frequenz $f = 1 \text{ Hz}$ . . . . .	43
24	Rechtecksignal mit Frequenz $f = 1 \text{ MHz}$ . . . . .	43
25	Rechtecksignal mit Frequenz $f = 10 \text{ MHz}$ . . . . .	44
26	Sinus-Signal mit Amplitude $\hat{U} = 1 \text{ V}$ . . . . .	45
27	Sinus-Signal mit Offset $\bar{U} = 1 \text{ V}$ . . . . .	46
28	45° Phasenversatz zwischen beiden Ausgangskanälen . . . . .	46
29	Rechtecksignal mit 10 % Duty-Cycle . . . . .	47
30	Signalunterbrechung bei Parameterumschaltung . . . . .	47

## 1 Einleitung

Im Rahmen des Fachs *Praktikum Messtechnik 2* (PMT2) innerhalb des Bachelor-Studienganges Elektro- und Informationstechnik an der Ostbayerischen Technischen Hochschule Regensburg wurde bereits ein Funktionsgenerator von den Projektteilnehmern entwickelt [1]. Dieser sollte zwei unabhängige Ausgänge besitzen und unterschiedliche Signalverläufe ausgeben können, wobei die Frequenz von 1 Hz bis 1 MHz einstellbar sein und der Ausgangsspannungsbereich bis zu  $\pm 10\text{ V}$  betragen sollte.

Während der Entwicklung dieses Geräts sind jedoch einige Probleme in Hard- und Software aufgetreten. Vor allem die Schaltung für die analoge Signalerzeugung wies deutliche Defizite auf. Da die Dokumentation des verwendeten Bausteins zur Signalgenerierung unvollständig und teilweise falsch war, konnten die für das Projekt angesetzten Ziele nicht vollständig erfüllt werden. Auch die Beschaltung des Bausteins und die nachfolgenden Stufen zur analogen Signalverarbeitung wiesen einige Fehler auf, weshalb zum Beispiel keine große Frequenzdifferenz zwischen den beiden Kanälen eingestellt werden konnte. Für die Interaktion mit dem Benutzer und die Einstellung der Parameter beider Kanäle wurde ein 4x20 Zeichen großes Display verwendet. Dieses hat sich während der Entwicklung des User Interface als zu klein und wenig flexibel für die Anforderungen der Darstellung und Menüführung herausgestellt. Aus Sicht der Software war das größte Problem, dass ohne ein vorher ausgearbeitetes Konzept programmiert wurde. Daher hatte der entstandene Code wenig Struktur und wurde schnell unübersichtlich.

Aus diesen Gründen war das Endergebnis des Projekts insgesamt nicht zufriedenstellend und es wurde sich dazu entschieden, eine neue Version des Funktionsgenerators von Grund auf neu zu entwickeln. Diese sollte die gleichen Anforderungen und Spezifikationen wie die erste Version aufweisen. Aufgrund des Umfangs und des hohen zeitlichen Aufwands wurde der Entwicklungsprozess von Version 2 auf die beiden Fächer *Vertiefung Mikrocontrollertechnik* (VMCB) und *Vertiefung Mess- und Sensortechnik* (VMS) aufgeteilt. Dieses Dokument bildet eine ausführliche Dokumentation und Projektbeschreibung für den *Funktionsgenerator V2*. Dabei wird auf die Entwicklung der Hard- und Software eingegangen und herausgestellt, wie die Probleme aus der ersten Version mit den vorgestellten Ansätzen gelöst wurden.

Am Ende dieses Dokuments befindet sich zudem eine Bedienungsanleitung für den Funktionsgenerator, welche die gesamte Benutzerinteraktion und die Einstellung der unterschiedlichen Parameter der beiden Ausgangssignale beschreibt.

## 2 Konzept

Zu Beginn der Neuentwicklung von Version 2 des Funktionsgenerators wurde ein umfangreiches Konzept entworfen. Der erste Schritt bestand darin, die Anforderungen an das System festzulegen und sämtliche Funktionen des Signalgenerators zu definieren. Dieser Prozess wird in Kapitel 2.1 beschrieben. Darauf aufbauend wurde die zu entwickelnde Hardware spezifiziert, was in Kapitel 2.2 erläutert wird. Im Anschluss wird in Kapitel 2.3 die Architektur der dazu passenden Software in Form einer Mikrocontroller-Firmware besprochen.

### 2.1 Systemanforderungen

Die Anforderungen und Spezifikationen des Funktionsgenerators sind grundsätzlich zweigeteilt. Zum einen wurde die analoge Signalerzeugung in ihren Parametern spezifiziert, zum anderen ist die Menüführung und Benutzerinteraktion definiert und festgelegt worden. Aus Sicht der Signalerzeugung wurden die Anforderungen im Vergleich zu Version 1 nicht verändert. Ziel war ein System mit zwei getrennten Ausgangskanälen, die unabhängig voneinander bedient und konfiguriert werden können. Jeder Kanal sollte hierbei die in Tabelle 1 genannten Spezifikationen aufweisen.

Tabelle 1: Spezifikationen der analogen Signalerzeugung für einen Ausgangskanal

Parameter	Min	Typisch	Max	Auflösung	Einheit
Signalform		Sinus, Rechteck (einstellbarer Duty-Cycle), Dreieck, Sägezahn (anstiegend und abfallend)			
Ausgangsfrequenz	1		$10^6$	1	Hz
Ausgangsspannung	-10		+10	$10^{-3}$	V
Signal-Offset	-10		+10	$10^{-3}$	V
Signal-Phase	-180		+180	1	°
Ausgangswiderstand		50			$\Omega$

Hinsichtlich der Ausgangsfrequenz wurde der Wert 1 MHz als Mindestziel festgelegt. Höhere Frequenzen sollten mit der fertigen Hardware bezüglich Signal-Integrität evaluiert und getestet werden. Die Parameter Ausgangsspannung und Signal-Offset sind jeweils bezogen auf ein Ausgangssignal ohne  $50\ \Omega$ -Abschlusswiderstand. Inklusive Abschluss halbiert sich die Amplitude des Signals entsprechend dem internen Ausgangswiderstand von  $50\ \Omega$ . Durch eine Synchronisierung der beiden Ausgangskanäle kann mit dem Parameter Signal-Phase vor allem der Phasenversatz zwischen den beiden Kanälen beeinflusst werden.

Die Menüführung und Benutzerinteraktion wurde grundsätzlich von Version 1 übernommen. Auf der Vorderseite des Gerätes sind für Benutzereingaben acht Taster und ein Drehencoder vorhanden. Visuelles Feedback erfolgt über eine dreifarbiges Hintergrundbeleuchtung der acht Taster (rot, grün, gelb) und ein Display. Bei der Wahl des Displays wurde für Version 2 ein deutlich größeres und frei programmierbares Modell gewählt, um die Anzeige kreativer gestalten zu können. Die Menüführung basiert im wesentlichen auf drei Menüs: Einem Hauptmenü und jeweils einem Menü pro Ausgangskanal. Innerhalb des Hauptmenüs können Einstellungen bezüglich des Displays und des Gesamtsystems getroffen und Informationen wie die Firmware-Version abgefragt werden. Im Menü eines Ausgangskanals werden entsprechend die Parameter des Signals verändert. Während des System-Starts findet zudem ein Test aller visuellen Funktionen statt, damit der Benutzer ein Problem der Hardware schnell identifizieren kann. Damit waren alle Spezifikationen und Funktionen von Version 2 des Funktionsgenerators festgelegt. Darauf aufbauend wurde die benötigte Hardware und Software ausgearbeitet, was in den folgenden beiden Kapiteln genauer beschrieben wird.

## 2.2 Hardware-Design

Um die Anforderungen an das System aus Kapitel 2.1 umsetzen zu können, war ein umfangreicher Hardware-Entwurf notwendig. Aufgrund der Platz-Bedingungen innerhalb des Gehäuses, welches von Version 1 übernommen wurde, war eine Aufteilung der Hardware notwendig (siehe Abbildung 4 auf Seite 10). Sämtliche Komponenten für die Interaktion mit dem Anwender wurden auf einer gesonderten Platine verbaut, welche direkt hinter der Abdeckung auf der Vorderseite des Gehäuses montiert ist. Dieses Frontpanel ist über eine Kabelverbindung mit der Hauptplatine des Systems verbunden, welche im unteren Bereich des Geräts befestigt ist. Hier sind sowohl der Logikteil des Funktionsgenerators als auch die analoge Signalerzeugung für die beiden Ausgangskanäle verbaut. Für die Generierung der Signale werden zusätzlich zur normalen Versorgungsspannung der Bauteile von +3,3 V zwei weitere bipolare Spannungsquellen benötigt ( $\pm 5$  V und  $\pm 15$  V). Diese fünf Spannungen werden mithilfe eines zugekauften Netzteils und einer eigens entwickelten dritten Platine direkt im Gerät erzeugt und über eine weitere Kabelverbindung auf der Hauptplatine bereitgestellt. In Abbildung 1 ist ein Blockschaltbild dieses Hardware-Entwurfs zu sehen, wobei die einzelnen Platinen nur grob gezeigt werden. In Kapitel 3 werden die drei Platinen jeweils noch detaillierter betrachtet.

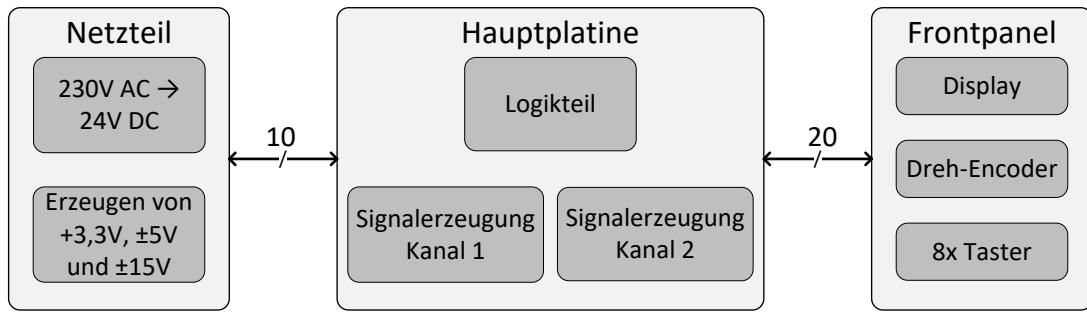


Abb. 1: Blockschaltbild des Gesamtsystems

## 2.3 Software-Architektur

Passend zur entwickelten Hardware war eine Firmware für den auf der Hauptplatine verbauten Mikrocontroller zu entwerfen, welche die Steuerung des gesamten Systems übernimmt. Grundlage für die Entwicklung der Software war ein *bare-metal* Ansatz. Das bedeutet, dass das Programm direkt auf die zugrundeliegende Hardware über deren Register zugreift, ohne ein Betriebssystem als Zwischen-Instanz. Aus einer früheren Arbeit [2] war bereits ein bestehendes Code-Gerüst vorhanden, welches auf einem solchen Ansatz aufbaut und für das vorliegende System mit einigen Modifikationen und Erweiterungen wiederverwendet werden konnte. Dieser Code basiert auf einem Schichtenmodell, in dem sämtliche Programmabläufe strukturiert und für höhere Schichten abstrahiert werden. In Kapitel 4.1 wird das Modell und die einzelnen Schichten detailliert beschrieben. Nach Anpassung des Codes an die vorliegende Hardware war eine Grundlage geschaffen, auf der die eigentlichen Funktionalitäten des Signalgenerators implementiert werden konnten. Dafür wurde für jede logische Funktionsgruppe eine eigene Software-Komponente erstellt, die unabhängig eine Aufgabe des Systems übernimmt. Diese einzelnen Komponenten werden in Kapitel 4.2 genauer betrachtet. Darauf aufbauend wurde eine Zustandsmaschine entwickelt, die unter Verwendung der Komponenten den eigentlichen Ablauf der Firmware bildet. Dies wird in Kapitel 4.3 besprochen. Insgesamt ist die entwickelte Software im Stande, das gesamte System zu steuern und alle Software-seitigen Anforderungen aus Kapitel 2.1 zu erfüllen.

## 3 Hardware

Wie bereits in Kapitel 2.2 beschrieben und in Abbildung 1 visualisiert, war für die Realisierung des Signalgenerators ein System aus drei Platinen erforderlich. Neben der Hauptplatine, welche den Logikteil und die analoge Signalerzeugung enthält, sind noch zwei Platinen für die Erzeugung der benötigten Versorgungsspannungen („Netzteil“) und für die Interaktion mit dem Anwender („Frontpanel“) vorhanden. Diese drei Platinen werden getrennt voneinander in den Kapiteln 3.2 bis 3.4 beschrieben, wobei die Hauptplatine auf zwei Kapitel aufgeteilt ist. Vereint werden alle Komponenten innerhalb eines Gehäuses, welches nach außen alle benötigten Schnittstellen für die Bedienung bereitstellt. Dieser Aufbau wird zu Beginn in Kapitel 3.1 genauer betrachtet.

### 3.1 Gehäuse und Anschlüsse

Als Gehäuse des Funktionsgenerators wird das Polystrolgehäuse AUS33 der Firma Teko [3] gewählt. Die Vorderseite des Gehäuses besteht aus einer Plexiglasscheibe, durch welche die direkt dahinter montierte Frontpanel-Platine erkennbar ist. Nach außen hin sind die acht Taster und der Drehencoder bereitgestellt, die für die Interaktion mit dem Anwender vorgesehen sind. Zudem sind noch das Display und die beiden BNC-Buchsen für die Ausgangssignale von der Vorderseite aus erreichbar. In Abbildung 2 ist die gesamte Vorderseite des Funktionsgenerators mit allen Komponenten zu sehen.



Abb. 2: Frontansicht des Funktionsgenerators als 3D-Modell

Auf der Rückseite des Gehäuses befindet sich ein Kaltgeräteanschluss, über wel-

chen die 230 V Versorgungsspannung für den Funktionsgenerator bereitgestellt wird. Darüber ist ein Schalter vorhanden, mit dem das Gerät ein- und ausgeschaltet werden kann (siehe Abbildung 3).

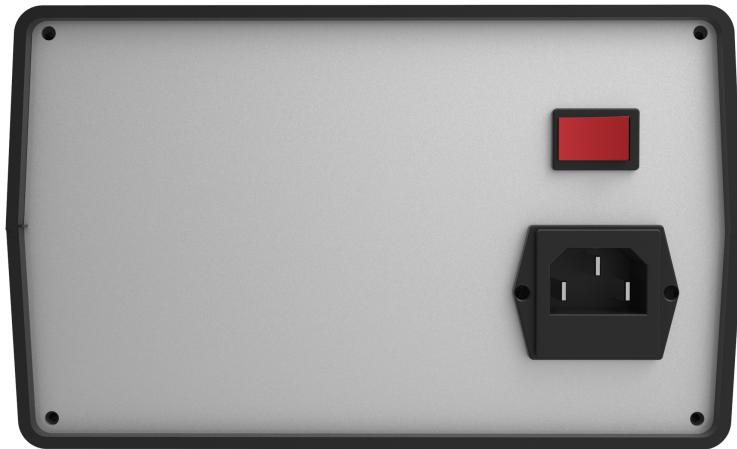


Abb. 3: Rückansicht des Funktionsgenerators als 3D-Modell

Innerhalb des Gehäuses sind die Platinen jeweils an ihrer entsprechenden Position verschraubt. Zudem wird das zugekauftes Netzteil (siehe Kapitel 3.2) im Gerät montiert. Insgesamt ergibt sich der in Abbildung 4 gezeigte Aufbau.

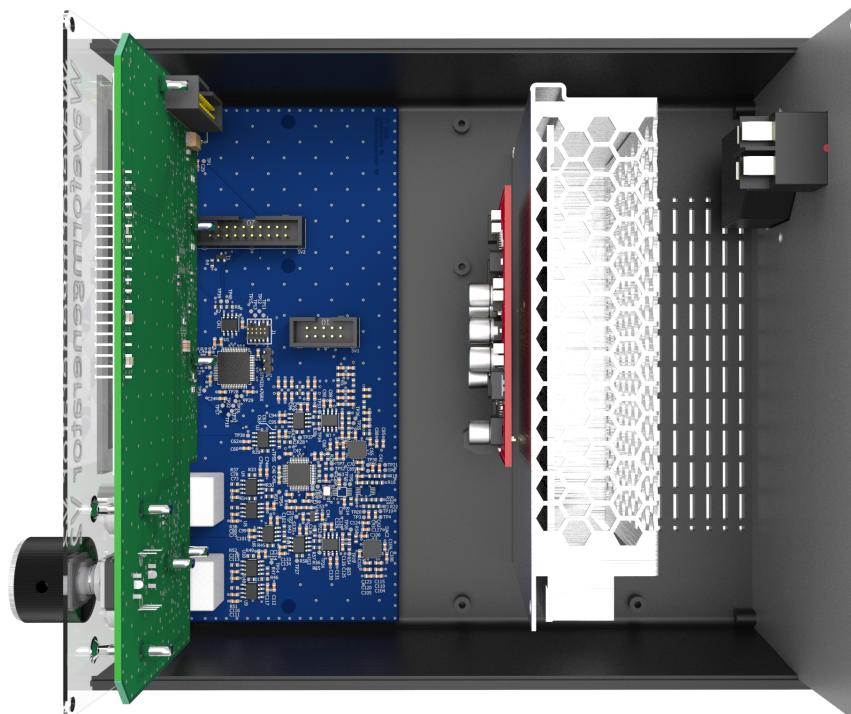


Abb. 4: Draufsicht des Funktionsgenerators als 3D-Modell

### 3.2 Netzteil

Für die analoge Signalerzeugung werden neben der Versorgungsspannung der Bauteile von +3,3 V noch die Spannungen  $\pm 15$  V und  $\pm 5$  V benötigt (siehe Kapitel 3.4). Damit diese Spannungen nicht alle von außen bereitgestellt werden müssen, wurde für den Funktionsgenerator eine eigene Spannungsversorgung entwickelt. Diese setzt sich aus zwei separaten Schaltungsteilen zusammen. Zum Einen kommt ein zugekauftes 100 W Netzteil der Firma MEAN WELL [4] zum Einsatz, welches aus der 230 V Netzspannung eine 24 V Gleichspannung erzeugt. Diese Komponente ist in Abbildung 5 zu sehen.



