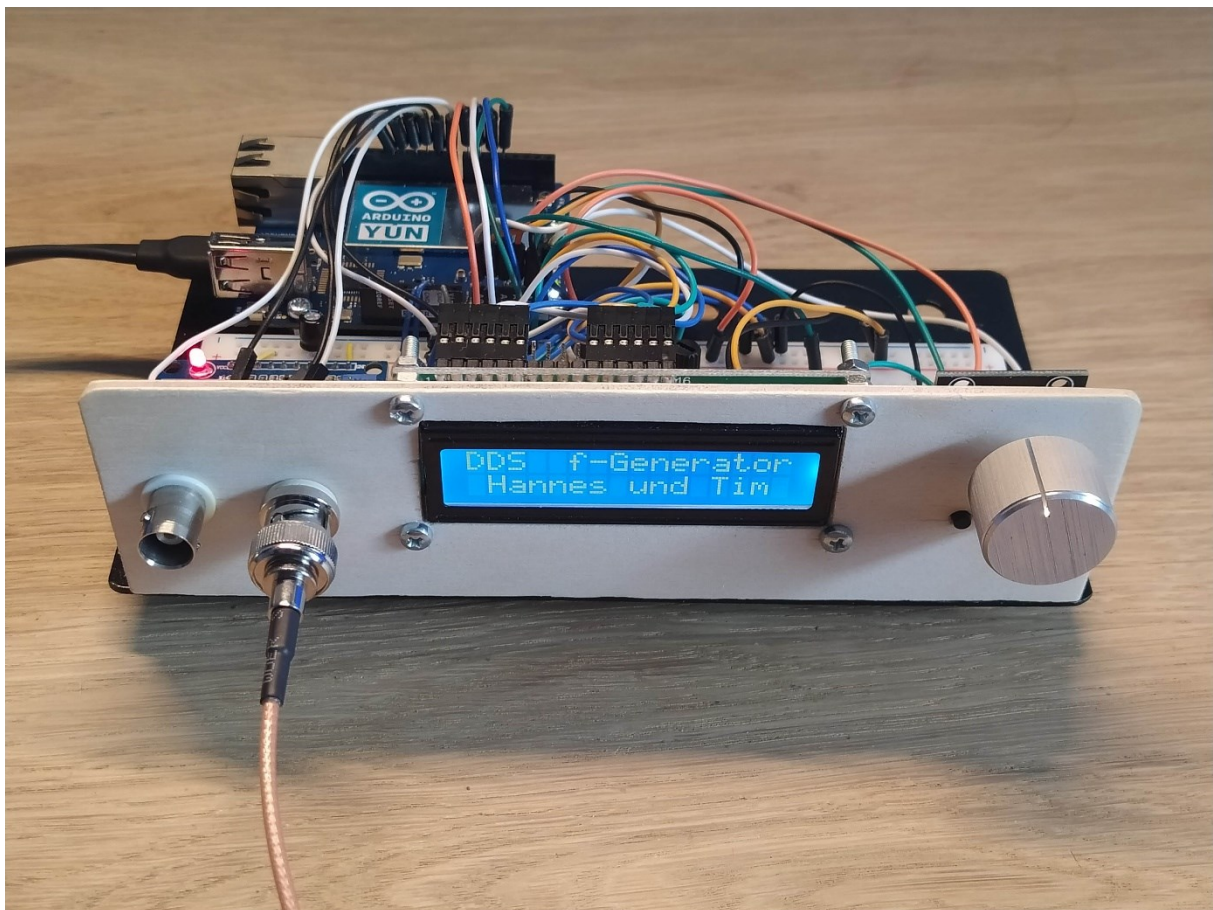


Programmierung eines DDS-Signalgenerators



Johannes Krause 15445095

Tim Fischer 15449007

Inhalt

Einleitung (Tim)	1
Aufgabenstellung (Hannes)	2
Physikalische Grundlagen (Hannes)	3
Was ist ein DDS-Generator?	3
Frequenzlauf.....	6
Linearer Sweep:	6
Logarithmischer Sweep:	7
Verwendete Hardware (Tim).....	8
Arduino Yún.....	8
ADS9850 (HC-SR08)	9
LC-Display mit HD44780-Controller	10
Drehinkrementalgeber	11
Der Aufbau	12
Das Arduino Programm (Hannes).....	13
Menüstruktur	13
Programmablauf.....	14
Ausgewählte Funktionen.....	15
Einstellen eines beliebigen Wertes – setHighValue	15
Logarithmischer Frequenzlauf – logSweep	17
Kommunikation mit der remote Steuerung (Tim).....	19
Programm zur remote Steuerung (Tim)	21
Die Klassen.....	21
App	21
Main.....	21
MainController	22
Connection	24
Alertbox	24
Messung und Fehleranalyse (Tim)	25
Erweiterungs- und Verbesserungsmöglichkeiten(zusammen)	26
Fazit (zusammen)	27

Anhang	I
Messung der Frequenzen und Amplituden bei verschiedenen Frequenzen	I
Fast-Fourier-Transformation (FFT) von ausgewählten Signalen	IV
Zusatzdokumentation Arduino.....	V
Zusatzdokumentation remote Programm.....	VI
Abbildungsverzeichnis.....	VII
Literaturverzeichnis.....	VIII

Einleitung (Tim)

Im Rahmen des Vertiefungspraktikums „Softwareentwicklung für Physical Computing Plattformen“ soll ein DDS Frequenzgenerator mit definiertem Funktionsumfang auf einem Atmel Mikrocontroller in der Programmiersprache C++ in der Arduino IDE¹ programmiert werden.

Zusätzlich wurde sich dazu entschieden, die Aufgabe um ein Programm zur Fernsteuerung des Funktionsumfang des DDS von einem Rechner aus, zu erweitern. Dieses Programm soll eine graphische Oberfläche besitzen und in der Programmiersprache Java in der IDE „IntelliJ“ der Firma „JetBrains“ erstellt werden.

Durch das Projekt soll das erarbeitete Wissen aus den Veranstaltungen Programmiersprachen 1 und Programmiersprachen 2 vertieft werden.

¹ IDE = Integrated development environment (Integrierte Entwicklungsumgebung)

Aufgabenstellung (Hannes)

Das grundlegende Ziel dieser Projektarbeit ist der Entwurf sowie die softwareseitige Umsetzung eines DDS-Signalgenerators mit mehreren, separat anwählbaren Funktionen. Neben der Ausgabe einer in 10Hz-Schritten im Bereich von 10Hz bis 40MHz einstellbaren Festfrequenz soll der Generator in der Lage sein, einen Linearen Frequenzlauf zwischen zwei frei wählbaren Frequenzen bzw. einen Logarithmischen Frequenzlauf von einer wählbaren Startfrequenz über eine definierte Anzahl an Dekaden auszuführen. Die Dauer sowie Anzahl der einzelnen Schritte des jeweiligen Sweeps sollen ebenfalls durch den Bediener angepasst werden können. Das Einstellen aller genannten Parameter erfolgt über ein Menü mithilfe eines Drehinkrementalgebers direkt am Gerät, sodass der Signalgenerator als Stand-Alone-Device unabhängig von anderen Geräten arbeiten kann.

Diese Aufgabenstellung wird um die Möglichkeit der komfortablen Einstellung aller Parameter und Starten der Funktionen am PC mithilfe einer Java-Anwendung erweitert, der notwendige Datenverkehr zwischen PC und DDS-Generator soll hierbei kabellos im TCP-Protokoll über die WLAN-Schnittstelle des verwendeten Arduino Yún erfolgen.

Physikalische Grundlagen (Hannes)

Um die Funktionsweise sowie Bedeutung eines DDS-Signalgenerators für die moderne Elektrotechnik nachvollziehen zu können, sollen in den folgenden Abschnitten die notwendigen Grundlagen kurz erläutert werden.

Was ist ein DDS-Generator?

Ein DDS-Generator beruht auf dem Prinzip der *Direct Digital Synthesis* (direkte digitale Synthese/DDS). Hierbei handelt es sich um ein Verfahren zur Erzeugung analoger Wellenformen, indem zunächst ein digitales zeitveränderliches Signal erzeugt wird, welches dann mithilfe eines Digital-Analogwandlers in das analoge Ausgangssignal umgewandelt wird. In den meisten Fällen kommen sogenannte *complete DDS solutions* zum Einsatz, welche den eigentlichen DDS und den benötigten D/A-Wandler auf einem gemeinsamen Chip integrieren. [1] [2] [3] [4]

Zur Erzeugung einfacher sinusförmiger Signale änderbarer Frequenz benötigt ein solcher Baustein lediglich einen externen Referenztakt sowie ein variables Steuerwort M (*Tuning Word*), welches die eigentliche Frequenz bestimmt. Der schematische Aufbau eines Complete-DDS-Devices ist in Abbildung 1 dargestellt. Der binäre Wert des *Tuning Words* wird dem Phasenakkumulator (*Phase Accumulator*) übergeben, anschließend wird der dem Wert des Phasenakkumulators entsprechenden Phase ein digitaler Signalwert zugeordnet (*Phase-to-amplitude Converter*), welcher mittels des D/A-Umsetzers letztlich in das Ausgangssignal in Form einer analogen Spannung gewandelt wird.

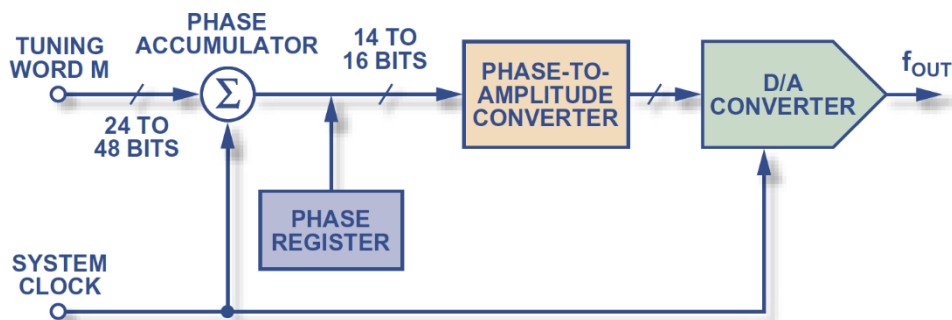


Abbildung 1: Prinzipieller Aufbau eines DDS Systems [1]

Der Phasenakkumulator addiert dabei mit jedem Taktzyklus den Wert des Steuerwortes M , welcher proportional zur generierten Frequenz ist, auf seinen aktuell gespeicherten Wert. Ist der Speicher des Phasenakkumulators (meist 24-48 Bit) vollständig gefüllt, findet ein Überlauf statt und der Wert wird auf Null zurückgesetzt. Der zeitliche Verlauf dieses Wertes hat somit die Form eines Sägezahns und kann -normiert mit 2π - als ein diskretisiertes Maß für den Phasenwinkel einer Sinusfunktion betrachtet werden. Der Akkumulator lässt sich veranschaulichend als Kreis aus 2^n äquidistant verteilten Punkten darstellen, wobei n der Länge des Akkumulators in Bit entspricht (Abbildung 2). Eine Akkumulator-Länge von beispielsweise 8 Bit entspräche also einem Kreis mit $2^8 = 256$ Punkten. Der aktuelle Wert des Akkumulators lässt sich als umlaufender Zeiger interpretieren, welcher mit jedem Taktzyklus um M Punkte weiterspringt. Je nachdem wie groß das Verhältnis vom Steuerwort zur Länge des Akkumulators ist, benötigt der Zeiger mehr oder weniger Taktzyklen, um mindestens eine volle Umdrehung auszuführen, er rotiert also mit einer niedrigeren oder höheren Frequenz.

Die Frequenz des Ausgangssignals der DDS (f_{out}) ist somit berechenbar in Abhängigkeit des Steuerworts M , der Länge des Phasenakkumulators in Bit (n) sowie der Frequenz des Referenztaktes f_c :

$$f_{out} = \frac{M * f_c}{2^n}$$

Nach dem Nyquist-Shannon-Theorem kann die theoretisch mögliche Frequenz maximal $f_c/2$ betragen, sodass das Ausgangssignal wieder vollständig rekonstruiert werden kann. In der Praxis ist die maximale Ausgangsfrequenz jedoch auf einen deutlich niedrigeren Wert begrenzt, um die Qualität des Ausgangssignals zu verbessern und eine weitere Verarbeitung des Signals, z.B. Filterung, zu ermöglichen.

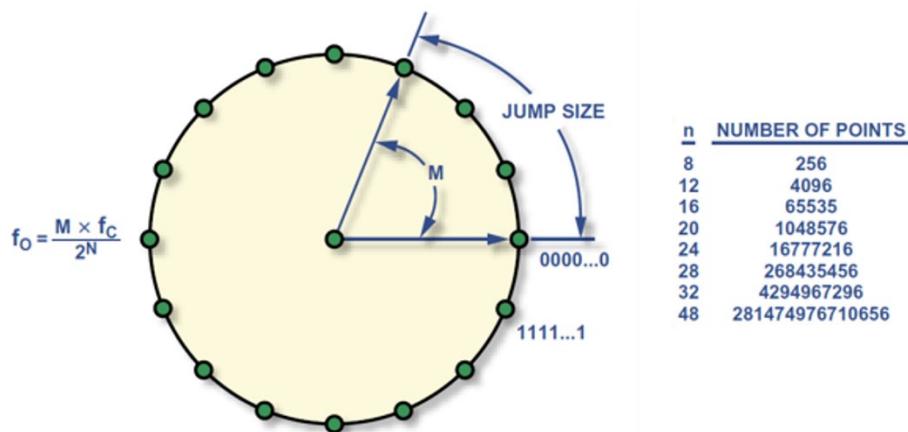


Abbildung 2: Modell des Phasenakkumulators als umlaufender Phasenzeiger [1]

Den normierten Phasenwerten können zur Erzeugung des sinusförmigen Ausgangssignals nun entsprechende Funktionswerte zugeordnet werden. Hierzu wird eine im Flashspeicher des DDS-Moduls abgelegte Lookup-Table verwendet, wobei den verschiedenen Phasenwinkeln entsprechende Werte für die Amplitude der Ausgangsfunktion hinterlegt sind. Der D/A-Wandler erzeugt aus diesen diskreten Funktionswerten das endgültige, analoge und somit quasikontinuierliche Ausgangssignal (Abbildung 3). Alternativ zur Lookup-Table kommen Algorithmen wie z.B. der CORDIC-Algorithmus zum Einsatz, welche zur Laufzeit die Ausgabewerte für einen sinusförmigen Signalverlauf berechnen.

Aufgrund des einfachen funktionellen Aufbaus können Anwendungen auf DDS-Basis sehr klein und kostengünstig gefertigt werden. Da ein DDS vollständig digital programmierbar ist, werden keine externen Bauteile benötigt, um Einstellungen am ausgegebenen Signal vornehmen zu können.

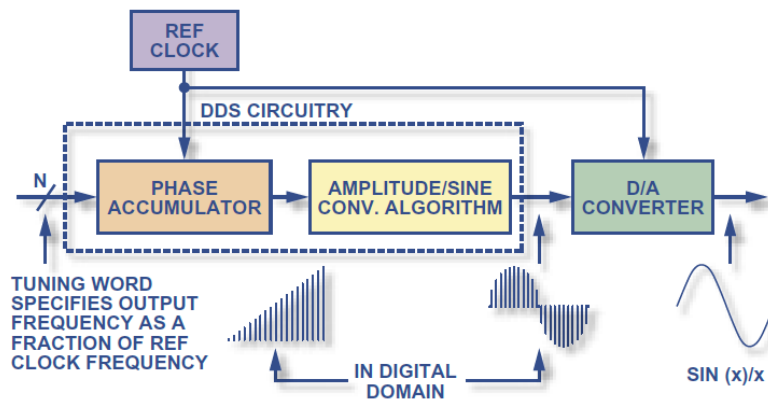


Abbildung 3: Signalfluss bei der DDS [1]

Nichtsdestotrotz kann die Frequenz mit einer Auflösung im Microhertz-Bereich sehr genau justiert werden und Frequenzänderungen erfolgen extrem schnell und exakt ohne Überschwinger (der AD9850 ermöglicht bis zu 23 Millionen Frequenzänderungen pro Sekunde), sodass Einsätze in Echtzeitumgebung möglich sind. Darüber hinaus ermöglicht die im Vergleich zu aus diskreten Bauteilen aufgebauten Frequenzgeneratoren simple Hardware eine kostenoptimierte Herstellung (wenn nicht gerade pandemiebedingt extreme Halbleiterknappheit herrscht...). Das Ausgangssignal von DDS-Generatoren umfasst ein weites Frequenzspektrum von $<1\text{Hz}$ bis in den dreistelligen Megahertz-Bereich, außerdem ist dank des Einsatzes von CMOS-Technik der Leistungsbedarf von DDS sehr gering ($<380\text{mW}$ beim AD9850). Wird ein hochwertiger Referenztakt mit exakter Frequenz sowie ein qualitativ hochwertiger D/A-Wandler verwendet, weist das Spektrum des Ausgangssignals exzellente Rauscheigenschaften wie ein sehr geringes Phasenrauschen, eine geringe Störleistung und geringen Jitter auf.

Diese Vorteile sorgen dafür, dass die DDS eine bedeutende Rolle in der Kommunikationstechnik, z.B. als agile (d.h. sofort reagierende) Frequenzquelle für verschiedenste Anwendungen der Modulation, als justierbare Referenz für Phasenregelschleifen oder zur direkten Hochfrequenz-Übertragung.

Eine weiterer wichtiger Anwendungsfall ist der DDS-basierte, programmierbare Frequenzgenerator, wozu auch der in diesem Praktikum vorgestellte Aufbau zählt, und welche in der Messtechnik z.B. zur Bestimmung von Resonanzfrequenzen zum Einsatz kommen.