Abb. 5: 100W Netzteil der Firma MEAN WELL [4]

Zum Anderen wurde eine Platine entwickelt, welche aus der 24 V Gleichspannung die benötigten Spannungen +15 V, +5 V, -5 V, -15 V und +3,3 V erzeugt. Diese Platine ist in Abbildung 6 als 3D-Modell visualisiert.

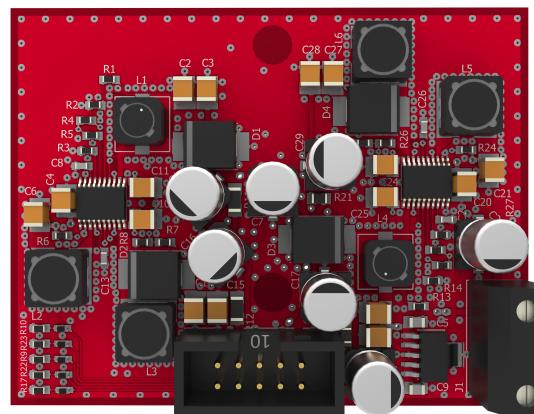


Abb. 6: Netzteilplatine als 3D-Modell

Der Anschluss unten rechts bildet die Klemme für die 24 V Eingangsspannung. Die beiden bipolaren Spannungsquellen wurden mit Hilfe zweier DC/DC Konverter des Typs LT8471 von der Firma Analog Devices [5] realisiert. Diese erzeugen durch interne Buck Konverter mit entsprechender Ausgangsbeschaltung aus den 24 V Eingangsspannung die separaten Ausgangsspannungen, wie im Blockschaltbild in Abbildung 7 visualisiert. Die +3,3 V Versorgungsspannung wird durch den Linear-Spannungsregler TPS7A4533 der Firma Texas Instruments [6] aus der +5 V Spannung erzeugt. Alle Spannungen sind jeweils über eine 1 A PTC Sicherung abgesichert, was eine Zerstörung der Bausteine durch Überlast am Ausgang verhindert. Für die visuelle Überprüfung der Ausgänge wurde zusätzlich je Spannung eine LED angebracht, über welche angezeigt wird ob diese jeweils durch das Netzteil bereitgestellt werden. Über den 10-poligen Stecker, welcher mittig im unteren Teil der Grafik zu sehen ist, werden die separaten Spannungen mit jeweils einer Masseverbindung für die Hauptplatine bereitgestellt. Mithilfe dieses Gesamtsystems kann der Funktionsgenerator mit nur einem Kabel an jeder normalen Steckdose betrieben werden, ohne dass sich der Anwender um eine komplexe Spannungsversorgung kümmern muss. Dies ermöglicht einen flexiblen und umfangreichen Einsatz des Geräts.

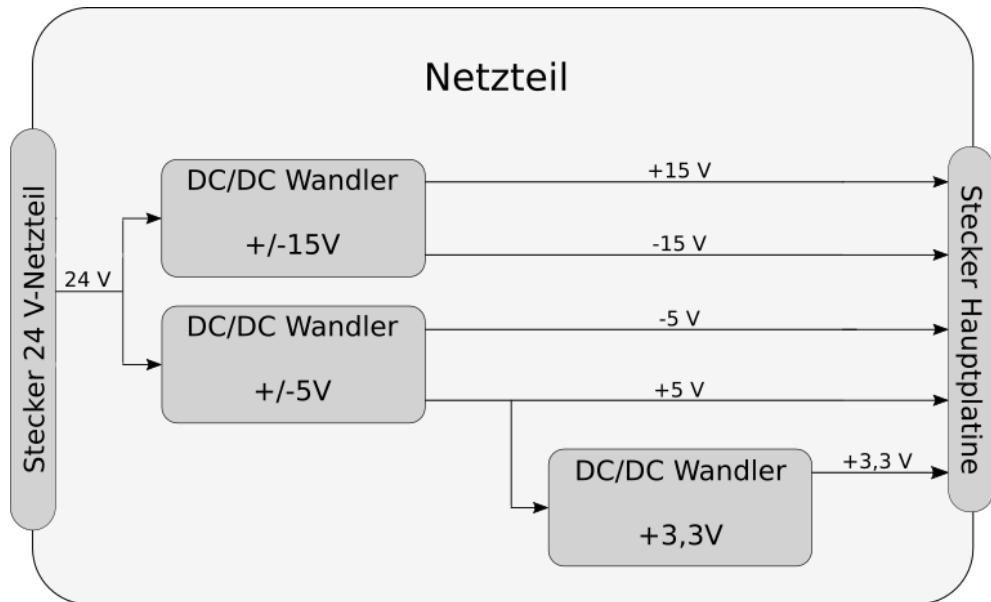


Abb. 7: Blockschaltbild der Netzteilplatine

### 3.3 Frontpanel

Die Frontpanel-Platine beinhaltet alle Komponenten für die Interaktion mit dem Anwender. Neben den acht Tastern und dem Drehencoder für Eingaben des Benutzers stellt vor allem das Display eine Kernkomponente dar. Hierbei handelt es sich um ein Punktmatrixdisplay mit einer Auflösung von 240 x 128 Pixeln, welches zur Visualisierung der Benutzeroberfläche verwendet wird. Die Taster sind in zwei Gruppen eingeteilt. Vier davon sind unterhalb des Display platziert und besitzen eine variable Funktion, welche jeweils durch das Display angezeigt wird. Daher wird für diese im Folgenden der Name *Display Buttons* verwendet. Die restlichen vier Taster sind rechts vom Display unter dem Drehencoder angebracht und besitzen feste Funktionen. Jeweils zwei Taster sind direkt oberhalb der BNC-Buchse eines Ausgangskanals platziert und sind somit diesem Kanal zugewiesen. Der obere Taster wählt dabei das entsprechende Menü in der Benutzeroberfläche aus und ermöglicht somit das Einstellen der Parameter des Ausgangssignals. Mit dem unteren Taster wird das eingestellte Signal aktiviert und auf die Buchse geschaltet. Die beiden Taster-Paare erhalten zur einfacheren Beschreibung im Folgenden die Namen *Channel-Select Buttons* und *Channel-Activate Buttons*. Mithilfe des Drehencoders können die Parameter der Kanäle innerhalb des entsprechenden Menüs verändert werden. Eine ausführliche Anleitung zur Bedienung des Systems und den jeweiligen Funktionen der Taster sowie des Drehencoders befindet sich in der Bedienungsanleitung am Ende dieses Dokuments. In Abbildung 8 ist zudem die gesamte Frontpanel-Platine als 3D-Modell zu sehen.

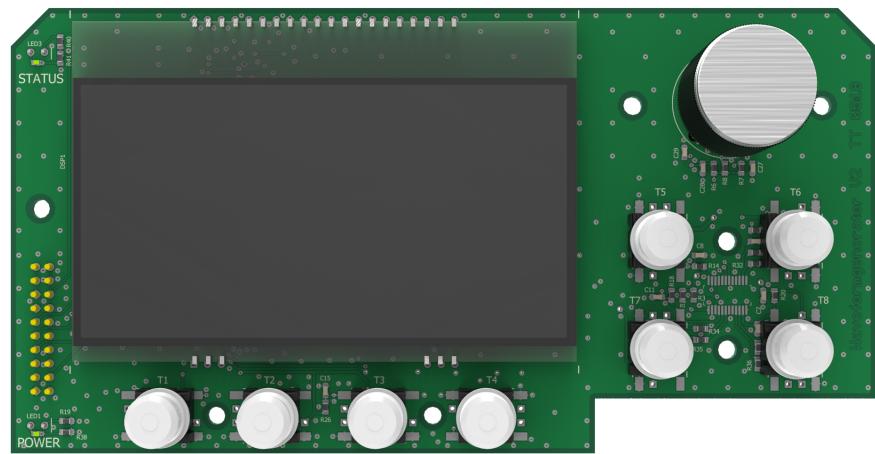


Abb. 8: Platine des Frontpanel als 3D-Modell

Im linken Teil der Platine (siehe Abbildung 8) befinden sich zudem noch zwei LEDs.

Die untere zeigt an, ob eine Spannungsversorgung vorhanden ist und leuchtet daher dauerhaft. Die obere LED zeigt einen *Heartbeat* des Mikrocontrollers und deutet durch Blinken im Sekudentakt eine aktive Firmware an. Für das visuelle Feedback befindet sich in allen Tastern zudem eine rote und eine grüne LED, wodurch die Farben rot, grün oder gelb als Beleuchtung eingestellt werden kann. Dies wird für die Visualisierung des aktuell ausgewählten Menüs verwendet. Intern werden jeweils vier Taster und deren LEDs über einen Port-Expander PCA9555 der Firma NXP [7] gesteuert, welcher wiederum über einen I<sup>2</sup>C-Bus vom Mikrocontroller angesprochen wird. Das Display bekommt seine Daten über eine SPI-Schnittstelle und der Drehencoder ist direkt auf ein passendes Interface des Controllers gelegt, wobei sämtliche Signale und Schnittstellen über eine Kabel-Verbindung von der Hauptplatine bereitgestellt werden. Somit sind alle für die Interaktion mit dem Anwender benötigten Komponenten auf der Frontpanel-Platine konzentriert und vom Mikrocontroller auf der Logikplatine ansprechbar.

## 3.4 Hauptplatine

Das Herzstück des Funktionsgenerators wird durch die Hauptplatine gebildet. Diese beinhaltet den gesamten Logikteil zum Steuern des Geräts sowie die analoge Signalerzeugung für die beiden Ausgangskanäle.

### 3.4.1 Logikteil

Der Logikteil besteht im wesentlichen aus einem Mikrocontroller von STMicroelectronics des Typs *STM32L476* [8]. Dieser basiert auf der Cortex M4 Architektur von ARM und bietet mit 80 MHz Systemtakt und 128 kB RAM genug Leistung für die anfallenden Aufgaben. Diese umfassen im wesentlichen die Ansteuerung sämtlicher externer Komponenten. Dabei handelt es sich zum einen um die Baugruppen des Frontpanels (Display, Port-Expander und Drehencoder; siehe Kapitel 3.3), zum anderen um die Elemente der analogen Signalerzeugung. Dafür kommt eine Vielzahl an interner Peripherie des Mikrocontrollers zum Einsatz, worauf jedoch in Kapitel 4.2 im Zusammenhang mit der dazu gehörigen Software noch genauer eingegangen wird. Neben dem Controller an sich beinhaltet der Logikteil noch einen Real-Time Quarz, welcher als zusätzliche Taktquelle dient. Um die letzten Einstellungen des Anwenders über einen Neustart des Systems hinaus speichern zu können, ist zudem ein EEPROM verbaut. In diesem Speicherbaustein werden sämtliche Parameter der analogen Ausgangssignale und Daten wie die aktuelle Hintergrundbeleuchtung oder der Kontrast des Displays gespeichert. Angesteuert wird diese Komponente über den

bestehenden I<sup>2</sup>C-Bus, welcher bereits für die Port-Expander des Frontpanels verwendet wird. Auf der Hauptplatine an sich sind des weiteren noch mehrere Stecker vorhanden, die zum einen die Verbindung zu den anderen beiden Platinen herstellen und zum anderen für das Programmieren des Mikrocontrollers verwendet werden. Des weiteren sind noch ein Reset-Taster und die beiden BNC-Buchsen verbaut, wobei hier die Platzierung so gewählt wurde, dass ein Zugriff auf die drei Elemente über die Vorderseite des Geräts ermöglicht wird. Die restlichen Komponenten der Hauptplatine sind hingegen Teil der analogen Signalerzeugung, welche getrennt im nächsten Kapitel betrachtet wird.

### 3.4.2 Analoge Signalerzeugung

Um ein Signal auf den BNC-Buchsen ausgeben zu können, welches den Einstellungen des Anwenders entspricht, wurde eine Schaltung zur Generierung von analogen Spannungssignalen auf der Hauptplatine entwickelt. Damit die beiden Ausgangskanäle des Funktionsgenerators unabhängig voneinander verwendet werden können, wurde die Schaltung im Gegensatz zu Version 1 für einen einzelnen Kanal ausgelegt und in zweifacher Ausführung auf der Platine verbaut. Lediglich zwei Baugruppen werden von beiden Kanälen gemeinsam verwendet, da hier eine unabhängige Ansteuerung zweier Ausgänge möglich ist. Darauf wird jedoch im weiteren Verlauf des Kapitels noch genauer eingegangen. Abbildung 9 zeigt das Blockschaltbild der Hauptplatine, wobei die Signalerzeugung nur für einen Kanal visualisiert ist, um die Grafik übersichtlich zu halten. Neben dem Logikteil (siehe Kapitel 3.4.1) sind die einzelnen Komponenten der Schaltung zur Generierung der Ausgangssignale zu erkennen. Diese werden im Folgenden schrittweise erläutert.

#### Generierung der Signale

Im wesentlichen basiert die Erzeugung der Ausgangssignale auf einem speziell für diese Anwendung entwickelten IC, der intern mittels eines Digital-Analog-Wandlers und einem sogenannten *Direct Digital Synthesizer* (DDS) [9] analoge Signalverläufe generiert. Im vorliegenden Fall wurde sich für den IC AD9102 von Analog Devices [10] entschieden. Dieser wird durch Konfiguration einiger interner Register über ein SPI-Interface angesteuert und sollte laut Datenblatt alle festgelegten Signalformen als differenzielles Stromsignal erzeugen können. Dieses Signal wird mithilfe zweier Impedanzwandler und eines anschließenden Differenzverstärkers in ein Masse-bezogenes Spannungssignal mit einer maximalen Amplitude von  $\pm 2V$  umgewandelt. Durch diese Limitierung der Amplitude wird der Leistungsfluss auf der Platine minimiert und die verwendeten Operations-

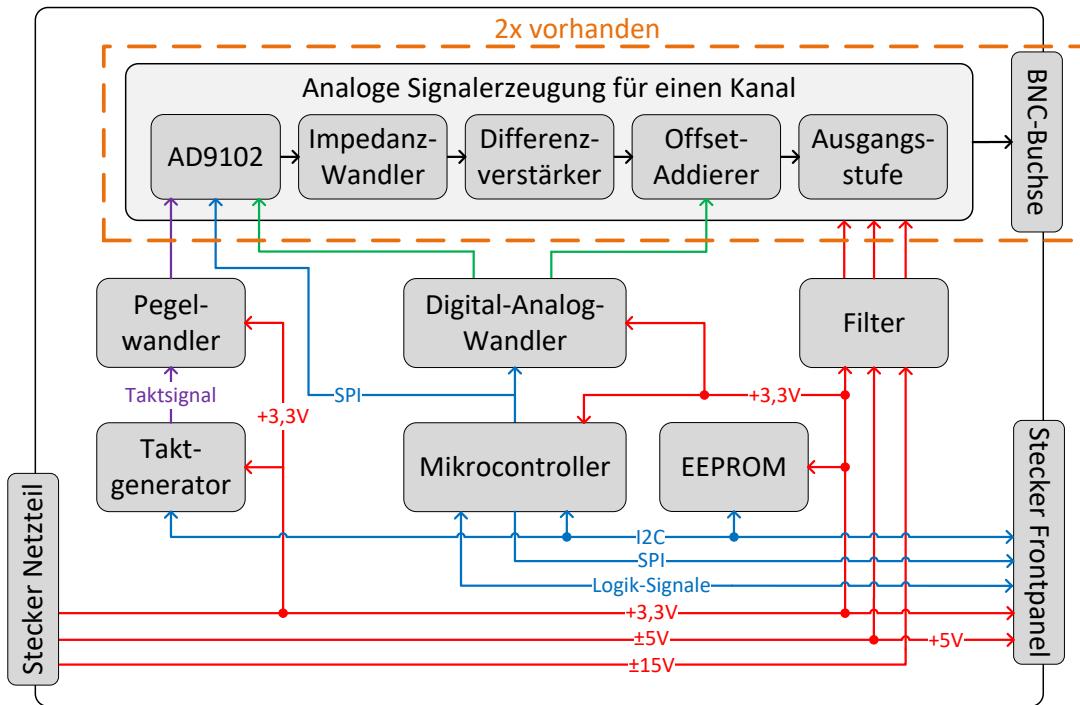


Abb. 9: Blockschaltbild der Hauptplatine

verstärker sind bei einer Versorgungsspannung von  $\pm 5V$  noch weit von ihren Aussteuer-Grenzen entfernt. Im weiteren Verlauf der Schaltung findet dann eine Verstärkung des Signals auf den eigentlichen Ausgangsspannungsbereich des Funktionsgenerators statt. Die Einstellung der Amplitude erfolgt über die Variation einer externen Spannungsreferenz für den AD9102, worauf jedoch später noch eingegangen wird. Die Frequenz des Signals kann mithilfe eines internen 24-bit Teilers des AD9102 festgelegt werden, welcher dazu ein extern bereitgestelltes Signal mit definierter Eingangsfrequenz benötigt. Der Ablauf wird ebenfalls im weiteren Verlauf noch genauer beschrieben. Auch die Phase der beiden Ausgangssignale zueinander wird innerhalb des AD9102 konfiguriert. Am Ausgang des Differenzverstärkers liegt somit ein masse-bezogenes, symmetrisches Spannungssignal mit passender Frequenz, Amplitude und Phase vor. Dieses Signal durchläuft in einem nächsten Schritt eine Additionsstufe, welche den eingestellten Offset (angepasst an die Amplitude von  $\pm 2V$ ) aufaddiert. Zuletzt folgt eine Ausgangsstufe, welche das Signal um den Faktor 5 verstärkt, damit der in Kapitel 2.1 definierte maximale Ausgangsspannungsbereich von  $\pm 10V$  erreicht wird. Diese Stufe hat einen angepassten Ausgangswiderstand von  $50\Omega$  und ist direkt mit der entsprechenden BNC-Buchse verbunden.

## Erzeugung zusätzlich benötigter Spannungen

Für die Einstellung der Amplitude wird, wie bereits erwähnt, eine einstellbare externe Spannungsreferenz für den AD9102 benötigt. Zudem muss eine den Benutzer-Einstellungen entsprechende Offset-Spannung für die Additionsstufe erzeugt werden. Diese beiden Signale pro Kanal werden mithilfe eines Digital-Analog-Wandlers mit vier unabhängigen Ausgängen erzeugt. Dabei handelt es sich um ein Modell von Texas Instruments mit dem Namen *DAC8734* [11]. Durch diesen werden für beide Kanäle sowohl die Referenzspannung für den AD9102 als auch die Offset-Spannung für die Additionsstufe erzeugt. Die Ansteuerung des Bausteins erfolgt ebenfalls über ein SPI-Interface, wobei für die insgesamt drei SPI-Slaves auf der Hauptplatine ein Bus mit getrennten Chip-Select-Leitungen verwendet wird.

## Zusatzkomponente für die Frequenzeinstellung

Der AD9102 benötigt für die Generierung eines Signals ein externes Taktsignal, welches als Referenz für den internen Teiler dient. Die Frequenz-Einstellung mittels dieser externen Eingangsfrequenz und dem 24-bit Teiler musste aufgrund der Anforderungen aus Kapitel 2.1 um eine Komponente erweitert werden. Die maximale Eingangsfrequenz des AD9102 beträgt 180MHz. Um bei Ausgangssignalen mit Frequenzen im MHz-Bereich eine akzeptable Signalform mit möglichst vielen Samples pro Periode zu erhalten, muss dieses obere Limit maximal ausgenutzt werden. Dadurch entsteht jedoch ein Problem bei sehr kleinen Ausgangsfrequenzen. Mit 180MHz Eingangsfrequenz beträgt die Auflösung des 24-bit Teilers 10,73Hz. Somit lässt sich die System-Anforderung von minimal 1Hz Ausgangsfrequenz nicht realisieren. Um diese Problematik zu lösen, kommt ein zusätzliches Bauteil auf der Platine zum Einsatz. Mithilfe des Clock Generators *Si5351* von Silicon Labs [12] kann die Eingangsfrequenz der beiden AD9102 unabhängig voneinander variabel in einem Bereich von 2kHz bis 180MHz eingestellt werden. Dieser Baustein benötigt ein 25MHz Taktsignal als Eingangsfrequenz und erzeugt mithilfe zweier PLLs (Phase-Locked-Loop, siehe [13]) und unterschiedlichen Multiplizierern und Dividierern die beiden konfigurierten Frequenzen für die Takteingänge des AD9102. Die Ansteuerung des Bausteins erfolgt über I<sup>2</sup>C, wobei der bestehende Bus für die Port-Expander des Frontpanels verwendet wird. Mit dieser zusätzlichen Komponente können die Eingangsfrequenzen der AD9102 Bausteine unabhängig und entsprechend der jeweiligen Ausgangsfrequenz angepasst werden, was eine maximale Auflösung über den gesamten Frequenz-Bereich für beide Kanäle ermöglicht.

## Weitere Komponenten

Neben diesen Kernkomponenten der Signalerzeugung sind noch weitere Bausteine erforderlich, um die angesetzten Ziele zu erreichen. Für den Digital-Analog-Wandler ist zur Generierung der Spannungen eine externe 2,5V Referenz auf der Platine verbaut. Um die Taktsignale des Si5351 Clock Generators mit den beiden AD9102 Bausteinen verwenden zu können, war zudem eine Pegelwandlung von CMOS-Level (0V und 3,3V) auf LVPECL (1,3V und 2,0V) erforderlich. Um weiterhin Störeinflüsse der Versorgungsspannungen nicht auf das analoge Ausgangssignal zu koppeln, wurde jeweils noch eine Filterschaltung aus mehreren Kondensatoren und einem Ferrit zwischen Logik- und Analogteil platziert.

## Verlauf der Entwicklung und aufgetretene Probleme

Zur Entwicklung der finalen Schaltung für die Signalgenerierung wurde als Zwischenschritt eine Testplatine entwickelt, wodurch eine Evaluierung der Komponenten und der Schaltung durchgeführt werden konnte. Mithilfe dieser konnte zudem schon frühzeitig mit der Entwicklung der Software begonnen werden. Während der Evaluierung durch die Testplatine sind einige Probleme aufgefallen, welche den endgültigen Schaltplan maßgeblich verbessert haben. In der Auswahlphase der Komponenten für die Signalerzeugung fiel die Entscheidung auf den IC AD9102, da dieser laut Datenblatt alle benötigten Signalformen außer einem Rechtecksignal intern durch Konfiguration verschiedener Register generieren kann. Somit wäre eine simple Software-seitige Ansteuerung möglich gewesen, wobei das Rechtecksignal Hardware-seitig mit einer optional zuschaltbaren Schaltung erzeugt worden wäre. Während der Software-Entwicklung mithilfe der Testplatine haben sich jedoch einige, teils schwerwiegende Probleme mit diesem Chip herausgestellt. Neben mehrerer Fehler in verschiedenen Beschreibungen des Datenblatts, welche die Software-Entwicklung erschwert und verzögert haben, war die erste große Hürde die Generierung der Sägezahn- und Dreieck-Signale. Es hat sich herausgestellt, dass diese einen deutlich geringeren möglichen Frequenzbereich (maximal wenige kHz) im Vergleich zum Sinus-Signal aufweisen, was dem Datenblatt nicht zu entnehmen ist und den in Kapitel 2.1 beschriebenen Anforderungen nicht ansatzweise gerecht wird. Als Lösungsansatz wurde die Verwendung einer weiteren Funktion des AD9102 herangezogen. Dieser bietet neben seinen Registern noch einen internen SRAM, in dem digitale Signale in Form von 12-bit Sample-Werten gespeichert und zur Generierung des Ausgangssignals verwendet werden können. Zudem sollte der Zeitpunkt der Umschaltung vom aktuellen Sample zum nächsten flexibel auf Basis der externen Taktquelle und dem internen Frequenzteiler festlegbar sein.

Somit wäre es möglich, die drei Signalformen einmalig während der Startphase des Funktionsgenerators in diesem SRAM zu speichern und zur Laufzeit durch Konfiguration der Register am Ausgang mit entsprechender Frequenz auszugeben. Allerdings hat sich hierbei das zweite schwerwiegende Problem mit dem IC herausgestellt, was nach Kontakt mit Analog Devices als Fehler im Hardware-Design identifiziert wurde. Die flexible Einstellung des Sample-Umschaltens auf Basis des internen Frequenzteilers ist so nicht möglich, weshalb der geplante Lösungsansatz in dieser Weise nicht umsetzbar war. Daher wurde ein zweiter Lösungsansatz entworfen, welcher jedoch umfangreiche Änderungen in Hardware und Software zur Folge hatte. Ohne die flexible Einstellung erfolgt die Umschaltung des aktuellen Sample-Werts aus dem SRAM mit dem Taktsignal, welches extern für den AD9102 bereitgestellt wird. Um die Frequenz der Signale aus dem SRAM verändern zu können, muss also die Eingangsfrequenz entsprechend angepasst werden, wobei die Bedingung in Gleichung 1 gelten muss.

$$f_{\text{in}} = \text{numOfSamples} \cdot f_{\text{out}} \quad (1)$$

Das Verhältnis aus Eingangsfrequenz  $f_{\text{in}}$  des AD9102 und der Signalfrequenz am Ausgang  $f_{\text{out}}$  muss genau der Anzahl der Samples **numOfSamples** im SRAM entsprechen. Die Anzahl der Sample-Werte wird hierbei mithilfe eines Algorithmus bestimmt (siehe Kapitel 4.2.5), der neben der Ausgangsfrequenz auch die gewünschte Phase des Signals beachtet, da diese in direktem Zusammenhang mit den Werten steht und so der resultierende Phasenfehler minimiert werden kann. Aufgrund der Frequenzauflösung des Ausgangssignals von 1Hz (siehe Tabelle 1) ist es notwendig, die Taktquelle ebenfalls sehr fein einzustellen zu können. Mit dieser erweiterten Anforderung für das externe Taktsignal konnte die bisher verwendete Lösung auf Basis eines VHDL Designs [14] für einen CPLD [15], welcher somit als  $2^x$ -Teiler für ein hochfrequentes Taktsignal fungiert hat, nicht mehr verwendet werden. Als Alternative wurde daher der bereits erwähnte Clock Generator Si5351 ausgewählt, der die erweiterten Anforderungen erfüllt. Auf diese Weise können alle Probleme mit dem AD9102 umgangen werden, wobei der Software-seitige Aufwand hinsichtlich der Ansteuerung der Bausteine deutlich gestiegen ist.

Da mit der finalen Schaltung bezüglich der Taktquelle und der internen Handhabung von Dreieck- und Sägezahn-Signalen die Funktionalität, Signale in Form von digitalen Sample-Werten mit dem AD9102 zu generieren, bereits implementiert war, wurde die Erzeugung des Rechteck-Signals für die finale Version der Platine nochmals überarbeitet. Die bisherige Umsetzung mit einer optional zuschaltbaren

Schaltung wurde komplett verworfen. Stattdessen werden Rechtecksignale ebenfalls mithilfe von Samples generiert, welche im Mikrocontroller berechnet und im SRAM des AD9102 gespeichert werden. Damit hat sich die Anzahl der erforderlichen Komponenten der gesamten Schaltung deutlich reduziert, was vor allem das Erstellen des Platinen-Layouts vereinfacht hat.

In Abbildung 10 ist die gesamte Hauptplatine mit allen Komponenten als 3D-Modell zu sehen. Insgesamt stellt die Hauptplatine mit dem Logikteil und den beiden Schaltungen zur analogen Signalerzeugung das Herzstück des Funktionsgenerators dar. Im Zusammenspiel mit dem Netzteil zur Erzeugung der benötigten Versorgungsspannungen und dem Frontpanel, welches sich um die gesamte Benutzerinteraktion kümmert, wird die komplette Hardware des Geräts gebildet. Diese ist im Stande, alle Hardware-relevanten Anforderungen aus Kapitel 2.1 zu erfüllen. Zusammen mit der in Kapitel 4 beschriebenen Firmware für den verbauten Mikrocontroller entsteht dann der voll funktionsfähige Funktionsgenerator.

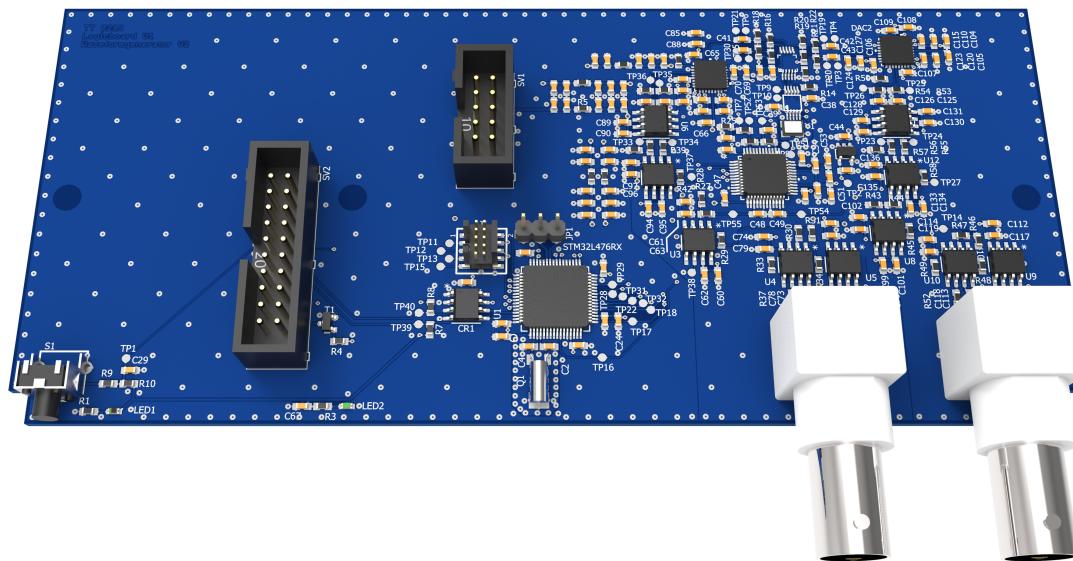


Abb. 10: Hauptplatine als 3D-Modell

## 4 Software

Neben der Hardware in Form des Platinen-Systems war eine umfangreiche Firmware für den auf der Hauptplatine verbauten Mikrocontroller zu entwickeln. Diese hat die Aufgabe, sämtliche Komponenten des Funktionsgenerators anzusteuern und die Interaktion mit dem Anwender zu koordinieren. Wie bereits in Kapitel 2.3 erwähnt, wurde die Firmware unter Verwendung eines bestehenden Code-Gerüsts entwickelt. Dieses bildet ein Schichtenmodell, welches den Zugriff auf die zugrundeliegende Hardware weitestgehend abstrahiert und durch einen Event-gesteuerten Programmablauf kooperatives Multitasking ermöglicht. Dieses Schichtenmodell wird in Kapitel 4.1 genauer beschrieben. Auf dieser Basis aufbauend wurde für jede logische Funktionsgruppe eine eigene Software-Komponente entworfen, die unabhängig einer fest definierte Aufgabe übernimmt und für höhere Schichten ein Interface bereitstellt, auf diese Funktionalität in abstrakter Weise zugreifen zu können. Die einzelnen Komponenten werden in Kapitel 4.2 jeweils aufgearbeitet. Unter Verwendung dieser Komponenten ist der eigentliche Programmablauf in Form einer Zustandsmaschine entwickelt worden, der die Interaktion des Benutzers mit dem Gerät steuert und die resultierenden Aufgaben an die entsprechenden Komponenten übergibt. Diese Anwendungs-Schicht wird in Kapitel 4.3 erläutert.

### 4.1 Hardware-Abstraktion in Form eines Schichtenmodells

Das bereits vorhandene Code-Gerüst basiert grundsätzlich auf der Arbeit *bare-metal-cpp* von Alex Robenko [16]. In diesem E-Book beschreibt er die Idee und die Funktionsweise des Schichtenmodells. Durch die Aufteilung der Firmware auf die vier Schichten *Device*, *Driver*, *Component* und *Application* kann gewisse Funktionalität für den Rest der Software in wenige Funktionsaufrufe abstrahiert werden, was die Struktur des Gesamtsystems und die Implementierung von darauf aufbauenden Funktionen deutlich vereinfacht. Die vier Schichten erfüllen unterschiedliche Aufgaben und sind in Abbildung 11 dargestellt. Dabei baut eine Schicht immer auf die darunterliegende auf und stellt ein abstrahiertes Interface für die nächst-höhere Schicht bereitstellt. Neben den Schichten an sich, die jeweils aus mehreren in C++ geschriebenen Klassen bestehen und im Folgenden nacheinander genauer beschrieben werden, existieren noch zwei weitere Klassen innerhalb des Modells mit den Namen *System* und *EventLoop*. Diese bringen zusätzliche Funktionalität in das System ein oder strukturieren den Programmablauf noch weiter. Diese Zusatzmodule werden nach den vier Schichten ebenfalls genauer erläutert.