Frequenzlauf

Unter einem Frequenzlauf, auch Sweep oder Chirp genannt, versteht man ein Wechsignal konstanter Amplitude, dessen Frequenz sich zeitlich periodisch zwischen einem Start- und einem Endwert ändert. [5] Diese Änderung kann in unterschiedlicher Form geschehen:

Linearer Sweep:

Die Frequenz ändert sich ab der Startfrequenz f_0 bis zum Erreichen der Endfrequenz f_1 mit konstanter Geschwindigkeit in der Form

$$f(t) = ct + f_0$$

Die Variable c beschreibt die Frequenzänderung und lässt sich als Differenzenquotient von End- und Startfrequenz sowie der gewünschten Sweepzeit T beschreiben:

$$c = \frac{f_1 - f_0}{T}$$

Um von der Frequenz einer Wechselgröße zur zugehörigen Zeitfunktion der Phase zu gelangen, wird die Funktion der Frequenz integriert:

$$\varphi(t) = \varphi_0 + 2\pi \int_0^t f(\tau) d\tau$$

$$\varphi(t) = \varphi_0 + 2\pi \int_0^t c\tau + f_0 d\tau$$

$$\varphi(t) = \varphi_0 + 2\pi \left(\frac{c}{2} * t^2 + f_0 t \right)$$

Für die Zeitfunktion des gesweepeten Signals mit linearer Frequenzänderung ergibt sich somit:

$$x(t) = \sin[\varphi(t)] = \sin \left[\varphi_0 + 2\pi \left(\frac{c}{2} * t^2 + f_0 t \right) \right]$$

Dieser Verlauf ist in Abbildung 4 mithilfe eines Matlab-Plots beispielhaft dargestellt. Das zugehörige Amplitudenspektrum (ohne Störgrößen) bestünde entsprechend zu gleichen Anteilen aus allen durchlaufenen Frequenzen.

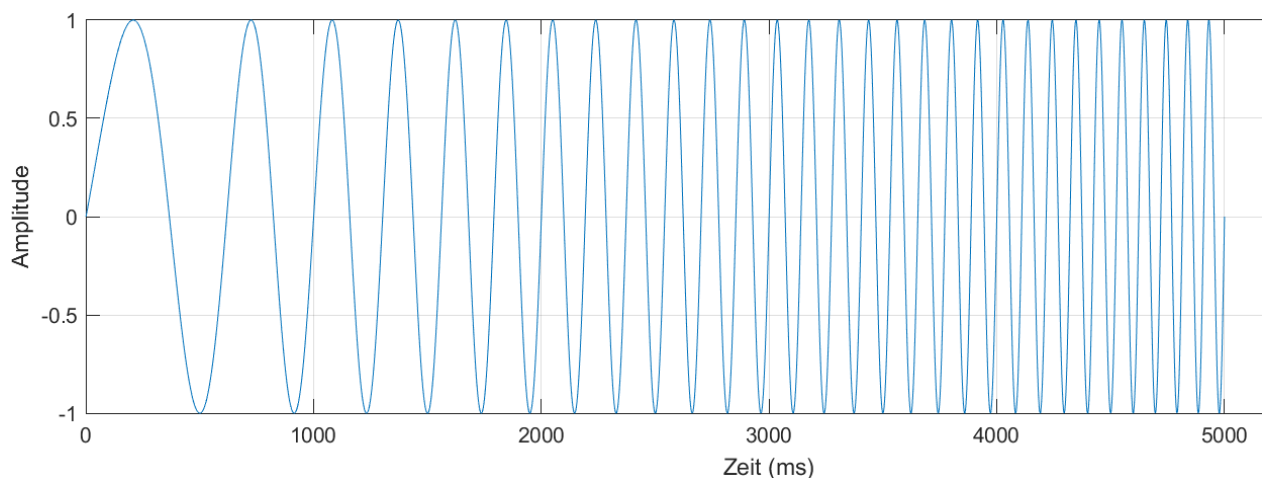


Abbildung 4: Beispielhafter Zeitverlauf eines linear gesweepeten Sinussignals

Logarithmischer Sweep:

Beim logarithmischen Frequenzlauf ändert sich die Frequenz zeitlich nicht mehr konstant, sondern exponentiell in der Form:

$$f(t) = f_0 k^t$$

Die Variable k ist hierbei die Basis der Potenzfunktion und ergibt sich aus dem Verhältnis von End- zu Startfrequenz, potenziert mit dem Kehrwert der Sweepdauer:

$$k = \left(\frac{f_1}{f_0} \right)^{\frac{1}{T}}$$

Analog zum linearen Sweep liefert eine Integration der Frequenzfunktion die Phasenfunktion:

$$\varphi(t) = \varphi_0 + 2\pi \int_0^t f(\tau) d\tau$$

$$\varphi(t) = \varphi_0 + 2\pi f_0 \left(\frac{k^t - 1}{\ln(k)} \right)$$

Somit ergibt sich für die Zeitfunktion eines logarithmisch gesweepen Sinussignals:

$$x(t) = \sin[\varphi(t)] = \sin \left[\varphi_0 + 2\pi f_0 \left(\frac{k^t - 1}{\ln(k)} \right) \right]$$

Ein beispielhafter zeitlicher Verlauf eines logarithmischen Frequenzlaufs ist mit einem weiteren Plot in Abbildung 5 dargestellt, dieser verdeutlicht sehr anschaulich die Nichtlinearität der Frequenzänderung.

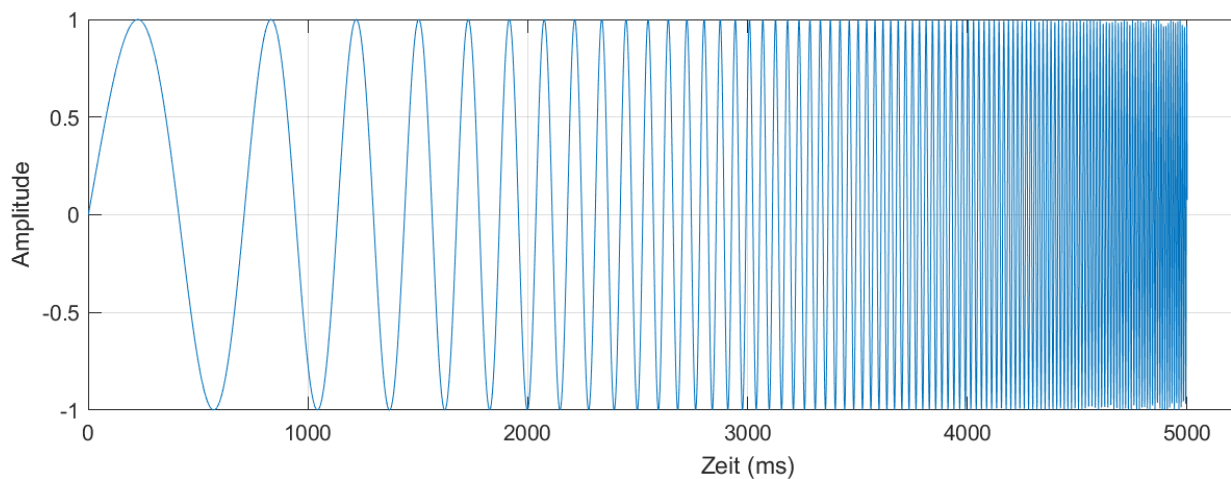


Abbildung 5: Beispielhafter Zeitverlauf eines logarithmisch gesweepen Sinussignals

Verwendete Hardware (Tim)

Arduino Yún

Als Mikrocontrollerplattform wird in diesem Projekt ein Arduino Yún verwendet [6]. Der Arduino Yún besitzt im Gegensatz zu anderen Mikrocontrollerplattformen zusätzlich zu dem Mikrocontroller (ATmega32U4) einen Atheros AR9331 Mikroprozessor, auf welchem ein embedded Linux Echtzeit Betriebssystem läuft. Dieses ermöglicht durch eine wesentlich höhere Rechenleistung als ein Mikrocontroller, unter anderem die Interaktion mit einer Ethernet-/WLAN-Schnittstelle als auch den Umgang mit USB- und SD-Speichermedien (Abbildung 6). In diesem Projekt wurde die WLAN-Schnittstelle des Arduino Yún verwendet.

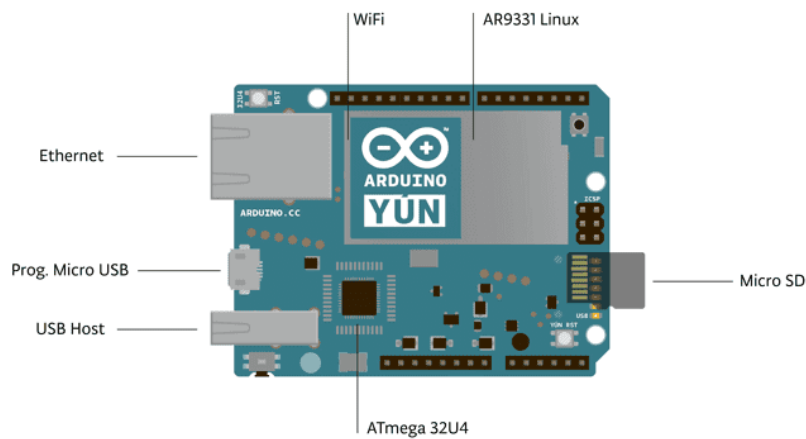


Abbildung 6: Übersicht über die Komponenten des Arduino Yún

ADS9850 (HC-SR08)

Um die Sinus- und Rechteckfrequenz zu erzeugen, wird der ADS9850 der Firma Analog Devices verwendet, dieser ist bereits auf einem Entwicklungsmodul mit der Bezeichnung „HC-SR08“ mitsamt seiner benötigten Peripherie verbaut.