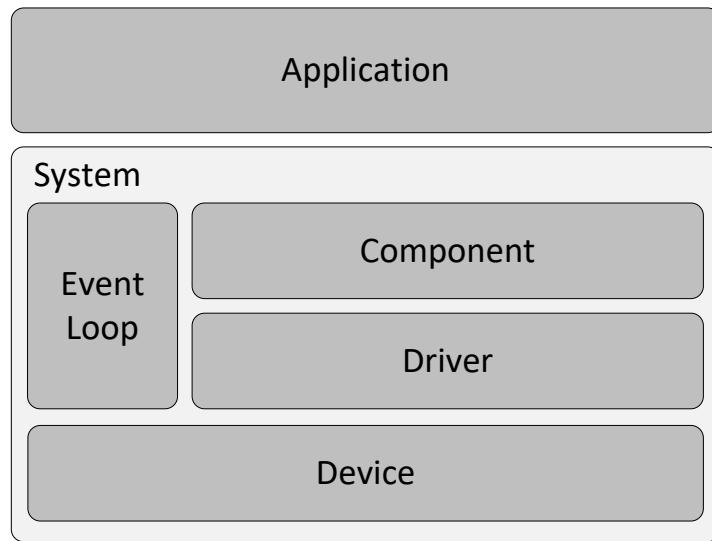


Abb. 11: Struktur des Schichtenmodells

### Device

Die *Device*-Schicht bildet die unterste Schicht im Modell. Hier wird direkt auf die Register der Hardware zugegriffen und sämtliche Konfiguration vorgenommen. Dabei existiert für jede Hardware-Peripherie des Mikrocontrollers eine eigene C++ Klasse, die durch öffentliche Methoden den Zugriff auf die jeweiligen Hardware-Funktionen abstrahiert. Zudem kann durch mehrere Instanzen derselben Klasse baugleiche, aber unabhängige Peripherie der Hardware ohne zusätzlichen Aufwand in Betrieb genommen werden. Es ist beispielsweise mit der Klasse *HardwareGpio* möglich, durch mehrere Instanzen die Baugruppen GPIOA, GPIOB und GPIOC unabhängig voneinander einzusetzen, ohne neuen Programmcode für die entsprechende Peripherie erstellen zu müssen. Ändert sich die zugrundeliegende Hardware, muss lediglich die Implementierung der einzelnen Device-Klassen angepasst werden. Durch die unveränderten Klassen-Methoden bleibt der darauf basierende Code höherer Schichten von Änderungen befreit.

### Driver

Die nächst-höhere Schicht des Modells trägt den Namen *Driver*. Klassen-Instanzen dieser Ebene abstrahieren eine vom Gesamt-System isoliert betrachtbare Funktionsgruppe. So existiert zum Beispiel eine Klasse *IoPin*, bei der jede Instanz genau einen I/O Pin des Mikrocontrollers ansteuert. Die Klasse *I2cSlaveDriver* wiederum beschreibt einen I<sup>2</sup>C-Slave mit eindeutiger Adresse, der Teil eines I<sup>2</sup>C-Busses mit dem Controller als Master ist. Intern verwendet jedes Driver-Objekt ein oder mehrere Objekte der Device-Schicht mit deren öffentlichen Interfaces. Folglich ist

die Driver-Schicht unabhängig von der Implementierung der Device-Objekte, was die Driver-Klassen universell einsetzbar macht.

## Component

Die nächste Ebene des Abstraktionsmodells ist die *Component*-Schicht. In dieser Ebene werden mehrere Funktionsgruppen in logische Einheiten verpackt, um eine nach außen hin geschlossene Funktionalität bereitzustellen. Die einzelnen Elemente sind Objekte der Driver-Schicht. Implementiert sind die Einheiten als eigenständige Klassen, die durch öffentliche Methoden ein abstrahiertes Interface bereitstellen und intern möglichst viel Funktionalität bündeln. Somit kann durch das Erstellen mehrerer Components jede Aufgabe des Controllers getrennt modelliert werden.

## System

Mit der *System*-Schicht, welche im Modell in Abbildung 11 folgt, wird nur bedingt ein weiteres Abstraktionslevel gebildet. Sie kann eher als Zusatzmodul innerhalb des Systems gewertet werden, da sie die Funktion hat, sämtliche Objekte der darunterliegenden Schichten zu instantiiieren und somit in sich zu bündeln. Nur Objekte, auf die in der nächst-höheren Schicht zugegriffen werden muss, werden durch Methoden der System-Klasse verfügbar gemacht, alle anderen sind nicht zugänglich. Somit arbeitet die System-Schicht, bestehend aus nur einer einzelnen Klasse, als eine Art Wrapper für die unteren Schichten. Der Anwender erhält nur Zugriff auf Teile der Firmware, die er auch benötigt. Alles andere ist innerhalb der System-Schicht abstrahiert.

## Application

Die höchste Schicht des Modells trägt den Namen *Application* und bildet die tatsächliche Anwendungsschicht. Hier wird unter Verwendung der einzelnen Components der gesamte Ablauf der Firmware definiert. Der Zugriff auf die Objekte erfolgt wie bereits erwähnt über die System-Schicht. Je nach Komplexität des Gesamt-Systems kann diese Schicht mehr oder weniger komplex ausfallen.

## EventLoop

Der *EventLoop* ist genau betrachtet keine Schicht des Modells, sondern ein Zusatzmodul. Es handelt sich dabei um eine einzelne Klasse, die ebenfalls innerhalb der System-Schicht instantiiert wird und durch Zugriff auf die Schichten Device, Driver und Component den gesamten Programmablauf koordiniert. Das hier beschriebene Schichtenmodell, auf dem die Firmware des Mikrocontrollers aufbaut,

basiert auf einem Ereignis-gesteuerten Programmfluss. Das heißt, dass die Firmware bei Eintreten eines bestimmten Ereignisses eine fest definierte Handlung ausübt und anschließend auf das nächste Ereignis wartet. Bei den Ereignissen handelt es sich allgemein um Interrupts des Controllers, bei den Handlungen um *Callback-Funktionen*, die von den einzelnen Schichten festgelegt werden. Wichtig hierbei ist, dass die potentiell sehr umfangreichen und komplexen Callbacks nicht direkt aus dem Interrupt-Kontext ausgeführt werden, sondern einer Warteschlange hinzugefügt werden, die im normalen Programm-Kontext abgearbeitet wird. Somit wird sichergestellt, dass die Interrupt Routinen sehr kurz bleiben und sich nicht gegenseitig blockieren oder aufgrund unterschiedlicher Prioritäten ausschließen. Durch die Verwendung einer Warteschlange mit FIFO-Prinzip wird der Programmablauf nicht beeinflusst. Mit dieser Struktur wird eine hohe Performance und Parallelität der Firmware bei hoher Stabilität geschaffen. Innerhalb der EventLoop-Klasse wird diese Funktionalität geschaffen, wobei durch entsprechende Methoden auf Elemente wie die Warteschlange zugegriffen werden kann.

Durch die Abstraktion in verschiedene Schichten war nach Anpassung der Device-Klassen an den verwendeten Mikrocontroller ein solides Software-Fundament geschaffen, mit dem auf sämtliche Hardware-Funktionen zugegriffen werden kann. Daraufhin konnte dann mit der Implementierung der benötigten Components und der Application-Schicht begonnen werden.

## 4.2 Entwickelte Komponenten der Components-Schicht

Auf Basis des Schichtenmodells, welches in Kapitel 4.2 beschrieben wird, wurden innerhalb der Components-Schicht mehrere Software-Komponenten entwickelt. Diese sind unabhängig für eine bestimmte Aufgabe des Mikrocontrollers zuständig und stellen durch einige wenige Klassen-Methoden eine abstrahierte Schnittstelle zu ihrer Funktionalität für höhere Schichten bereit.

In den folgenden Kapiteln werden jeweiligen Komponenten im Detail beschrieben. Dafür wird auf die jeweils verwendeten Driver- und Device-Objekte sowie auf die Methoden zur Benutzung der Komponente eingegangen. Grundsätzlich lassen sich die entstandenen Klassen in zwei Gruppen aufteilen. Für die Ansteuerung der Baugruppen des Frontpanels und die allgemeine Benutzerinteraktion sind die Komponenten *Display*, *Encoder*, *ExternalEEPROM* und *FourButtonArray* entwickelt worden. Für die Erzeugung der analogen Ausgangssignale wurden die Komponenten *FrequencyController*, *SupplyVoltageGenerator*, *DirectDigitalSynthesizer* und *SignalGenerator* entworfen.

### 4.2.1 Benutzereingabe durch Taster am Frontpanel

Für Eingaben des Anwenders sind auf dem Frontpanel insgesamt acht Taster verbaut. Wie bereits in Kapitel 3.3 beschrieben, sind die Taster in drei Gruppen aufgeteilt. Vier Stück sind unterhalb des Displays platziert und besitzen eine variable Funktion. Jeweils zwei der restlichen vier sind einem der beiden Ausgangskanäle zugeordnet. Sie haben die Funktionen, das entsprechende Menü zum Verändern der Signal-Parameter innerhalb der Benutzeroberfläche zu aktivieren und das Signal am Ausgang zu aktivieren. Aus Hardware-Sicht werden zum einen die vier Display Buttons und zum anderen die Channel-Select und Channel-Activate Buttons inklusive der LEDs jeweils mithilfe eines I<sup>2</sup>C Port-Expanders angesteuert. Aus diesem Grund wurde diese Trennung auch innerhalb der Software-Ansteuerung übernommen. Mithilfe der entwickelten Komponente *FourButtonArray* werden jeweils die vier Taster eines Port-Expanders in einer Klasse modelliert. Diese Klasse wird im Rahmen der Firmware zweimal instantiiert und somit sowohl für die Dispaly Buttons als auch für die beiden Channel Button-Paare verwendet. Intern benutzt die Komponente zwei Driver-Klassen. Damit kann einerseits durch die Methoden des *I2cSlaveDriver* auf die Kommunikation mit dem Port-Expander über den globalen I<sup>2</sup>C-Bus zugegriffen werden, andererseits wird mithilfe des *IoPinDriver* ein Interrupt am Event-Pin des Port-Expanders abgefangen und über den EventLoop verarbeitet. Sobald einer der vier Taster gedrückt wird, was den eben erwähnten Interrupt zur Folge hat, erfolgt

ein asynchrones Auslesen des Daten-Registers des Port-Expanders. Damit kann bestimmt werden, welcher Taster gedrückt wurde, was das Ausführen einer von höheren Schichten zuvor festgelegten Handler-Methode über den EventLoop zur Folge hat. In den Tastern sind jeweils eine rote und eine grüne LED verbaut, die ebenfalls über den Port-Expander angesteuert werden. Somit sind vier Farben der Taster möglich: Rot, Grün, Orange und Weiß (keine LED an). Mithilfe einer Methode der FourButtonArray-Klasse kann die Farbe für jeden Taster getrennt eingestellt werden. Intern werden dazu die entsprechenden Pins des Port-Expanders über die I<sup>2</sup>C-Schnittstelle konfiguriert. Insgesamt wird mithilfe dieser Klasse eine simple Ansteuerung der acht Taster des Frontpanels ermöglicht. Höhere Schichten müssen lediglich entsprechende Callbacks festlegen, die bei einem Tastendruck ausgeführt werden. Zudem kann flexibel die Farbe der Taster verändert werden und dem aktuellen Stand der Benutzeroberfläche angepasst werden.

#### 4.2.2 Benutzereingabe durch Dreh-Encoder am Frontpanel

Die Komponente des Encoders verwendet die beiden Treiber *RotaryEncoderDriver* und *IoPinDriver*. Der *RotaryEncoderDriver* fügt je nach Drehung des Encoders nach links oder rechts im Eventloop den Callback für Links- oder Rechtsdrehung hinzu. Der Treiber stellt außerdem zwei Funktionen bereit, mit welchen die beiden Callback-Handler überladen werden können. Der *IoPinDriver* behandelt die gleiche Funktion für das Drücken des Encoders und fügt dem Eventloop den vorher definierten Callback-Handler bei Betätigung des Encoders hinzu. Das Überladen der jeweiligen Callbacks ist über die Funktionen `void addPressedHandler(...)`, `void addRotateLeftHandler(...)` und `void addRotateRightHandler(...)` möglich. Die Encoder-Komponente nutzt diese Treiber zur Nutzereingabe des Funktionsgenerators um u.a. die Einstellungen der Kanäle zu verändern. Dabei ist der Encoder zunächst deaktiviert und wird nur beim Betreten eines Submenüs (siehe Kapitel 4.3.6) aktiviert. Beim Betreten wird außerdem der aktuelle Parameter des Submenüs (z.B. Frequenz) zur Encoder-Komponente mit Hilfe der Funktion `void calculateDigitsFromValue(...)` übergeben. Diese wandelt den übergebenen Wert in 3 Ziffern (Digits) mit Position des Kommas, Vorzeichen und Einheitenfaktor um. Dieses Prinzip zur Umwandlung ist in Abbildung 12 dargestellt und ist zur Visualisierung der einzelnen Stellen am Display notwendig und zusätzlich für das Prinzip der Veränderung eines Wertes notwendig.

Im aktivierte Zustand besitzt der Encoder die zwei verschiedenen Modi *changeDigit* und *changePosition*. Im Modus *changePosition*, welcher beim Betreten eines

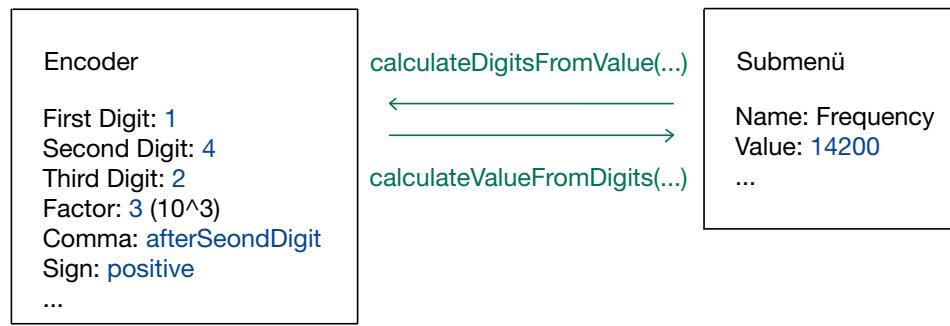


Abb. 12: Prinzip der Umwandlung zwischen Submenü-Parameter und Encoder

Submenüs ausgewählt ist, kann bestimmt werden welche der drei Ziffern verändert werden soll. Durch Drehen des Encoders wird die entsprechende Ziffer ausgewählt, was durch einen Strich unter einer der Ziffern im Display angezeigt wird (siehe Abbildung 13). Durch Drücken des Encoders kann der Mode zu *changeDigit* verändert werden, in welchem das Ändern der ausgewählten Ziffer ermöglicht wird. Visualisiert wird dies durch die pixelweise Invertierung der ausgewählten Ziffer im Display, wie in Abbildung 14 dargestellt. Durch Drehen nach links wird der Wert der Ziffer um eins verringert und durch Drehen nach rechts um eins erhöht. Die nächst höherwertige Ziffer wird bei einem Über- oder Unterlauf ebenfalls entsprechend verändert, wodurch dem Anwender eine angenehmen Änderungen des Wertes ermöglicht wird. Die beiden Funktionen eines Submenüs zur Multiplikation des Wertes mit 10 bzw. das Teilen durch 10 (siehe Kapitel 4.3.6) wird ebenfalls in der Encoder-Komponente umgesetzt.



Abb. 13: Encoder im changePosition Modus



Abb. 14: Encoder im changeValue Modus

Die Komponente stellt außerdem die Funktion Tvalue `calculateValueFromDigits(...)` bereit, welche aus den Ziffern, dem Vorzeichen und der aktuellen Position des Kommas einen neuen Wert für den Parameter des aufrufenden Submenüs berechnet und anschließend zurückgibt. Das Prinzip zur Berechnung des neuen Wertes eines Submenüs aus den einzelnen Parametern des Encoders ist ebenfalls in Abbildung 12 dargestellt und funktioniert umgekehrt zur `void calculateDigitsFromValue(...)` Funktion.

### 4.2.3 Visualisierung mittels Grafik-Display

Für die Darstellung der eingestellten Parameter der jeweiligen Kanäle wurde, wie in Kapitel 3.3 erwähnt, ein Punktmatrix Display verwendet, welches über eine Display-Komponente gesteuert wird.

#### Steuerung der Datenübertragung

Diese Komponente verwendet einen *SpiSlaveDriver* um mit der Hardware zu kommunizieren. Der Treiber stellt die Funktion `void asyncWrite(...)` bereit, durch welche die Daten an das Display gesendet werden können. Die sequentielle Koordinierung der zu sendenden Nachrichten wird von einem *SpiMasterBusManager* verwaltet. Da die verwendete SPI-Peripherie auf Grund der hohen Datenrate ausschließlich für das Display reserviert wurde, ist ein eigener *SpiMasterBusManager* für die Nachrichten des Display-*SpiSlaveDriver* implementiert.