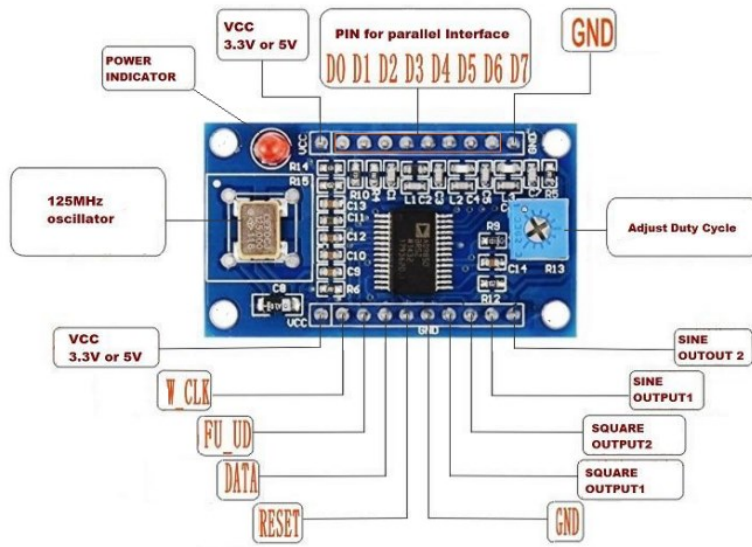


Abbildung 7: Ein- und Ausgänge des HC-SR08 Modul

Der ADS9850 hat sowohl eine parallele als auch eine serielle Schnittstelle, mit welcher die Parameter (Frequenz, Phase und Trimmfrequenz) übergeben werden können. Die eingestellte Frequenz wird dann auf den Ausgängen „SINE OUTPUT 1“ und „SINE OUTPUT 2“ als um die Eingestellte Phase verschobenes Sinussignal ausgegeben. Auf den Ausgängen „SQUARE OUTPUT 1“ und „SQUARE OUTPUT 2“ wird ein Rechtecksignal mit derselben Frequenz ausgegeben.

Zur Ausgabe von Frequenzen mit verschiedenen Phasen wird die „AD9850“ Library verwendet [7].

Bei dieser wird zuerst der AD9850 initialisiert und kalibriert:

```
213 //Start AD9850 and calibrate it
214 DDS.begin(AD9850_W_CLK, AD9850_FQ_UD, AD9850_DATA, AD9850_RESET);
215 DDS.calibrate(trimFreq);
```

Danach kann nun eine Frequenz und eine Phase eingestellt werden:

```
419 DDS.setfreq(staticFreq, phase);
```

LC-Display mit HD44780-Controller

Zur Ausgabe von Informationen an den Benutzer wird ein LC-Display mit 2 Zeilen und 16 Zeichen pro Zeile (16x2) verwendet. Dieses besitzt zur Ansteuerung den Controller HD44780. Dieser ist in der Lage, die Informationen, welche über eine Parallele wahlweise 8- oder 4-Bit Schnittstelle übertragen werden [8], in Zeichen auf dem Display umsetzen. Hierbei befindet im RAM des Controllers ein Standartzeichensatz, es können jedoch auch eigene selbst erstellte Zeichen gespeichert und dann benutzt werden. [9]

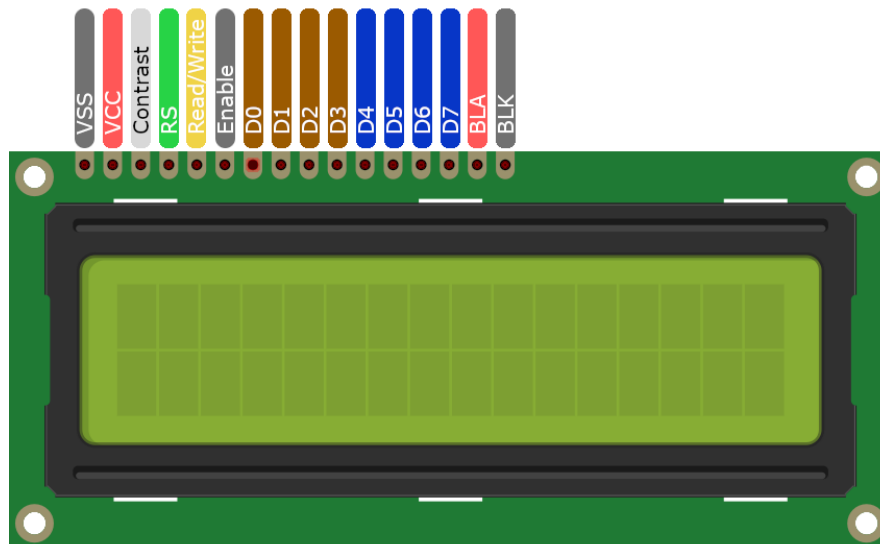


Abbildung 8: Darstellung des LCD mit den Bezeichnungen der Anschlüsse

Die Versorgungsspannung des Displays ist 5V, der Kontrast wird über ein 220 Ω Potentiometer eingestellt. Es wird die 4-Bit Schnittstelle mit den unteren Bits (Bit 4 bis Bit 7) verwendet (Siehe Abbildung 13).

Zur Ansteuerung wird in diesem Projekt die Arduino eigene „LiquidCrystal“ Library verwendet. [10]

Mit dieser können Objekte vom Typ LiquidCrystal erzeugt werden.

```
164 LiquidCrystal lcd = LiquidCrystal(LCD_RS, LCD_EN, LCD_D4, LCD_D5, LCD_D6, LCD_D7);
```

Bei diesen Objekten kann nun der Cursor an die richtige Stelle gesetzt werden, Text ausgegeben oder der gesamte Text gelöscht werden.

```
183 lcd.setCursor(1, 1);
```

```
177 lcd.print(">");
```

```
176 lcd.clear();
```

Drehinkrementalgeber

Zur Interaktion des Benutzers mit dem Signalgenerator wird ein Drehinkrementalgeber (engl. rotary encoder) verwendet. Dieser kann Inkrementweises drehen nach rechts und links sowie ein Pressen des Drehschafes nach unten erkennen. Umgesetzt wird dies über die beiden Schleifkontakte A und B (beim verwendeten Modul als CLK und DT bezeichnet). [11] Diese werden beim Drehen über die Kontaktscheibe entweder auf das Potential mit den logischen Wert 1 oder auf das Potential mit dem logischen Wert 0 (Siehe Abbildungen: Abbildung 9 und Abbildung 10) gezogen. Zur Verwendung des Rotary encoders über Interrupts wird die Library „ClickEncoder“ benutzt. [12]

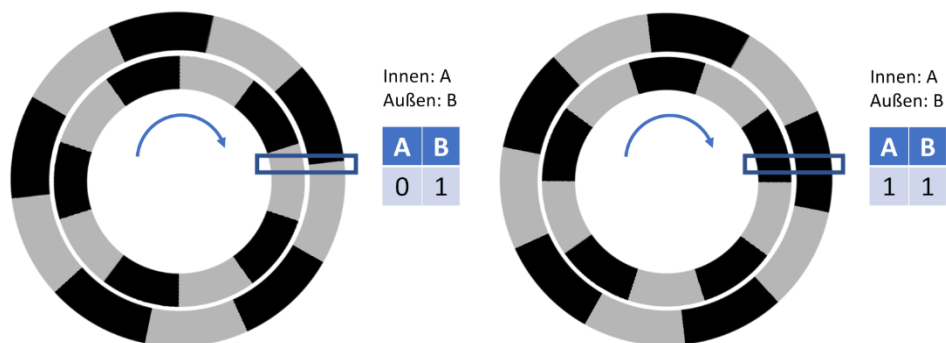


Abbildung 9: Verlauf der beiden Schleifkontakte A und B beim Drehen nach rechts

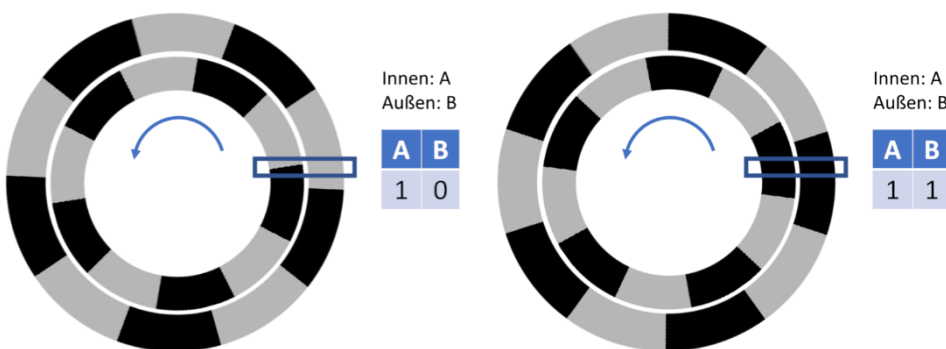


Abbildung 10: Verlauf der beiden Schleifkontakte A und B beim Drehen nach links

Dieses Verhalten lässt sich nun in zwei spezielle, von Arduino interpretierbare, Signale übertragen (Siehe: Abbildung 11 und Abbildung 12).



Abbildung 11: Signal beim Drehen nach links

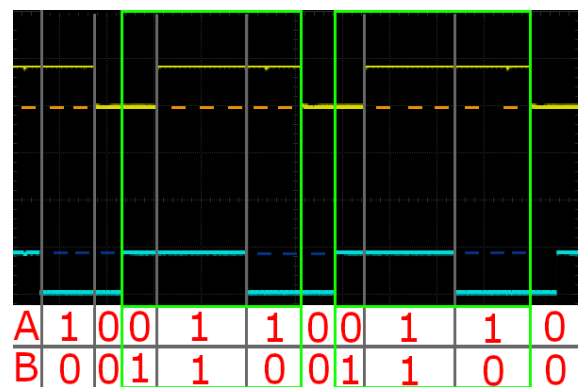


Abbildung 12: Oszilloskop beim Drehen nach rechts

Der Aufbau

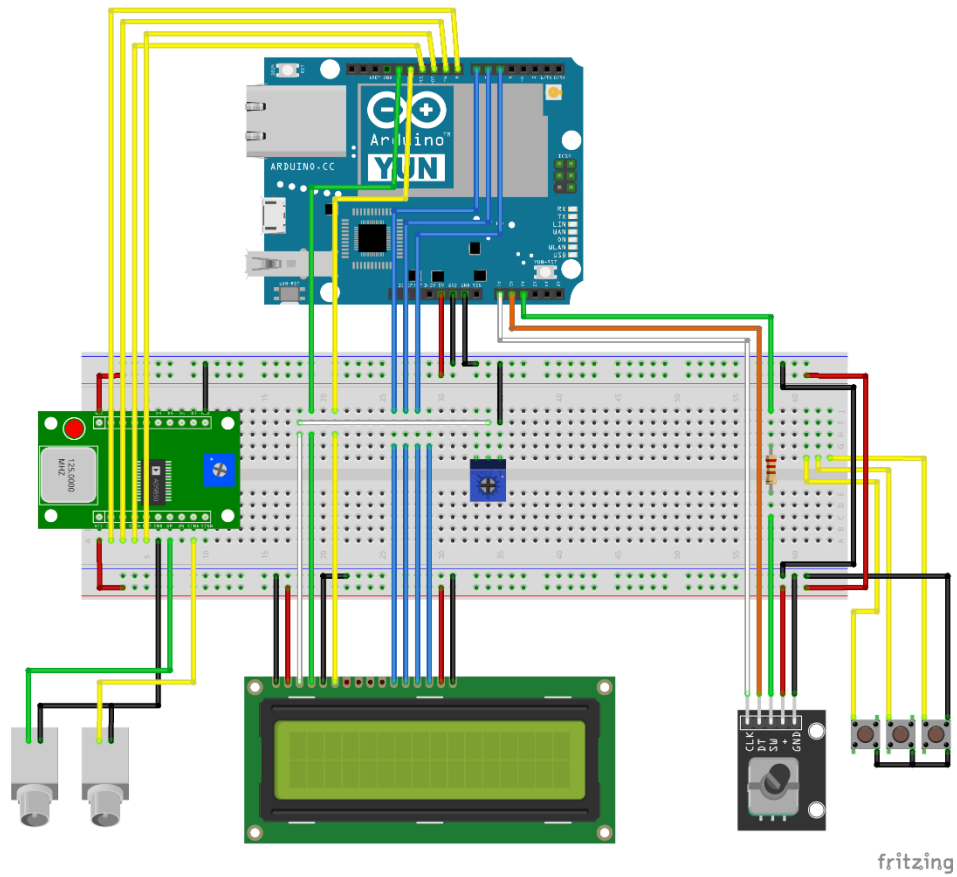


Abbildung 13: Aufbau des Projektes [erstellt mit der Software FRITZING]

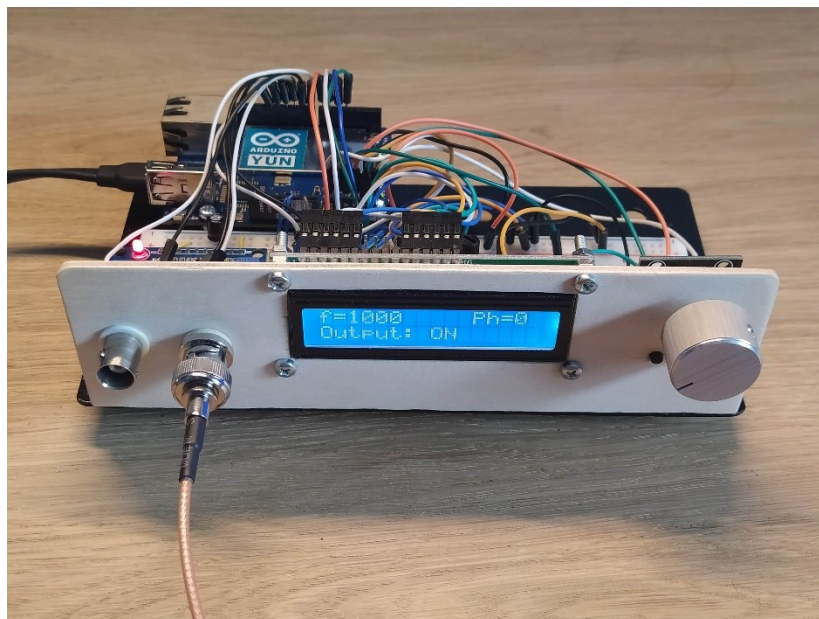


Abbildung 14: Aufbau des Projektes [Eigenes Foto]

Das Arduino Programm (Hannes)

Menüstruktur

Um die verschiedenen Funktionen des Frequenzgenerators am Gerät nutzen zu können, ist zunächst eine grundlegende Menüstruktur gefordert. Diese muss in der Lage sein, mithilfe des Drehinkrementalgebers mit Tastfunktion und des zweizeiligen LCDs alle Unterpunkte anzuwählen, Funktionen aufzurufen, Variablenwerte nach Nutzereingabe richtig einzustellen und notwendige Informationen über das Display auszugeben. Hierzu wird die Open-Source-Bibliothek LCD-Simple-Menu-Library [13] verwendet, welche eine einfache Menüstruktur zur Verfügung stellt und als SimpleMenu.h im Arduino-Sketch eingebunden wird. Es stehen unterschiedliche Menütypen zur Auswahl, welche entsprechend der Anforderungen ausgewählt werden können. Beispielsweise kann ein Menüpunkt in folgender Struktur

```
SimpleMenu(String _name, void (*_Callback)()); //function menu
```

bereits eine weitere Methode aufrufen, ein Menüpunkt dieser Struktur

```
SimpleMenu(String _name,int *_value, int _min, int _max); //Value menu with min and max
```

ermöglicht die Einstellung eines Integer-Wertes innerhalb eines vordefinierten Wertebereichs mithilfe des Drehinkrementalgebers, ohne dass hierfür weiterer Code erforderlich ist. Darüber hinaus stehen natürlich weitere Varianten zum Erzeugen eines Untermenüs ohne weitere Aktionen zur Verfügung.

Die Definition der Menüstruktur erfolgt im Arduino-Sketch unmittelbar nach dem einleitenden Block aus #include-Anweisungen der eingebundenen Bibliotheken, #define-Anweisungen zur Pinbelegung, der Deklaration aller globalen Variablen sowie der Prototypen der im weiteren Programmverlauf referenzierten Funktionen. In unserem Fall wird zunächst ein Hauptmenü angelegt, welches vier Menüpunkte enthält:

```
153 SimpleMenu Menu[4] = {  
154     SimpleMenu("Statische Freq.", 4, MenuSubStatic),  
155     SimpleMenu("Lin. Freqlauf", 6, MenuSubLin),  
156     SimpleMenu("Log. Freqlauf", 6, MenuSubLog),  
157     SimpleMenu("Info", info)  
158 };
```

Die ersten drei Punkte rufen jeweils ein Untermenü zur Einstellung der benötigten Parameter zur Ausgabe einer festen Frequenz oder eines linearen bzw. logarithmischen Frequenzlaufs auf. Der Punkt Info ruft direkt eine Funktion auf, welche die wichtigsten Informationen zum Frequenzgenerator auf dem LCD ausgibt. Das Untermenü für den logarithmischen Frequenzlauf beispielsweise besteht wiederum aus sechs einzelnen Unterpunkten:

```
144 SimpleMenu MenuSubLog[6] = {  
145     SimpleMenu("Startfrequenz", setLogStartFreq),  
146     SimpleMenu("Anzahl Dekaden", &logDecades, 0, 7),  
147     SimpleMenu("Anzahl Schritte", &logSteps, 50, 500),  
148     SimpleMenu("Sweepzeit", setLogSweepTime),  
149     SimpleMenu("Starten", logFreqStart),  
150     SimpleMenu("Zurueck", goBack)  
151 };
```

Neben Funktionen aufrufenden Menüpunkten wird hier zur Anpassung der Dekaden- und Schrittzahl die Möglichkeit der direkten Anpassung eines Variablenwertes genutzt, die Anzahl der Schritte kann somit ganzzahlig zwischen 50 und 500 variiert werden.