Zusätzlich zur seriellen Schnittstelle wurde ein Pin des Mikrocontrollers mit dem Display verbunden, welcher die Behandlung der übertragenen Daten als Steuerkommando oder Displayinhalt definiert. Ein Steuerkommando entspricht dabei z.B. einem Sprung zu einer anderen Stelle der Datenbeschreibung des Displays. Der Displayinhalt codiert die Pixel des Displays, wobei ein Byte immer zwei Pixel untereinander im Display entsprechen. Das niederwertigere Nibbel des Bytes entspricht dabei immer der oberen Zeile und das höherwertige Nibbel dem Pixel in der unteren Zeile. Dieses Prinzip ist in Abbildung 15 für die ersten beiden Spalten dargestellt.



Abb. 15: Zusammenhang der Nutzdatenbytes mit Pixelposition im Display

#### Verändern des anzuzeigenden Inhalts

Der Ursprung des Koordinatensystems zur Beschreibung des Displays befindet

sich im linken oberen Eck und hat in horizontaler Richtung den maximalen Wert von 240 und in vertikaler Richtung eine Begrenzung bei 128. Durch die Codierung eines jeden Pixel mit 4 Bit können 16 unterschiedliche Helligkeitswerte pro Pixel eingestellt werden. Die Komponente wurde so realisiert, dass alle Helligkeitswerte verwendet werden können. Allerdings beschränkt sich das User-Interface lediglich auf die Verwendung des niedrigsten (aus) und des höchsten Helligkeitswerts (an). Auf Grund der Aufteilung von zwei Pixeln pro Byte wurde für das 240x128 Pixel große Display ein Byte-Array der Größe 240x64 im Speicher des Mikrocontrollers angelegt. In diesem Array wird der anzuzeigende Inhalt des Displays gespeichert und verändert.

Um Text auf dem Display ausgeben zu können, wurde eine zusätzliche Header-Datei erstellt, welche alle Buchstaben, die Zahlen 0-9 und einige Sonderzeichen in zwei unterschiedliche Schriftgrößen enthält. Die Zeichen wurden dabei in jeweils zwei unterschiedlich großen Byte-Arrays gespeichert, welche die Pixelinformation aller Symbole der Größe 5x7 und 7x11 enthalten. Die Display Komponente stellt für die Ausgabe eines Textes die Funktion `uint8_t printText(...)` bereit, welche den übergebenen String als Text an die x- und y-position des Arrays speichert. Zum Beschreiben des Displayinhalts wurden außerdem die Funktionen `uint8_t printNumber(...)`, `uint8_t printSign(...)` und `uint8_t printDigits(...)` implementiert, welche jeweils unterschiedliche Symbole ausgeben. Diese Funktionen geben alle eine Position für ein darauffolgendes Zeichen zurück, sodass auch eine Kombination der Funktionen zur Darstellung von Zahlen, Sonderzeichen und Text verwendet werden kann. Zur logischen Abgrenzung von Menübereichen ist weiterhin die Funktion `void drawLine(...)` entstanden, welche eine vertikale oder horizontale Linie mit variabler Länge an die übergebene Position platziert. Dies wurde zur Aufteilung des Displayinhalts mit Menüheader, Menüinhalt und der Separierung der variablen Buttons verwendet.

### Schreiben der Daten vom Speicher auf das Display

Nachdem der Inhalt des Arrays im Speicher des Mikrocontrollers verändert wurde, kann entweder das komplette oder nur einige Zeilen des Displays über die Funktionen `void reloadRows(...)` oder `void reloadContent(void)` neu geladen werden. Ein neues Laden des kompletten Displayinhalts ist z.B. bei einem Wechsel des Menüs notwendig. Einzelne Zeilen werden u.a. beim Wechsel der Menüseite im Bereich der variablen Buttons und im Menüheader verwendet. Zudem wurden außerdem noch die Funktionen `void reloadHeaderContent(...)` und `void reloadButtonContent(...)` implementiert, welche die für den Funktionsgenera-

tor definierten Menübereiche *Header* und *variable Buttons* einzeln überladen. Dies erleichtert die Handhabung des Inhalts bei der Implementierung des *User Interface*. Daneben sind Funktionen mit den Namen `void clearArea(...)` und `void invertArea(...)` entstanden, um die Anzeigemöglichkeiten beim Entwickeln des User Interface zu erweitern.

Die Display-Komponente steuert zusätzlich mit der Funktion `void setContrast(...)` die Kontrasteinstellung indem es den übergebenen Wert als Steuerkommando an das Display sendet. Diese Funktion wird im Kontrast-Submenü (siehe Kapitel 4.3.6) des Funktionsgenerators aufgerufen. Weiterhin wird die rote und die grüne Hintergrundbeleuchtung durch diese Komponente mit Hilfe der jeweiligen PWM-Treiber und den bereitgestellten Funktionen `void setBGred(...)` bzw. `void setBGgreen(...)` gesteuert.

#### 4.2.4 Speichern der Benutzereinstellungen

Um sämtliche Einstellungen des Anwenders über einen Neustart des Funktionsgenerators hinweg speichern zu können, wurde ein externer EEPROM auf der Hauptplatine verbaut (siehe Kapitel 3.4.1). Software-seitig wird er mithilfe der Klasse *ExternalEEPROM* angesteuert, wobei diese die beiden Driver-Klassen *I2cSlaveDriver* für die Kommunikation via I<sup>2</sup>C und *IoPinDriver* zum Steuern des Write-Control Pins verwendet. Im wesentlichen stellt die Klasse zwei Methoden zum Lesen und Schreiben der Parameter und Einstellungen bereit, wobei die Daten in einem global definierten Format vorliegen. Zudem besteht die Möglichkeit, nur einzelne Teile der Daten auszulesen oder zu aktualisieren, was vor allem die Effizienz steigert. Insgesamt verbessert diese eher kleine Software-Komponente das Benutzer-Erlebnis des Funktionsgenerators erheblich.

#### 4.2.5 Einstellen der Eingangsfrequenz des Signalgenerator ICs

In Kapitel 3.4.2 wird erklärt, warum die Eingangsfrequenz für den IC zur Signalerzeugung AD9102 variabel veränderbar sein muss. Zudem wird erwähnt, dass dies mithilfe des Bausteins Si5351 realisiert wird, welcher zwei unabhängige und einstellbare Frequenzen für die beiden Ausgangskanäle erzeugt. Dieser Chip wird über den globalen I<sup>2</sup>C-Bus als Slave angesteuert und so konfiguriert. Innerhalb der Mikrocontroller-Firmware erfolgt dies in der Klasse *FrequencyController* der Component-Schicht. Diese verwendet den bereits erwähnten *I2cSlaveDriver*, um mit wenigen Funktionen auf die I<sup>2</sup>C-Kommunikation zugreifen zu können. Für den Zugriff von außen stellt die Klasse neben einer Funktion zur Initialisierung lediglich die Methode

`uint32_t setFrequencyForChannel(channelNo, targetFrequency)` bereit. Die Übergabeparameter der Funktion bestehen aus dem betreffenden Kanal und den beiden Parametern Frequenz und Phase, die das finale Ausgangssignal des Funktionsgenerators annehmen soll. Daraus werden mithilfe eines selbst entwickelten Algorithmus die optimale Eingangsfrequenz für den AD9102 und die dafür benötigten Parameter des Si5351 berechnet und entsprechend über I<sup>2</sup>C konfiguriert. Hierbei wird darauf geachtet, dass die Anzahl der Sample-Werte für den SRAM (siehe Kapitel 3.4.2) einen akzeptablen Signalverlauf mit minimalen Phasenfehler ermöglicht, ohne einen Maximalwert zu überschreiten. Die neue Eingangsfrequenz wird dann als Rückgabeparameter für weitere Schritte zur Verfügung gestellt (siehe Kapitel 4.2.7). Somit muss von außen lediglich die gewünschte Frequenz am Kanalausgang vorgegeben werden, was die Anwendung dieser Komponente deutlich vereinfacht.

#### 4.2.6 Erzeugen zusätzlich benötigter Spannungen

Neben dem Clock Generator Si5351 gibt es noch eine weitere Baugruppe, die von den beiden Ausgangskanälen gemeinsam verwendet wird. Um die benötigten Hilfsspannungen für die Einstellung der Signalamplitude und der Offset-Spannung beider Kanäle generieren zu können, kommt auf der Hauptplatine ein Vier-Kanal Digital-Analog-Wandler zum Einsatz (siehe Kapitel 3.4.2). Dieser wird via SPI vom Mikrocontroller aus angesteuert und mit Daten versorgt, was Software-intern mit der Klasse *SupportVoltageGenerator* realisiert wird. Diese stellt die beiden Methoden `setAmplitudeVoltage(...)` und `setOffsetVoltage(...)` bereit, mit denen die vier Spannungen getrennt voneinander eingestellt werden können. Die Übergabeparameter werden intern in den entsprechenden Wertebereich konvertiert und anschließend mithilfe der Driver-Klasse *SpiSlaveDriver* an den IC gesendet. Mit dieser im Vergleich zu den restlichen eher kleinen Komponente wird die Generierung der Hilfsspannungen hinter Funktionsaufrufen abstrahiert, wodurch die gesamte Ansteuerung der Schaltung zur Signalerzeugung übersichtlicher und strukturierter wird.

#### 4.2.7 Erzeugen des Ausgangssignals

Die Kernkomponente der analogen Signalerzeugung ist der IC AD9102 von Analog Devices. Mithilfe einer extern bereitgestellten Taktquelle als Eingangsfrequenz und einer Spannungsreferenz kann durch Konfiguration interner Register via SPI ein differenzielles Stromsignal erzeugt werden. Die Ansteuerung der Taktquelle und der Spannungsreferenz wurden bereits in den Kapiteln 4.2.5 und 4.2.6 beschrieben. In diesem Kapitel wird nun auf die Implementierung der Klasse *DirectDigitalSyn-*

*thesizer* genauer eingegangen, welche mittels zweier Instanzen für die Steuerung der beiden AD9102 ICs zuständig ist. Intern wird zum einen der bereits erwähnte *SpiSlaveDriver* für den Zugriff auf die SPI Kommunikation verwendet. Um die Generierung eines Ausgangssignals zu starten, ist eine fallende Flanke an einem der Pins des AD9102 notwendig, was zudem für die Synchronisierung der beiden sonst unabhängigen Kanäle verwendet wird. Daher wird zum anderen noch der *IoPin-Driver* für die Steuerung dieses Pins innerhalb der Klasse benutzt. Nach außen hin stehen neben einer Initialisierungsfunktion zwei Methoden bereit, die einerseits die Signalerzeugung des Kanals aktivieren, was für die Funktionalität der Channel Active Buttons des Frontpanels verwendet wird (siehe Kapitel 3.3), andererseits den Stromausgang mit den gewünschten Parametern konfigurieren. Intern werden dazu auf Basis des in Kapitel 3.4.2 beschriebenen finalen Vorgehens zur Generierung von Signalen die entsprechenden Register konfiguriert und gegebenenfalls neue Sample-Werte berechnet und via SPI in den SRAM des AD9102 geschrieben. Somit können alle erforderlichen Signalformen mit den gewünschten Parametern erzeugt und ausgegeben werden.

#### 4.2.8 Bündelung der analogen Signalerzeugung

Um die Ansteuerung der analogen Signalerzeugung von höheren Schichten so einfach wie möglich zu gestalten, wurden die drei Komponenten *FrequencyController*, *SupportVoltageGenerator* und *DirectDigitalSynthesizer* in einer zusätzlichen Komponente gebündelt. Die hierfür entwickelte Klasse trägt den Namen *SignalGenerator* und wird zweimal je für einen Ausgangskanal innerhalb der Mikrocontroller-Firmware instantiiert. Sie stellt mehrere Methoden bereit, die jeweils einen Parameter des Signals verändern (beispielsweise `setWaveform(...)` oder `setFrequency(...)`), welche aus den höheren Schichten zum Einstellen der Ausgangssignale verwendet werden. Zudem gibt es neben einer Initialisierungsfunktion eine Methode zum Aktivieren und Deaktivieren des Ausgangssignals. Intern werden die je nach Funktion erforderlichen Aufgaben an die drei Komponenten zur tatsächlichen Ansteuerung der Hardware-Bausteine weitergegeben. Somit wird die Konfiguration der Ausgangssignale für die höheren Schichten auf nur eine Schnittstelle begrenzt, was das Design erheblich vereinfacht hat.

## 4.3 Entwicklung des Betriebssystems innerhalb der Application-Schicht

Innerhalb der Application-Schicht des Modells (siehe Kapitel 4.1) wurde die eigentliche Firmware inklusive dem Programmablauf erstellt. Dazu kommt eine *State Machine* zum Einsatz, welche die Interaktion mit dem Anwender koordiniert und intern je nach Eingabe entsprechende Schritte einleitet. Auf Basis dieser Aufgabe wurde der übergeordnete Name *User Interface* gewählt. Für die Implementierung werden Instanzen der Komponenten *FourButtonArray*, *Display*, *Encoder*, *ExternalEEPROM* und *SignalGenerator* aus Kapitel 4.2 verwendet. Das Zusammenspiel der einzelnen Komponenten und die Steuerung der verschiedenen Zustände des *User Interface* durch einen *MenuController* wird im Folgenden erläutert.

### 4.3.1 State Machine

Die *State Machine* entspricht dem Grundgerüst des *User Interface* und wird durch den *MenuController* (siehe Kapitel 4.3.7) gesteuert. Die einzelnen Zustände entsprechen den unterschiedlichen Menüs mit den Namen *Mainmenu*, *Channel1*, *Channel2* sowie *Bootscreen*, auf welche in den folgenden Unterkapiteln noch genauer eingegangen wird. Der *MenuController* startet nach dem Einschalten des Geräts zunächst mit dem *Bootscreen* Menü, welches in Kapitel 4.3.3 näher beschrieben wird. Innerhalb dieses Zustandes werden alle Nutzereingaben durch die Taster und den Encoder ignoriert. Nachdem das *Bootscreen* Menü für drei Sekunden angezeigt wurde, wird in das Hauptmenü gewechselt und der eigentliche Ablauf der State Machine wird gestartet. Beim Betreten des Hauptmenüs werden die Komponenten zur Benutzerinteraktion, mit Ausnahme des Encoders, zum ersten mal aktiviert.

Das Wechseln der States wird durch die beiden *ChannelSelect*-Buttons über eine Instanz der Komponente *FourButtonArray* (siehe Kapitel 4.2.1) realisiert. Der linke Taster ist dabei Channel 1 zugeordnet, der rechte entsprechend Channel 2. Durch Drücken innerhalb des Hauptmenüs gelangt man in das jeweilige Menü des Kanals. Ein erneutes Betätigen führt zurück in das Hauptmenü. Da die Taster, wie in Kapitel 3.3 erwähnt, jeweils die drei Farben rot, grün und gelb annehmen können, wurde jedem Menü eine eigene Farbe zugewiesen. Channel 1 entspricht dabei der Farbe grün, Channel 2 der Farbe rot und das Hauptmenü der Farbe gelb. Die beiden *ChannelSelect*-Buttons bewirken also je nach Zustand das Aufrufen eines bestimmten Menüs, was durch die entsprechende Farbe der Taster visualisiert wird. So sind diese beiden Taster im State *Mainmenu* grün und rot, im State *Channel1* gelb und rot und im State *Channel2* grün und gelb. Die vier Zustände mit den jeweiligen

Übergangsbedingungen und den dazu gehörigen Farben der Taster sind in Abbildung 16 dargestellt.