Um innerhalb der Menüstruktur navigieren zu können bzw. mit den bereits im Menü enthaltenen Variablen arbeiten zu können, enthält die SimpleMenu-Lib entsprechende Funktionen:

```
32 void home();
33 void select();
34 void back();
35 void returned();
36 void up();
37 void down();
```

```
40 SimpleMenu *next();
41 SimpleMenu *next(int index);
42 int getValue();
43 bool hasValue();
44 int getIndex();
45 void print();
```

Die wichtigsten Funktionen sind hierbei `select()` zum Anwählen eines Menüpunktes und Ausführen der zugehörigen Aktionen, `back()` um auf die hierarchisch nächsthöhere Menüebene zu gelangen, `up()` bzw. `down()` um zum vorigen bzw. nächsten Punkt eines Menüs zu navigieren sowie `getValue()`, um den Wert einer Variable im Menü weiter nutzen zu können. Der Aufruf dieser Funktionen wiederum erfolgt anhand von mit dem Rotary-Encoder getätigten Eingaben.

Programmablauf

Bevor ausgewählte Funktionen näher betrachtet werden, soll zunächst ein Blick auf den zusammenhängenden Programmablauf geworfen werden. Hierbei kann der Programmablaufplan (Abbildung 25) bei einem ersten Überblick unterstützen. Die Kommunikation mit der Remote-Steuerung wird in einem nachfolgenden Abschnitt separat genauer betrachtet.

Zuallererst werden wie bereits erwähnt die verwendeten Bibliotheken eingebunden und anschließend die Anschlussbelegung des Arduinos mittels `#define`-Anweisungen festgelegt. Es folgt die Deklaration und Definition der globalen Variablen sowie die Auflistung der Funktionen-Prototypen. Im `setup`-Bereich des Arduino-Sketches wird die TCP-Bridge gestartet, um die Kommunikation zur remote-Steuerung aufbauen zu können. Außerdem werden alle Bausteine des Frequenzgenerators initialisiert. Es wird ein Objekt vom Typ `ClickEncoder` erzeugt und gemeinsam mit den zugehörigen Interrupts initialisiert. Der AD9850 wird gestartet und einmalig kalibriert, abschließend öffnet sich die „Startseite“ des Hauptmenüs auf dem Display, das Setup ist abgeschlossen.

Die mit jedem Programmzyklus durchlaufene Hauptschleife `loop()` wurde besonders übersichtlich gestaltet und enthält lediglich die beiden Funktionsaufrufe zum Auslesen des Rotary-Encoders sowie der Abfrage der TCP-Daten:

```
225 void loop()
226 {
227     readRotaryEncoder();
228     getTCPData();
229 }
```

Die Funktion `readRotaryEncoder()` überprüft, ob am Drehgeber gedreht oder ob der Taster gedrückt wurde. Hat eine Drehung nach rechts stattgefunden, springt der Cursor im Menü zum darunterliegenden Eintrag, ist gerade eine Eingabefunktion, z.B. zum Einstellen der ausgegebenen Frequenz aktiv, so wird der Wert der zugehörigen Variable inkrementiert. Bei einer Drehung nach links springt der Cursor entsprechend eine Position nach oben bzw. der Variablenwert wird dekrementiert. Falls der Rotary-Encoder gedrückt wurde, findet eine Unterscheidung zwischen kurzem und langem Drücken statt. Bei einem kurzen „Klicken“ wird im Menü der aktuell mit dem Cursor markierte Punkt aufgerufen, bei einem längeren Drücken springt das Menü eine Ebene nach oben bzw. die aktuell ausgeführte Funktion wird verlassen.

Anschließend wird die Funktion `getTCPData()` aufgerufen, auf welche zu einem späteren Zeitpunkt genauere Details folgen. Es folgt das erneute Auslesen des Rotary-Encoders, die `loop()` wird erst wieder verlassen, wenn der Funktionsgenerator vom Netz getrennt wird.

Ausgewählte Funktionen

In diesem Abschnitt werden exemplarisch einzelne Funktionen detaillierter vorgestellt, die von essentieller Bedeutung für die Steuerung des Frequenzgenerators sind oder interessante Eigenschaften aufweisen.

Einstellen eines beliebigen Wertes – `setHighValue`

Viele Funktionen des Frequenzgenerators fordern den Nutzer auf, den notwendigen Wert einer Variable frei zu wählen. Hierbei handelt es sich entweder um Frequenzwerte oder Zeiten, welche jeweils unterschiedliche Wertebereiche umfassen. Der Wertebereich der wählbaren Frequenz entspricht dem Frequenzumfang des DDS-Moduls -also 10Hz bis 40MHz- und umfasst somit mehr als sieben Dekaden. Die Eingabe einer Zeit ist für die Anpassung der Dauer des linearen sowie logarithmischen Frequenzlaufs erforderlich und soll in unserem Fall mikrosekundengenau zu maximal 60 Sekunden gewählt werden können, was einem Wertebereich von „nur“ vier Dekaden entspricht.

Die zum Einstellen dieser Werte genutzte Funktion `setHighValue` besitzt folgenden Funktionskopf:

```
493 void setHighValue(long int *value, int mode) {  
494     valueMode = mode;  
495     stepMode = 0;
```

Als Parameter übergeben werden die Adresse der zu ändernden Variable in Form des Pointers `*value` sowie das Eingabeformat über die Variable `mode`. Eine 1 entspricht hier der Eingabe einer Frequenz, eine 2 sorgt für das Einstellen einer Zeit. Mit `stepMode` kann zwischen verschiedenen Schrittweiten beim Drehen des Rotary-Encoders unterschieden werden. Damit die Einstellung der Werte so lange möglich bleibt, bis der Bediener durch längeres Drücken des Encoders seine Eingabe bestätigt, wird eine `while`-Schleife mit entsprechender Abbruchbedingung implementiert:

```
496     while (1) {  
497         //leave loop, if RotaryEncoder is held long  
498         readRotaryEncoder();  
499         if (doBreak == 1) {  
500             doBreak = 0;  
501             break;  
502         }
```

Wird der Encoder nur kurz geklickt, erfolgt ein Wechsel der Schrittweite, in diesem Fall kann zwischen vier unterschiedlichen Schrittweiten unterschieden werden:

```
503 //increment step mode by clicking the encoder  
504     if (doSelect == 1) {  
505         doSelect = 0;  
506         stepMode++;  
507         if (stepMode > 3) {  
508             stepMode = 0;  
509         }  
510     }
```

Mittels einer Switch-Case-Anweisung wird unterschieden, ob eine Frequenz oder eine Zeit eingestellt wird. Der Code für beide Varianten ist nahezu identisch, es unterscheiden sich lediglich die Maximalwerte und entsprechend die Abstufung der Schrittweiten. Es wird zunächst bestimmt, ob der

Encoder rechts- oder linksherum gedreht wird. Je nach Drehrichtung wird mit jedem Einrasten des Rotary-Encoders der Wert der über den Pointer *value referenzierten Variable um die Schrittweite changeValue inkrementiert bzw. dekrementiert. Werden die Grenzwerte über- bzw. unterschritten, bei der Frequenzeinstellung also Werte <10Hz oder >40MHz, darf keine weitere Erhöhung bzw. Verminderung stattfinden:

```

512     switch (valueMode) {
513     case 1:
514         //if Encoder is turned right:
515         if (turnRight == 1) {
516             turnRight = 0;
517             *value += changeValue;
518             if (*value > 40000000) {
519                 *value = lastValue;
520             }
521         }
522         //if Encoder is turned left:
523         else if (turnLeft == 1) {
524             turnLeft = 0;
525             *value -= changeValue;
526             if (*value < 10) {
527                 *value = lastValue;
528             }
529         }

```

Die Festlegung der Schrittweite wird mithilfe einer weiteren Switch-Case-Anweisung in Abhängigkeit der Variable stepMode realisiert. Bei der Frequenzeinstellung kann zwischen 10Hz-, 100Hz-, 10kHz- und 1MHz-Schritten gewählt werden. Die aktuell gewählte Schrittweite wird auf dem LCD ausgegeben:

```

532         //definig different step sizes
533     switch (stepMode) {
534         //1 step = 10Hz
535         case 0:
536             changeValue = 10;
537             lcd.setCursor(0, 1);
538             lcd.print("Step: 10Hz");
539             break;
540         case 1:
541             changeValue = 100;
542             lcd.setCursor(0, 1);
543             lcd.print("Step: 100Hz");
544             break;
545         case 2:

```

Der aktuelle Wert der Variablen wird abschließend ebenfalls auf dem LCD ausgegeben, dieses wird dabei nur dann aktualisiert, wenn auch wirklich eine Änderung des anzuzeigenden Wertes stattgefunden hat.

case 2 der ersten Switch-Case-Anweisung zur Einstellung einer Zeit ist identisch aufgebaut, der Maximalwert liegt nun jedoch bei 60000(ms) und die einzelnen Schrittweiten betragen 10ms, 100ms, 1s sowie 10s.