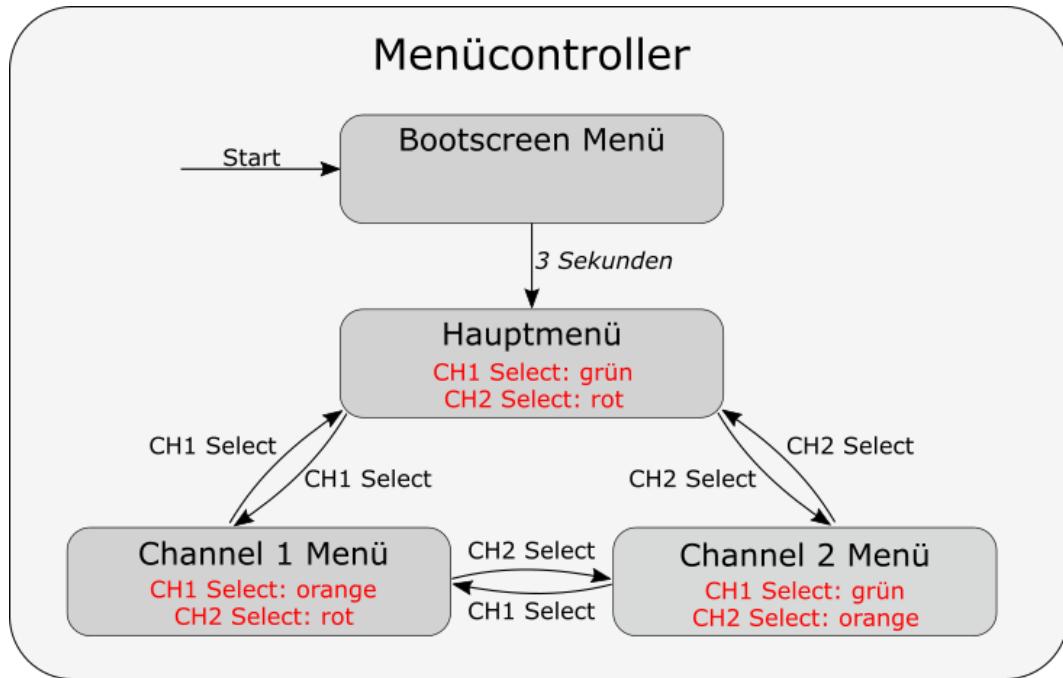


Abb. 16: Aufbau der State Machine mit Übergangsbedingungen im *MenuController*

### 4.3.2 MenuBase

Die vier entstandenen States der Zustandsmaschine, welche den Menüs *Mainmenu*, *Channel1*, *Channel2* und *Bootscreen* entsprechen, werden mithilfe der Klassen *MainMenu* (Kapitel 4.3.4), *ChannelMenu* (Kapitel 4.3.5) sowie *BootscreenMenu* (Kapitel 4.3.3) modelliert. Diese erben wiederum alle von der Klasse *MenuBase*, um eine leichtere Handhabung der unterschiedlichen Menüs innerhalb des *MenuControllers* zu ermöglichen. Durch die Basis-Klasse werden die Methoden `enterMenu(...)`, `exitMenu(...)` und `showOverview(...)` bereitgestellt, die vom *MenuController* verwendet werden. So wird trotz unterschiedlichem Aufbau der Menüs eine einheitliche Schnittstelle zur Steuerung sichergestellt.

Zusätzlich werden in der Klasse *MenuBase* Referenzen auf die von allen Menü-Instanzen verwendeten Komponenten *Display*, *Encoder*, *externalEEPROM* und des *FourButtonArray* für die variablen Taster unter dem Display definiert. Darüber hinaus wird die Adresse der Daten eines Menüs innerhalb des externen EEPROM gespeichert und ein Pointer auf das aktuell aufgerufene Submenü (siehe Kapitel 4.3.6) erzeugt.

### 4.3.3 BootscreenMenu

Das *Bootscreen* Menü wird beim Einschalten des Funktionsgenerator als erster Zustand der *State Machine* durch den *MenuController* aufgerufen. Dieses Menü erbt, wie alle anderen, von der *MenuBase* aus Kapitel 4.3.2, wodurch die wichtigsten Funktionen und Komponenten bereits definiert sind. Zusätzlich besitzt dieses Menü eine Referenz auf eine Timer-Komponente. Dieses Menü ist dafür zuständig, einen Startbildschirm beim Einschalten des Funktionsgenerators anzuzeigen. Hier werden die Namen des Projekts und der Teilnehmer sowie das Semester, in welchem der Funktionsgenerator entstanden ist, im Rahmen der Funktion `enterMenu(...)` angezeigt. Im unteren Bereich des Displays befindet sich die Visualisierung eines Ladebalkens, welcher durch 200 Schritte und mit festen zeitlichen Abständen durch die Timer-Komponente aufgefüllt wird. Dies erfolgt in einer Zeit von drei Sekunden, währenddessen alle Taster sequentiell zunächst auf die Farbe grün, anschließend auf gelb und zuletzt auf rot geschaltet werden. Dies soll dem Anwender dabei helfen, die Funktion der LEDs in den Tastern zu überprüfen. Das Bootscreen Menü wird nach diesem visuellen Test der Hardware automatisch durch den MenuController verlassen und das Hauptmenü wird angezeigt.

### 4.3.4 MainMenu

Das *MainMenu* ist das erste Menü, welches nach dem Bootscreen angezeigt wird. Zusätzlich zu den Grundfunktionen der *MenuBase*-Klasse besitzt das Hauptmenü die vier *Submenus* *illuminationGreen*, *illuminationRed*, *contrast* und *credits*. Diese kümmern sich um die Einstellung beziehungsweise Darstellung der jeweiligen Parameter und werden bei der Initialisierung des Hauptmenüs durch die geladenen Werte aus dem externen EEPROM auf den zuletzt gespeicherten Wert initialisiert. Der allgemeine Ablauf der Submenüs wird in Kapitel 4.3.6 beschrieben. Durch die `enterMenu(...)` Funktion wird im Hauptmenü eine Übersicht der ersten drei Submenüs mit deren aktuellen Werten angezeigt, was in Abbildung 17 zu erkennen ist. Zusätzlich werden in dieser Funktion die Callbacks für die vier variablen Taster unter dem Display mit den `enter_submenu(...)` Methoden der einzelnen Submenüs überladen (siehe Kapitel 4.2.1). Die aktuelle Funktion dieser vier Taster wird zudem durch die Funktion `reloadButtonContent(...)` im unteren Bereich des Displays und der aktuelle Menü-Name mit Hilfe der Funktion `reloadHeaderContent(...)` im so genannten Menüheader im oberen Bereich dargestellt. Dieser grundsätzliche visuelle Aufbau wird auch in den *ChannelMenus* für die beiden Kanäle verwendet, was jedoch in Kapitel 4.3.5 genauer beschrieben wird. In Abbildung 17 ist die visuelle

Oberfläche des Hauptmenüs im Gesamten zu sehen.

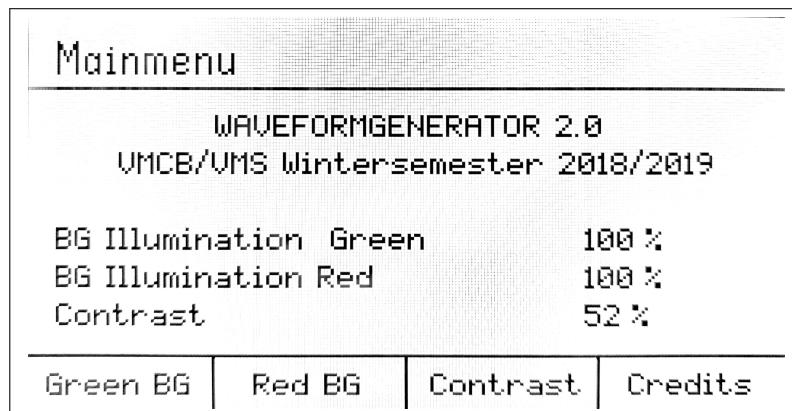


Abb. 17: Allgemeine Übersicht des Hauptmenüs

Das aktuell ausgewählte Submenü wird in dem bereits in der *MenuBase*-Klasse erwähnten *SubMenuBase Pointer* gespeichert. Damit kann bei einem Wechsel vom Hauptmenü zu einem der *ChannelMenus* zunächst das Submenü und anschließend das übergeordnete Menü verlassen werden. Diese Reihenfolge ist vor allem aufgrund des ausstehenden Speicherns des Wertes auf dem externen EEPROM vor Verlassen des Submenüs erforderlich.

Im Hauptmenü besitzen die variablen Taster unter dem Display die Farbe gelb. Die *ChannelSelect* Buttons erhalten, wie in Kapitel 4.3.1 beschrieben, die Farben grün und rot, während die *ChannelActivate* Buttons ihre Farbe nach dem Status der Aktivierung der beiden Kanäle erhalten. Vor dem Verlassen des Hauptmenüs wird wie erläutert zunächst die exit-Funktion des Submenüs aufgerufen und anschließend der Display-Inhalt im angelegten Speicher des Mikrocontrollers gelöscht.

### 4.3.5 ChannelMenu

Die *ChannelMenu*-Klasse ist für die Interaktion mit dem Anwender hinsichtlich der Einstellungen der Ausgangssignal-Parameter zuständig, wobei zwei Instanzen für die beiden Kanäle in der Firmware vorhanden sind. Die Klasse erbt, wie der *Bootscreen* und das *Mainmenu* von der Basisklasse *MenuBase*, wodurch einige Grundfunktionen bereits abgedeckt werden. Neben diesen standardmäßigen Funktionen besitzt dieses Menü zudem die Methoden *isActivated(...)* und *changeActivationTo(...)* für das Abfragen und Ändern der Aktivierung des Ausgangssignals, welche vom *MenuController* gesteuert wird. Außerdem besitzt dieses Menü eine *outputChannel*-Referenz des Typs *SignalGenerator*, welche für die Einstellung der Signalparameter (siehe Kapitel 4.2.8) verantwortlich ist. Darüber hinaus beinhaltet die Klasse

die 6 Submenüs *Frequenz*, *Amplitude*, *Offset*, *Phase*, *Signalform* und *Duty Cycle*, welche die Einstellung der jeweiligen Parameter beinhalten. Die Verwendung und eine ausführliche Beschreibung der Submenüs befindet sich in Kapitel 4.3.6. Wie auch beim Hauptmenü werden die variablen Taster mit den `enterSubMenu(...)` Methoden der jeweiligen Submenüs für die einzelnen Parameter überladen. Auf Grund der Menge von 6 Submenüs und nur 4 variablen Taster ist eine Aufteilung auf zwei Menü-Seiten notwendig, wobei auf der ersten Seite die Menüs Frequenz, Amplitude und Offset erreichbar sind. Der rechte variable Button der ersten Menüseite wird für den Wechsel auf die zweite Seite des Menüs verwendet, auf welcher die Submenüs Phase, Signalform und Duty Cycle aufgerufen werden können. Auf der zweiten Seite wird durch den linken variablen Taster ein Wechsel auf die erste Menüseite ermöglicht. Die Farben der variablen Buttons sind je nach betreffenden Kanal auf grün oder rot geschaltet. Es gilt zu beachten, dass das Submenü *Duty Cycle* lediglich bei ausgewählter Signalform *Rechteck* angezeigt wird, da nur in diesem Fall eine Änderung des Duty Cycles sinnvoll und möglich ist. Der rechte variable Button auf der zweiten Menüseite ist daher bei anderen Signalformen ohne Funktion. Bei der Initialisierung der Channel Menüs werden über die *EEPROM*-Komponente alle gespeicherten Parameter geladen und an die entsprechenden Submenüs übergeben. Somit werden nach einem Neustart des Geräts die Kanäle auf die Parameter der letzten Verwendung eingestellt.

Beim Betreten des *ChannelMenu* über die Funktion `enterMenu(...)` wird eine Übersicht zum Kanal auf dem Display angezeigt. Der Menüheader beinhaltet dabei neben dem Menünamen den aktuellen Aktivierungszustand des Ausgangssignals und die aktuell angezeigte Menüseite. Im mittleren Bereich der Übersicht werden alle Submenüs und deren eingestellte Werte angezeigt. Hierbei wird das *Duty Cycle* Submenü wieder ausgeblendet, falls die aktuell ausgewählte Signalform nicht *Rechteck* ist. Im unteren Bereich wird, wie auch im Hauptmenü, das Ziel des jeweiligen variablen Taster angezeigt. Diese Übersicht des Channel Menüs ist in Abbildung 18 zu sehen.

Mithilfe des `SubMenuBase Pointers` aus der *MenuBase* wird das aktuell betretene Submenü gespeichert. Dies dient dazu, dass vor dem Verlassen des *ChannelMenu* durch den *MenuController* das Submenü verlassen werden kann. Diese Reihenfolge ist wichtig, da die eingestellten Parameter erst nach dem Verlassen eines Submenüs auf dem externen EEPROM gespeichert werden und somit einem Verlust der Daten entgegengewirkt wird. Das Betreten und Verlassen der Menüs wird vom MenuController gesteuert, was in Kapitel 4.3.7 noch genauer beschrieben wird.

Channel 1		Inactive	PAGE: 1 / 2
Frequency		1 Hz	
Amplitude		1.00 V	
Offset		0 V	
Phase		0 °	
Waveform	Sinewave		
		Frequency	Amplitude
		Offset	Page 2

Abb. 18: Allgemeine Übersicht des Menüs für Channel 1

### 4.3.6 Submenüs

Wie bereits erwähnt, besitzen die erstellten Menüs jeweils eine unterschiedliche Anzahl an Submenüs, in welchen die Änderung der einzelnen Parametern gesteuert wird.

#### Basisklasse für die Submenüs

Wie bei den Menüs selbst wurde hierfür eine Basis-Klasse mit dem Namen *SubMenuBase* erstellt, durch welche Grundfunktionen wie `enterSubMenu(...)` und `printSummary(...)` definiert werden. Zusätzlich sind Referenzen auf die verwendeten Komponenten *Display*, *Encoder* und *FourButtonArray* für die variablen Taster unter dem Display in der Basisklasse angelegt. Darüber hinaus besitzt jedes Submenü mit dem Parameter `MenuBase& parent` eine Referenz auf das zugehörige Menü, wodurch der Wechsel vom Submenü zurück zum Menü ermöglicht wird. Der Name des Submenüs und der für die variablen Taster anzuzeigende Text ist ebenfalls in der *SubMenuBase* Klasse gespeichert.

#### Unterschiedliche Sübmenü-Typen

Von dieser Basisklasse werden die unterschiedlichen Submenüs *Value\_submenu*, *Form\_submenu* und *Credits\_submenu* abgeleitet. Letzteres kann vom Hauptmenü aufgerufen werden und zeigt eine Übersicht der Projektmitglieder sowie allgemeine Informationen des Projekts an. Das *Form\_submenu* wird zum Einstellen der aktuellen Signalform der Channel Menüs verwendet. Es besitzt neben den Methoden aus der Basisklasse lediglich einige Funktionen zum Verändern, Anzeigen und Zurückgeben der aktuellen Signalform. Das umfangreichste Submenü wird durch die Klasse *Value\_submenu* modelliert und für alle restlichen Parameter verwendet. Bei der Initialisierung der *Value\_submenus* werden neben dem zuletzt

gespeicherten Wert zudem die minimalen und maximalen Grenzwerte des Parameters übergeben. Darüber hinaus wird die Einheit des Submenüs sowie der relative Faktor, mit welchem der Wert bezogen auf die Einheit abgespeichert ist, definiert. Dabei entspricht beispielsweise die Zahl -3 dem Faktor Milli, 3 dem Faktor Kilo und 0 keinem Faktor.

Während der angezeigte Menüheader unverändert bleibt, werden beim Betreten eines Submenüs die Informationen zum aktuellen Wert, die Einstellungsgrenzen und die aus dem Faktor abgeleitete minimale Schrittgröße angezeigt. Ein Beispiel der Darstellung für das *ValueSubmenu* für die Amplitude ist in Abbildung 19 zu sehen. Beim Parameter Amplitude liegen demnach die Einstellungsgrenzen zwischen 50 mV und 10 V mit einer minimalen Schrittgröße von 1 mV. Ebenfalls in der Grafik zu erkennen sind die im Submenü überladenen variablen Taster-Funktionen „\* 10“ und „: 10“, welche in Verbindung mit der Komponente Encoder stehen, oder der *Back*-Taster, welcher zurück zum Hauptmenü führt.

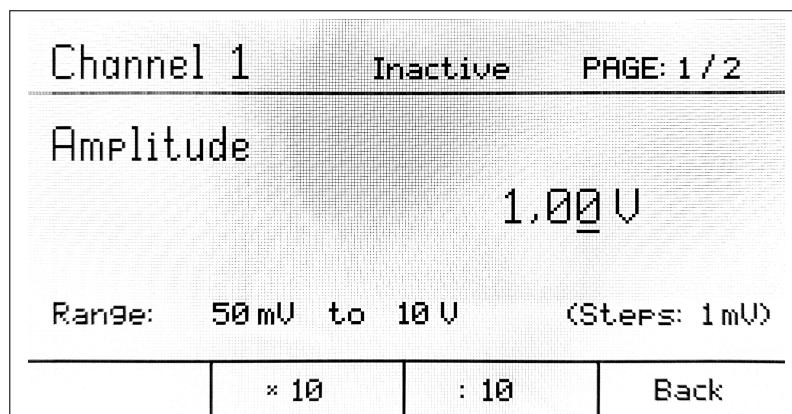


Abb. 19: Submenü mit dem Encoder im *changeDigit*-Modus

## Ändern eines Wertes

Beim Betreten eines *ValueSubmenus* wird der Encoder aktiviert, welcher in den übergeordneten Menüs deaktiviert ist. Mit der Aktivierung wird der Encoder-Komponente der aktuelle Wert des Submenüs als veränderbare Referenz übergeben, sodass der Parameter verändert werden kann. Wie in Kapitel 4.2.2 beschrieben, besitzt der Encoder dabei zwei unterschiedliche Modi. Der Modus *changeDigit* ist beim Aufruf eines Submenüs automatisch ausgewählt, was in Abbildung 19 durch den Strich unter der rechten Ziffer des Amplitudenwertes angezeigt wird. Der Wechsel zum Modus *changeValue* erfolgt durch ein Drücken des Encoders und wird durch die pixelweise Invertierung der ausgewählten Ziffer angezeigt, wie in Abbildung 20 zu sehen ist.