Logarithmischer Frequenzlauf – logSweep

Die Struktur der Funktionen zur Durchführung eines linearen bzw. logarithmischen Frequenzlaufs ist nahezu identisch. Da die mathematische Betrachtung beim logarithmischen Lauf jedoch interessanter ist, soll die zugehörige Funktion `logSweep` exemplarisch betrachtet werden.

`logSweep` berechnet aus den übergebenen Parametern Startfrequenz, Anzahl der zu durchlaufenden Dekaden, der Anzahl der Schritte bis zum Erreichen der Endfrequenz und der Solldauer des Sweeps einen Frequenzverlauf, der eine mit fortschreitender Zeit zunehmende Frequenzänderung aufweist, die Frequenz also exponentiell zunimmt. Außerdem sollen alle wichtigen Parameter des konfigurierten Frequenzlaufes während der Signalausgabe auch über das LCD einsehbar sein.

Die bereits im Vorfeld in den entsprechenden Menüpunkten mithilfe der Funktion `setHighValue` angepassten Parameter werden an `logSweep` übergeben:

```
695 void logSweep(long int freqStart, int nDec, int nstep, int tms) {  
696 // freqStart: start frequency  
697 // nDec: number of decades  
698 // nstep: number of sweep-steps  
699 // tms: sweep time in ms
```

Zur Ausgabe eines logarithmischen Frequenzlaufs soll über einen mittels der Sweepzeit festgelegten Zeitraum eine durch die Sweep-Schritte bestimmte Anzahl an Frequenzen zu äquidistanten Zeitpunkten ausgegeben werden. Der Betrag der Frequenz muss dabei so angepasst werden, dass der zeitliche Frequenzverlauf eine exponentielle Form aufweist. Die Anzahl der pro Zeiteinheit ausgeführten Schritte ist dabei ein Maß für die Auflösung des gesweepen Signals. Je höher diese gewählt wird, desto mehr entspricht der Frequenzverlauf einer kontinuierlichen Funktion.

Als erstes kann aus der übergebenen Startfrequenz und der Anzahl der Dekaden die Stoppfrequenz *freqStop* des logarithmischen Laufs bestimmt werden. Die Zeitspanne *dt*, die jede der einzelnen Frequenzen ausgegeben wird, lässt sich ebenfalls aus der übergebenen Sweepzeit und der Anzahl der Schritte bestimmen:

```
732 freqStop = freqStart * pow(10, nDec);  
733 //calculate max. duration of one step  
734 dt = tms / (nstep - 1);
```

Es folgt eine `for`-Schleife, die so oft durchlaufen wird, dass jedem Sweep-Schritt eine eigene Frequenz zugeordnet werden kann. Damit die Schleife jederzeit durch langes Drücken des Rotary-Encoders verlassen werden kann, ist wieder eine entsprechende Abbruchbedingung integriert:

```
737 for (i = 0; i < nstep; i++) {  
738     readRotaryEncoder();  
739     //leave loop, if Encoder is held long  
740     if (doBreak == 1) {  
741         break;  
742     }
```

Um den zeitabhängigen Frequenzwert zu berechnen, wird zunächst eine mathematische Funktion eingesetzt, die allgemein einen exponentiellen Anstieg über eine vorgegebene Anzahl an Schritten beschreibt. Diese hat folgende Form:

$$y(i) = 2^{\log_2(res) \cdot \frac{i+1}{nstep}} - 1$$

Die Variable *res* gibt hierbei den Wertebereich von *y* an. Wird z.B. *res* = 65536 gewählt (16 Bit), kann *y* alle Werte zwischen 0 und 65535 annehmen: Die Zählervariable *i* wird mit jedem

Schleifendurchlauf inkrementiert. Für $i = 0$ läuft der Funktionswert gegen Null, für $i = i_{max} = nstep - 1$ entspricht dieser 65535.

Mithilfe dieses Zusammenhangs kann nun die jedem Schritt i zugehörige, an das DDS-Modul zu übergebende Frequenz errechnet werden:

$$freq(i) = freqStart + \frac{freqStop - freqStart}{res - 1} * y(i)$$

Der Startfrequenz wird also entsprechend dem Wert von i mit jedem Durchlauf eine Frequenzänderung addiert, welche mit jedem Durchlauf exponentiell zunimmt und bei i_{max} die Endfrequenz $freqStop$ erreicht ist. Im Programm wurden diese Berechnungen wie folgt umgesetzt:

```
777 | y = pow(2, (log(res) / log(2)) * (i + 1) / nstep) - 1;
778 | freq = freqStart + ((freqStop - freqStart) / (res - 1.0)) * y;
779 | DDS.setfreq(freq, phase);
```

Alternativ zur Berechnung des exponentiellen Frequenzverlaufs zur Programmlaufzeit kann auch eine im EEPROM abgespeicherte Wertetabelle verwendet werden, welche für eine festgelegte Anzahl an Schritten die zugehörigen Funktionswerte enthält, so wie es z.B. auch beim Nutzen der Pulsweitenmodulation üblich ist [14]. Dadurch könnte die Rechenzeit und dynamische Speicherauslastung weiter optimiert werden, allerdings müssen entsprechende Einbußen bei der Variabilität in Kauf genommen werden, da die Auflösung der Wertetabelle nicht ohne weiteres geändert werden kann. Um die den einzelnen Schritten zugehörigen Frequenzen ausreichend lange auszugeben, wird eine entsprechende Warteschleife am Ende eines jeden Schleifendurchlaufs genutzt:

```
775 | t1 = millis();
782 | //wait until next step can be executed
783 | while ((millis() - t1) < dt) {}
```

Bei der Ausführung eines logarithmischen Sweeps mit einer sehr hohen Schrittzahl ist zu beachten, dass für jeden Schleifendurchlauf eine bestimmte Rechenzeit benötigt wird. Ist aufgrund einer sehr kurzen Sweepzeit die Zeitspanne dt zur Ausführung eines einzelnen Schritts ebenfalls sehr kurz, kann der Fall eintreten, dass die Rechenzeit die Zeitspanne dt überschreitet und das Programm nicht mehr korrekt ausgeführt werden kann. Bei den von uns durchgeführten Frequenzläufen ist dieses Problem bis dato jedoch nicht aufgetreten, Sweeps mit einer Dauer von 1s und 200 Schritten sind ohne Probleme ausführbar.

Um alle wichtigen eingestellten Parameter während der Ausführung eines logarithmischen Frequenzlaufs auf dem Display eindeutig einsehen zu können, wurde aufgrund der begrenzten Displaygröße eine zweiseitige Anzeige umgesetzt. Auf der ersten Seite werden die gewählte Startfrequenz und die Anzahl der zu durchlaufenden Dekaden ausgegeben, die zweite Seite gibt Auskunft über die Anzahl der Wobbelschritte sowie die Dauer eines Sweepzyklus. Der Wechsel zwischen beiden Anzeigen erfolgt zyklisch im Wechsel nach Ablauf einer bestimmten Zeit (interval), in unserem Fall nach 2 Sekunden:

```
744 | if ((millis() - previousMillis) > interval) {
745 |     previousMillis = millis();
746 |     lcd.clear();
747 |     if (page == 0) {
748 |         page = 1;
749 |     }
750 |     else {
751 |         page = 0;
752 |     }
753 | }
```

Kommunikation mit der remote Steuerung (Tim)

Zur Kommunikation mit der Remote Steuerung wird die Arduino eigene „Bridge“ Library verwendet. Diese ermöglicht eine Kommunikation zwischen dem Mikrocontroller und dem Mikroprozessor des Arduino Yún und somit einen Zugriff des Arduino Programms auf die Funktionen des Mikroprozessors, in diesem Fall die TCP/IP Schnittstelle.