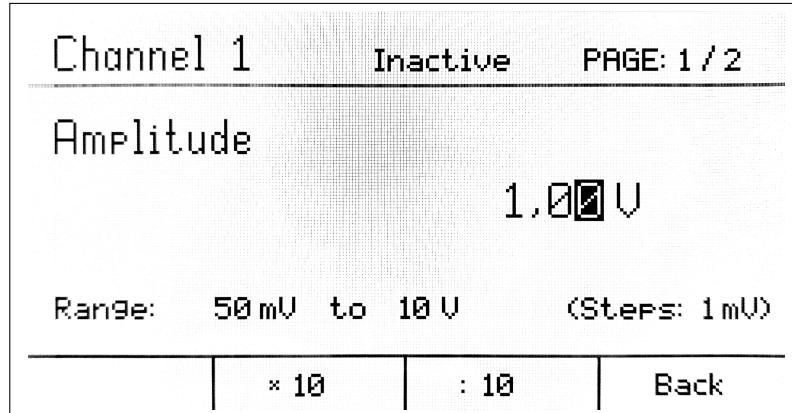


Abb. 20: Submenü mit dem Encoder im *changeValue*-Modus

Für die Handhabung von Links- und Rechtsdrehungen werden der Encoder Komponente zudem zwei Callbacks übergeben. In diesen wird der Wert des Parameters verändert und mithilfe der entsprechenden Funktion der *SignalGenerator* Komponente am Signalausgang eingestellt. Die beiden mittleren Display-Buttons erlauben über die Encoder Komponente das Ändern des Wertes um den Faktor 10, was ebenfalls direkt an den entsprechenden *SignalGenerator* weitergegeben wird. Somit wird es ermöglicht, dass die in der Benutzeroberfläche veränderten Werte direkt am Ausgang gemessen werden können. Durch Drücken des : 10-Tasters wird z.B. der Wert der Amplitude von 1,00 V aus Abbildung 20 auf 100 mV verringert, was Abbildung 21 zeigt. Während das Ändern der Parameter direkt an den *SignalGenerator* gesendet und dort verarbeitet wird, ist das Speichern des Parameter auf dem externen EEPROM auf den Zeitpunkt des Verlassens des Submenüs beschränkt. Neben dem Speichern der Werte wird außerdem der Encoder beim Verlassen des Submenüs wieder deaktiviert und der mittlere Bereich des Displays, welcher für die Anzeige der Submenü-Informationen verwendet wird, mit dem neuen Wert aktualisiert.

#### 4.3.7 MenuController

Der *MenuController* behandelt das Wechseln der Zustände innerhalb der State Machine, welche den einzelnen Menüs der Benutzeroberfläche entsprechen. In der Klasse existiert ein Array aus Pointern auf Objekte der Klasse *MenuBase* (siehe Kapitel 4.3.2), in welchem die einzelnen Zustände des Controllers gespeichert sind. Dieses Array wird über die Funktion `addMenu(...)` erweitert und während der Initialisierungsphase mit den Menüs *Bootscreen*, *Mainmenu*, *Channel 1 Menu* und *Channel 2 Menu* gefüllt.

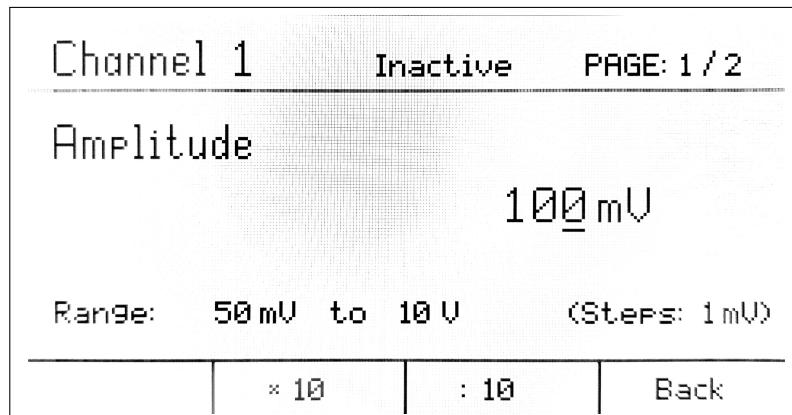


Abb. 21: Veränderter Wert nach Drücken des :10-Buttons

Die beiden *ChannelSelect-Buttons*, modelliert in einer Instanz des *FourButtonArray*, werden wie in Kapitel 4.3.1 erwähnt vom *MenuController* zur Steuerung der State Machine verwendet. Der linke Taster ist hierbei Kanal 1 zugeordnet und aktiviert durch Drücken das entsprechende Channel Menü. Der rechte Taster weist dieselbe Funktionalität für den zweiten Kanal auf. Befindet sich die State Machine in einem der beiden Channel Menüs und der dazugehörige ChannelSelect-Button wird erneut gedrückt, so wechselt die Benutzeroberfläche wieder in das Hauptmenü.

Für einen Zustandswechsel ruft der *MenuController* zunächst die Methode `exitMenu(...)` des zu verlassenden Menüs auf und anschließend die Funktion `enterMenu(...)` des neuen Menüs.

Durch die Exit-Funktion wird sichergestellt, dass das aktuelle Submenü des Menüs (siehe Kapitel 4.3.6) verlassen, ein noch nicht gespeicherter Wert auf dem externen EEPROM abgelegt, der Displayinhalt gelöscht und der Encoder deaktiviert wird. Beim Betreten des neuen Menüs bewirkt die Enter-Funktion, dass die variablen Taster mit den entsprechenden `enterSubmenu(...)` Funktionen für die Einstellung der Parameter eines Menüs überladen werden. Zusätzlich wird das Display mit den neuen Inhalten beschrieben.

Eine weitere Funktion des *MenuController* ist die Aktivierung und Deaktivierung der Ausgangssignale. Dies wird durch die beiden *ChannelActivate-Buttons* realisiert, welche innerhalb der gleichen Instanz des *FourButtonArray*s wie die *ChannelSelect-Buttons* angesteuert werden. Dabei ist der linke Button wieder Kanal 1 und der rechte Button Kanal 2 zugeordnet. Die Beleuchtung der beiden Buttons beschränkt sich auf grün bzw. rot für ein aktiviertes Signal am jeweiligen Kanal oder aus für ein deaktiviertes.

## 5 Fazit

Nach Fertigstellung von Hardware und Software wurden sämtliche Komponenten innerhalb des Gehäuses verbaut. Das fertige Gerät ist in Abbildung 22 dargestellt.



Abb. 22: Der fertige Funktionsgenerator

Der fertige Funktionsgenerator entspricht weitestgehend den 3D Modellen, die im Vorfeld erstellt wurden.

In Tabelle 1 innerhalb von Kapitel 2.1 wurden die Parameter der analogen Signalerzeugung definiert. Die tatsächlich erzeugten Signale werden mithilfe mehrerer Aufnahmen im Folgenden genauer betrachtet, wobei jeweils auf unterschiedliche Parameter des Ausgangssignals eingegangen wird.

### Ausgangsfrequenz

Der Bereich der Ausgangsfrequenz wurde während der Konzipierung von Version 2 des Funktionsgenerators auf das Intervall [1 Hz, 1 MHz] festgelegt (siehe Kapitel 2.1), wobei die obere Grenze die Mindestanforderung darstellt. Nachfolgend sind zwei Signalverläufe mit diesen beiden Grenzfrequenzen dargestellt. Hierbei wurden

Rechtecksignale für die Visualisierung gewählt, da diese Signalform mit den steilen Flanken das Potential des analogen Schaltungsteils am besten wiedergibt.

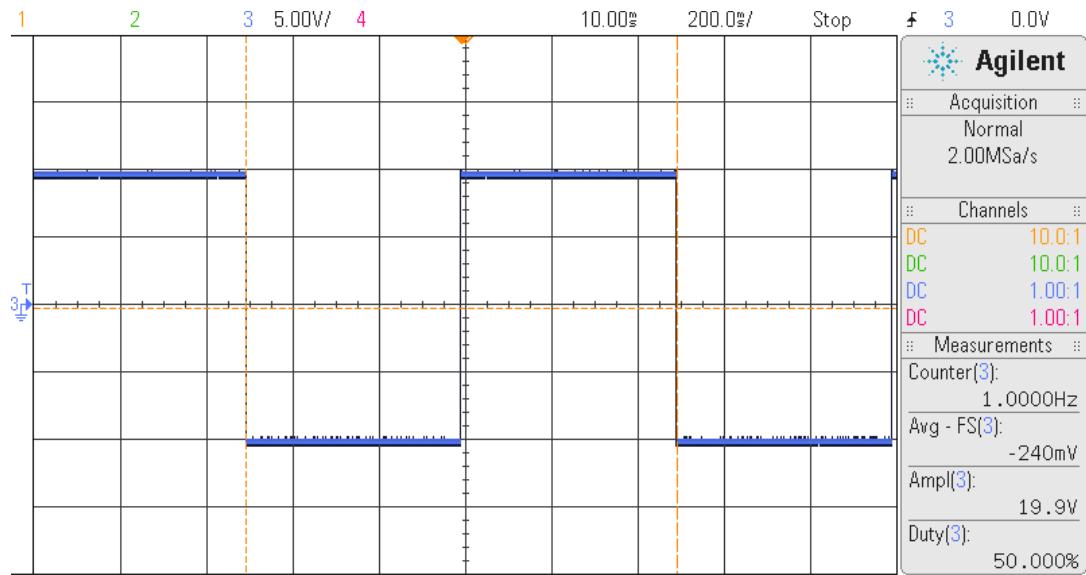


Abb. 23: Rechtecksignal mit Frequenz  $f = 1$  Hz

Abbildung 23 zeigt ein Signal mit der unteren Grenzfrequenz  $f = 1$  Hz. Zudem ist die Amplitude auf den Maximalwert von 10V und der Offset auf 0V eingestellt. Der Duty-Cycle beträgt 50 %, was dem Standardwert entspricht. Aufgrund einer nicht vollständig durchgeführten Kalibrierung des Digital-Analog-Wandlers entsprechen Amplitude und Offset nicht ganz den Sollwerten.

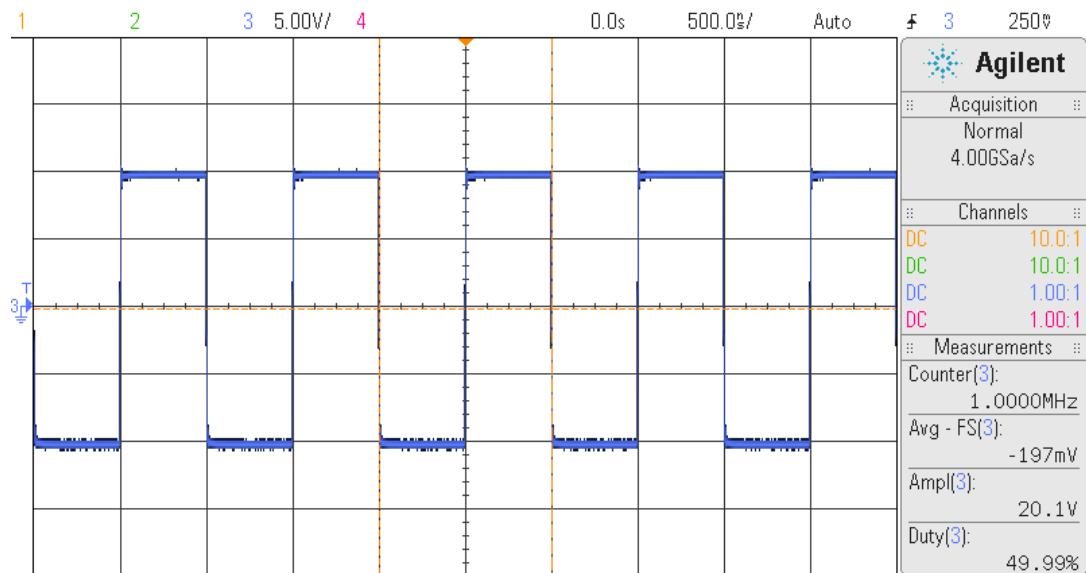


Abb. 24: Rechtecksignal mit Frequenz  $f = 1$  MHz

In Abbildung 24 ist ein Rechteck-förmiges Signal mit der ursprünglich als oberen Grenze festgelegten Frequenz  $f = 1 \text{ MHz}$  zu sehen. Die Parameter Amplitude und Offset sind wieder nicht optimal, was auf die Kalibrierung zurückzuführen ist. Der minimale Fehler des Duty-Cycles kann hingegen mit Messungenauigkeiten des Oszilloskops begründet werden. Eine Messung der Anstiegs- und Abfallzeiten des Signals mithilfe des Oszilloskops ergab Werte im Bereich um 6 ns. Folglich sind die Flanken des Signals auch bei dieser Frequenz noch sehr gut, weshalb die obere Grenzfrequenz nach oben gesetzt wurde.

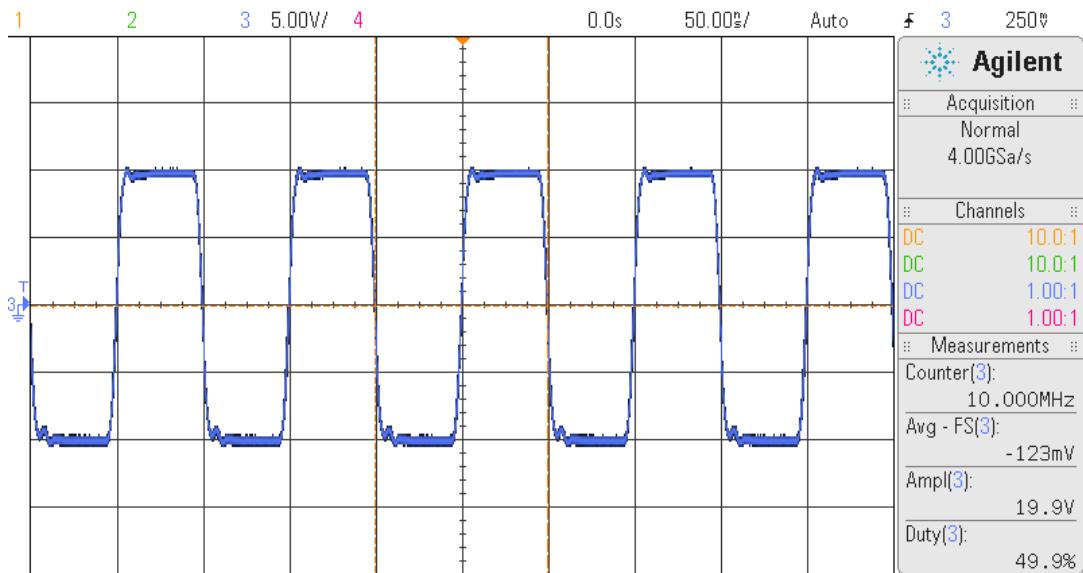


Abb. 25: Rechtecksignal mit Frequenz  $f = 10 \text{ MHz}$

Aufgrund der noch sehr guten Signaleigenschaften bei einer Frequenz von 1 MHz wurde die obere Grenzfrequenz auf 10 MHz angehoben. In Abbildung 25 ist ein Rechtecksignal mit dieser Frequenz dargestellt. Man kann deutlich erkennen, dass die Flankensteilheit wesentlich schlechter ist als noch bei 1 MHz, was auf die Anstiegs- und Abfallzeiten zurückzuführen ist. Diese liegen bei 10 MHz bereits in der selben Größenordnung wie die Periodendauer des Signals. Dennoch sind die Eigenschaften des Signals noch akzeptabel, weshalb diese neue obere Grenzfrequenz beibehalten wurde.

### Signalamplitude

Der Ausgangsspannungsbereich des Funktionsgenerators beträgt maximal  $\pm 10 \text{ V}$ . Die Amplitude des Signals kann daher beliebig im Intervall  $[100 \text{ mV}, 10 \text{ V}]$  eingestellt werden. In den bisherigen Abbildungen, die Ausgangssignale zeigen, war die Amplitude auf den maximalen Wert eingestellt, um die Slew-Rates des analogen

Schaltungsteils größtmöglich auszureißen. In Abbildung 26 ist ein Signal mit einer Amplitude von  $\hat{U} = 1 \text{ V}$  zu sehen. Die Amplitudenmessung des Oszilloskops betrachtet die Peak-Peak Spannung des Signals, weshalb hier das doppelte der eingestellten Amplitude angezeigt wird. Die minimale Abweichung der Amplitude und des Offsets ist wieder auf Ungenauigkeiten der Kalibrierung zurückzuführen.

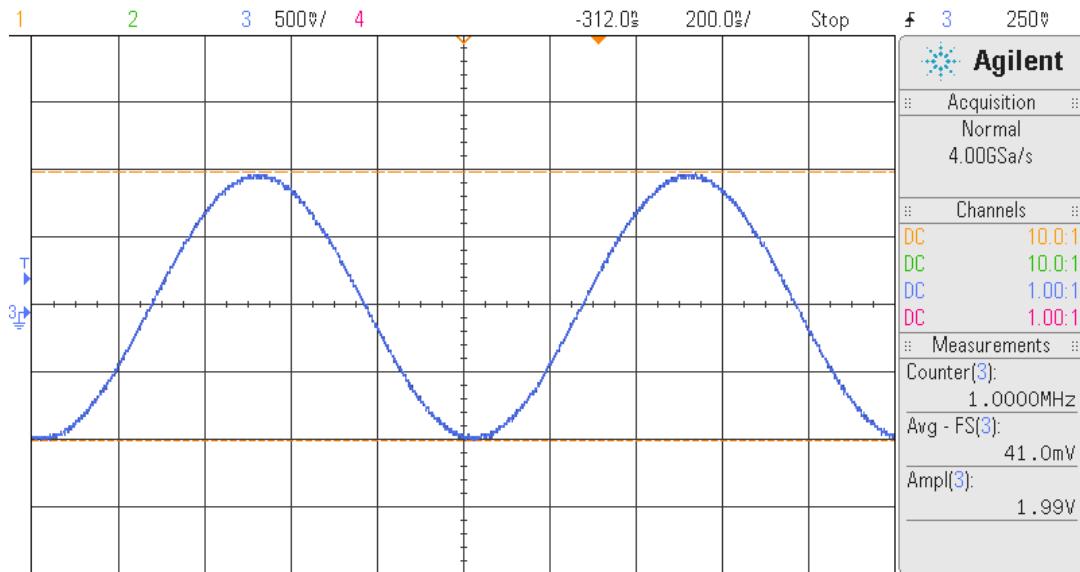


Abb. 26: Sinus-Signal mit Amplitude  $\hat{U} = 1 \text{ V}$

### Signaloffset

Neben der Frequenz und der Amplitude kann zudem der Offset beziehungsweise Gleichanteil des Signals verändert werden. In Abbildung 27 sind die Signale der beiden Ausgangskanäle zu sehen, wobei jeweils Frequenz und Amplitude identisch sind. Bei Kanal 1 (blaues Signal) ist zusätzlich noch ein Offset von  $\bar{U} = 1 \text{ V}$  eingestellt. Die Abweichungen der Werte können mit der nicht vollständig durchgeföhrten Kalibrierung begründet werden.

### Phasenversatz zwischen beiden Ausgangskanälen

Die Synchronisierung der beiden Ausgangskanäle ermöglicht es, einen Phasenversatz zwischen den beiden Signalen zu generieren. Dieser kann im Intervall  $[-180^\circ, 180^\circ]$  eingestellt werden. In Abbildung 28 ist exemplarisch ein Versatz von  $\varphi = 45^\circ$  dargestellt.

### Einstellen des Duty-Cycles von Rechtecksignalen

Ein Parameter, der lediglich bei Rechtecksignalen Relevanz besitzt, ist der Duty-Cycle, also der prozentuale Anteil, in dem das Signal den High-Pegel annimmt.

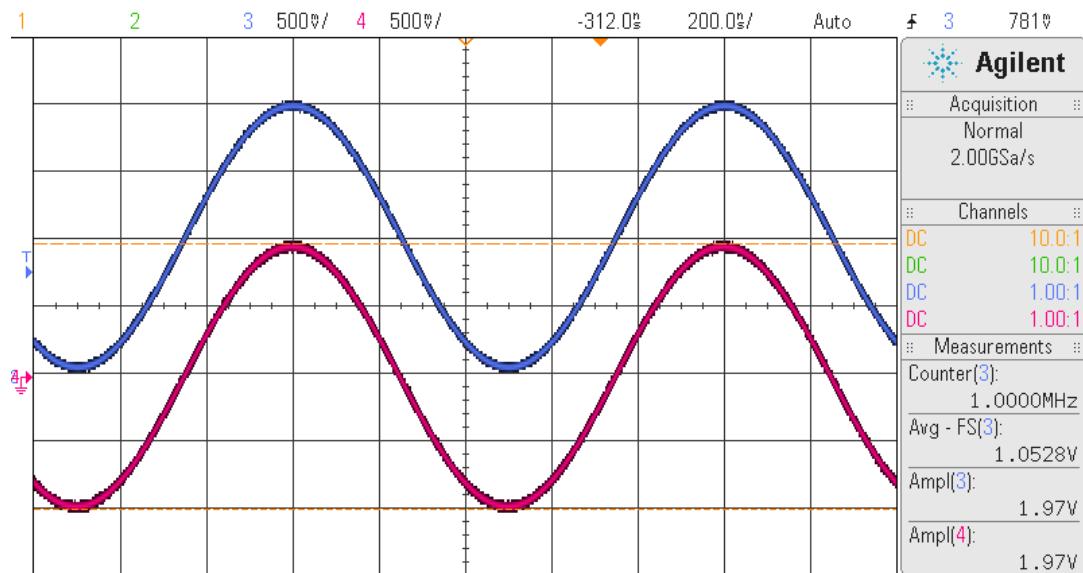
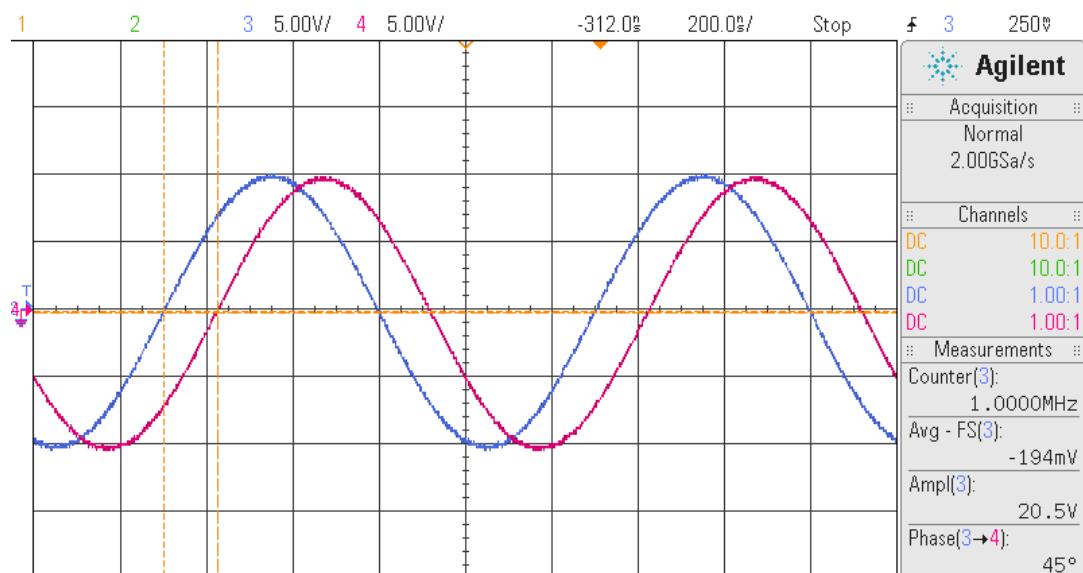
Abb. 27: Sinus-Signal mit Offset  $\bar{U} = 1 \text{ V}$ 

Abb. 28: 45° Phasenversatz zwischen beiden Ausgangskanälen

Dieser ist im Intervall [0 %, 100 %] einstellbar, wobei 0 % und 100 % einer konstanten Spannung mit Amplitude des Low- beziehungsweise High-Pegel entsprechen. Abbildung 29 zeigt als Beispiel ein Rechtecksignal mit einem Duty-Cycle von 10 %.

### Unterbrechung bei Änderung der Signalparameter

Die Synchronisation der beiden Ausgangskanäle erfolgt über ein Logik-Signal, welches gleichzeitig an beiden Kanälen anliegt und mittels fallender Flanke die Signalerzeugung startet. Da die Parameter der Ausgangssignale nur verändert

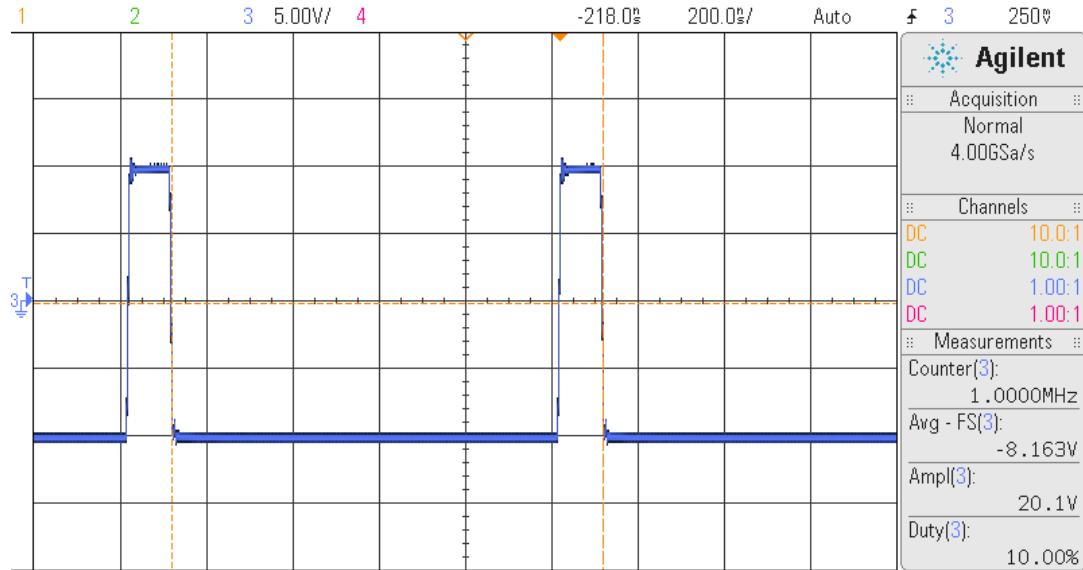


Abb. 29: Rechtecksignal mit 10 % Duty-Cycle

werden können, wenn kein Signal generiert wird, muss bei jeder Änderung eines Parameters die Signalerzeugung beider Kanäle kurz unterbrochen werden. Mithilfe eines Oszilloskops wurde diese Unterbrechung aufgezeichnet, was in Abbildung 30 visualisiert ist.

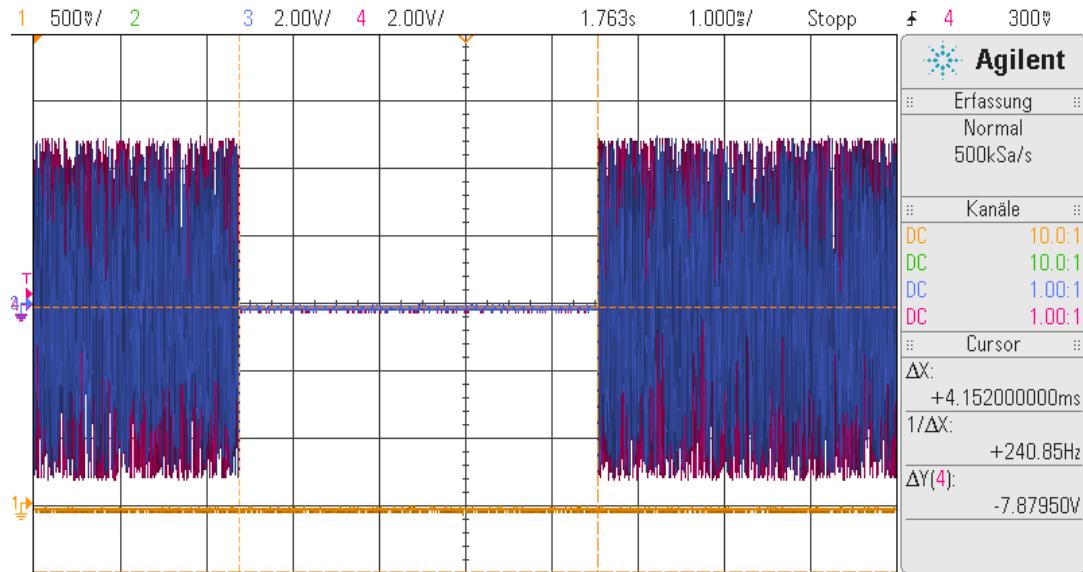


Abb. 30: Signalunterbrechung bei Parameterumschaltung

Durch verschiedene Software-Optimierungen konnte diese Unterbrechung auf wenige Millisekunden reduziert werden (etwa 4 ms in Abbildung 30). Diese Zeit ist kurz genug, damit der Anwender die Unterbrechung nicht mitbekommt.

Insgesamt betrachtet erfüllt der Funktionsgenerator in Version 2 alle definierten Anforderungen hinsichtlich der generierten Ausgangssignale oder übertrifft diese sogar. Zudem wurde viel Wert auf die Benutzerinteraktion und das User Interface gelegt, wodurch die Bedienung des Geräts sehr einfach und angenehm wird. Vor allem das große Display und die farbliche Gestaltung der Taster haben diesbezüglich viele Möglichkeiten eröffnet. Dennoch ist die Menüführung der Benutzeroberfläche sehr umfangreich und vielschichtig, weshalb neben der Projektdokumentation noch eine eigenständige Bedienungsanleitung entstanden ist, die sich im Anhang befindet.

## Literatur

- [1] MANUEL DENTGEN, TOBIAS FRAUENSCHLÄGER, JOHANNES MAIER, THOMAS TAUGENBECK: *Funktionsgenerator (mit DDS AD9106)*. [https://hps.hs-regensburg.de/~eiwiki/eiwiki/index.php?title=Funktionsgenerator\\_\(mit\\_DDS\\_AD9106\)](https://hps.hs-regensburg.de/~eiwiki/eiwiki/index.php?title=Funktionsgenerator_(mit_DDS_AD9106)). Version: 2018
- [2] FRAUENSCHLÄGER, Tobias: *Praktikumsbericht*. Bericht, 2018
- [3] TEKO: *AUS-33.pdf*. <http://www.teko.it/uploads/manuali/pdf/AUS-33.pdf>. Version: 2017
- [4] MEAN WELL: *100W Single Output Switching Power Supply*. <https://www.meanwell.com/Upload/PDF/LRS-100/LRS-100-SPEC.PDF>. Version: 2017
- [5] ANALOG DEVICES: *Dual Multitopology DC/DC Converters with 2A Switches and Synchronization*. <https://www.analog.com/media/en/technical-documentation/data-sheets/8471fd.pdf>. Version: 2014
- [6] TEXAS INSTRUMENTS: *TPS7A45xx Low-Noise Fast-Transient-Response 1.5-A Low-Dropout Voltage Regulators*. <http://www.ti.com/lit/ds/symlink/tps7a45.pdf>. Version: 2008
- [7] NXP SEMICONDUCTORS: *16-bit I<sup>2</sup>C-bus and SMBus I/O port with interrupt*. <https://www.nxp.com/docs/en/data-sheet/PCA9555.pdf>? Version: 2017
- [8] STMICROELECTRONICS: *STM32L476RG - Ultra-low-power with FPU ARM Cortex-M4 MCU*. Datenblatt. <https://www.st.com/resource/en/datasheet/stm32l476rg.pdf>, Abruf: 02. März 2019
- [9] WIKIPEDIA: *Direct Digital Synthesis*. [https://de.wikipedia.org/wiki/Direct\\_Digital\\_Synthesis](https://de.wikipedia.org/wiki/Direct_Digital_Synthesis)
- [10] ANALOG DEVICES: *AD9102 - Low Power, 14-Bit, 180 MSPS, Digital-to-Analog Converter and Waveform Generator*. Datenblatt. <https://www.analog.com/media/en/technical-documentation/data-sheets/ad9102.pdf>, Abruf: 02. März 2019
- [11] TEXAS INSTRUMENTS: *DAC8734 - Quad, 16-Bit, High-Accuracy, ±16V Output, Serial Input Digital-to-Analog Converter*. Datenblatt. <http://www.ti.com/lit/ds/symlink/dac8734.pdf>, Abruf: 04. März 2019

- [12] SILICON LABS: *Si5351 - I2C-Programmable Any-Frequency CMOS Clock Generator.* Datenblatt. <https://www.silabs.com/documents/public/datasheets/Si5351-B.pdf>, Abruf: 04. März 2019
- [13] WIKIPEDIA: *Phasenregelschleife.* <https://de.wikipedia.org/wiki/Phasenregelschleife>
- [14] WIKIPEDIA: *Very High Speed Integrated Circuit Hardware Description Language.* [https://de.wikipedia.org/wiki/Very\\_High\\_Speed\\_Integrated\\_Circuit\\_Hardware\\_Description\\_Language](https://de.wikipedia.org/wiki/Very_High_Speed_Integrated_Circuit_Hardware_Description_Language)
- [15] WIKIPEDIA: *Complex Programmable Logic Device.* [https://de.wikipedia.org/wiki/Complex\\_Programmable\\_Logic\\_Device](https://de.wikipedia.org/wiki/Complex_Programmable_Logic_Device)
- [16] ROBENKO, Alex: *Practical Guide to Bare Metal C++.* E-Book. [https://legacy.gitbook.com/book/arobenko/bare\\_metal\\_cpp/details](https://legacy.gitbook.com/book/arobenko/bare_metal_cpp/details), Abruf: 104. März 2019