Hierfür werden die Klassen „BridgeServer“ und „BridgeClient“ verwendet, um eine bidirektionale Kommunikation zu ermöglichen.

```
161 BridgeServer server(PORT);
```

Nach der Initialisierung und dem Starten des Servers auf dem passenden Port (255) kann nun eine Kommunikation stattfinden. Diese Kommunikation wird von der Funktion getTCPData gesteuert.

```
270 void getTCPData() {
```

Hier wird erst einmal geschaut, ob neue Daten über die TCP Schnittstelle zur Verfügung stehen. Im Regelfall sollte dies eine neu aufgebaute Verbindung sein, da die TCP Verbindung nach jeder Übertragung von Daten wieder abgebaut wird. Ist dies der Fall, wird zuerst der input buffer des client gelöscht, um eventuelle Restdaten aus gescheiterten oder unvollständigen Übertragungen zu löschen. Nun wird ein „DC1-Control Character“ (0x11 = device control 1) gesendet, um dem Remote Programm zu signalisieren, dass die Verbindung korrekt aufgebaut wurde.

```
275 if (client) {
276     if (!alreadyConnected) {
277         // clear out the input buffer:
278         client.flush();
279         client.write(0x11);
280         alreadyConnected = true;
281     }
282
283     if (client.available() > 0) {
```

Ist dies der Fall, so wird nun der empfangene String validiert. Ein korrekt vom Remote programm generierter und fehlerfrei übertragener String beginnt mit einem „STX-Control Character“ (0x02 = start of text). Nun folgen die eigentlichen Zeichen der Zeichenkette. Terminiert wird der String von einem „ETX-Control Character“ (0x03 = end of text). Ist dieser Character erreicht wird kein folgender char mehr in den Char-Array geschrieben. Jetzt folgt auch der „EOT-Control Character“ (0x04 = end of transmission), welcher dem Programm signalisiert, dass keine weiteren Daten folgen und die Verbindung abgebaut werden kann.

```
285     if(thisChar = client.read() == 0x02){
286         while ((thisChar = client.read()) != 0x03){
287             if(thisChar != 0x04){
```


Ist der String ein valider Datenstring im passenden Format wird dieser nun in seine einzelnen Bestandteile „zerschnitten“. Hierzu wird die Funktion strtok (split string into token) verwendet, dies ist eine Funktion der Standardbibliothek der Sprache C++ die eine Sequenz von Buchstaben bis zu einem definierbaren Zeichen aus dem String übergibt und danach diese Zeichen aus dem String löscht. Gleichzeitig wird von der Funktion ein NULL pointer auf die verbleibende Zeichenkette gesetzt, sodass die darauffolgende strtok Funktion nun den NULL pointer als Arbeitspunkt nehmen kann. Als nächstes werden auf diese ausgeschnittenen Teilzeichenketten die Funktionen atoi (Convert string to integer) und atol (Convert String to long integer) angewendet um aus den Zeichenketten, welche durch die Validierung in Remote Programm nur Zahlen enthalten, integer oder long integer zu machen. Diese werden nun den einzelnen Variablen für die Zustandsgrößen des DDS zugewiesen

```
299 |         currentString.toCharArray(workString, 30);
300 |
301 |         mode = atoi(strtok(workString, ";"));
302 |         startFreq = atol(strtok(NULL, ";"));
303 |         stopFreq = atol(strtok(NULL, ";"));
304 |         nDec = atoi(strtok(NULL, ";"));
305 |         nStep = atoi(strtok(NULL, ";"));
306 |         tms = atoi(strtok(NULL, ";"));
```

Zuletzt werden abhängig vom übergebenen Modus die passenden globalen Variablen gesetzt und die jeweilige Funktion gestartet. Hierzu werden die Funktionen der Menüsteuerung benutzt, um sicherzustellen, dass bei einer nachfolgenden Eingabe durch den Benutzer sich das Programm so verhält, als hätte der Benutzer sich selbst zu der Funktion navigiert.

```
308 |         switch(mode) {
309 |             case 1:
310 |                 staticFreq = startFreq;
311 |                 goBack();
312 |                 goBack();
313 |                 TopMenu.index(0);
314 |                 TopMenu.select();
315 |                 TopMenu.index(2);
316 |                 TopMenu.select();
317 |                 break;
```

Am Ende der Funktion werden nun noch das Char-Array und der Arbeitsstring gelöscht, um sicher zu gehen, dass auch im Fehlerfall, in welchem der String nicht komplett verarbeitet wurde in einer darauffolgenden Übertragung von Daten keine Teilzeichenketten übergeben werden können, welche zu Fehlern führen würden.

```
348 |         workString[0] = 0;
349 |         currentString = "";
```


Programm zur remote Steuerung (Tim)

Zur Steuerung und Konfiguration des Frequenzgenerators über den Rechner kann ein in der Programmiersprache Java geschriebenes Programm genutzt werden. Dieses wurde mit der GUI-Bibliothek JavaFX erstellt. [15]

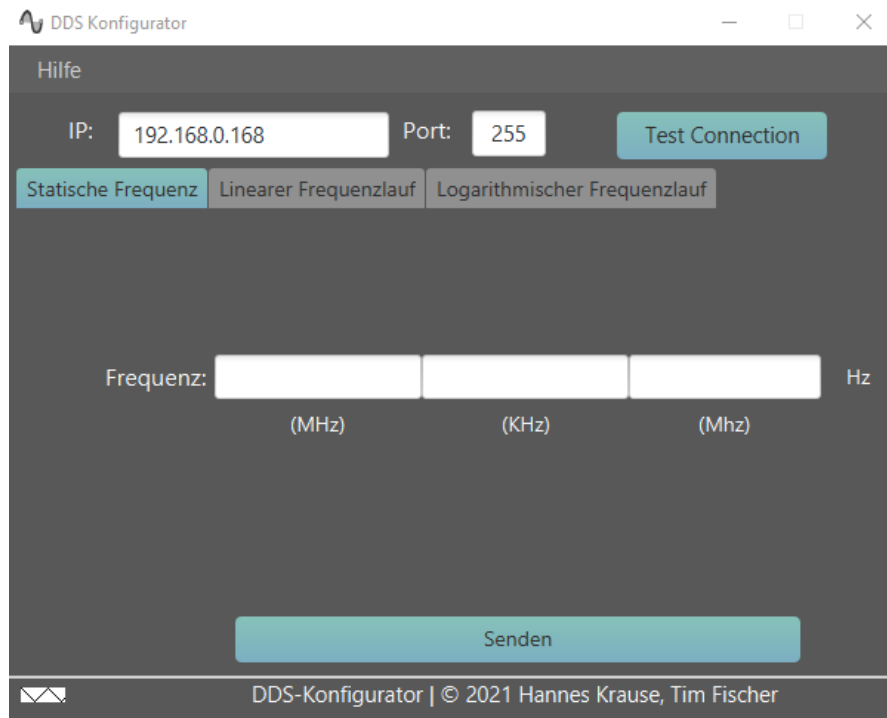


Abbildung 15: Screenshot des Remote-Programms

Die Klassen

App

Diese Klasse stellt sicher, dass das Programm Plattformübergreifend ausgeführt werden kann. Sie ruft beim Starten des Programmes die „main“ Methode der Klasse „Main“ auf.

Main

Wird die Methode main der Klasse „Main“ aufgerufen, so wird zuerst eine Leinwand (Stage) erzeugt, sowie das Standardlayout des Programmes geladen. Danach können die weiteren Funktionen der restlichen Klassen über das GUI aufgerufen werden.

Beim Beenden des Programmes durch den Benutzer wird die Methode closeProgram ausgeführt. Diese kann genutzt werden, um z.B. durch den Nutzer getätigte Einstellungen vor dem Beenden zu speichern.

MainController

Diese Klasse ist die Schnittstelle zwischen dem Hintergrundprogramm (Backend) und der Benutzeroberfläche (Frontend). Sie übergibt z.B. die Eingaben des Benutzers und führt Funktionen bei der Betätigung von Schaltflächen aus. Dafür werden verschiedene Java-Pakete der JavaFX-Bibliothek benutzt. Außerdem wird bei der Übergabe eine Validierung der Eingegebenen Daten durchgeführt, sodass nur Daten übergeben werden, die auch verarbeitet werden können.

Um eine einheitliche Kommunikation zwischen dem Programm und dem Arduino sicher zu stellen, wurde ein festes Datenformat festgelegt. Dieser speziell formatierte String wird aus den übergebenen Daten des Benutzers in der Methode `buildString` erzeugt. (Siehe: Tabelle 1)

Aufbau String:	{(STX)a;b;c;d;e;f(ETX)(EOT)}				
	Name	Wertebereich	Datentyp	Betreffende Modi	Beschreibung
a	mode	1;2;3;(4)	int	alle Modi	Gibt den gewünschten Modus an
b	firstFreq	10 - 40000000	long int	alle Modi	Gibt die erste Frequenz an
c	secFreq	20 – 40000000	long int	2	Gibt die zweite Frequenz an
d	numberDec	1 – 7	int	3	Gibt die Anzahl der Dekaden an
e	numberSteps	1 – 2000	int	2	Gibt die Anzahl der Schritte an
f	timeSweep	500 - 10000	int	2,3	Gibt die Zeit des Frequenzlaufes an


Tabelle 1: Aufbau des an den Arduino übergebenen Strings

Je nach verwendetem Modus werden die nicht verwendeten Teile des Strings mit dem Wert „0“ übergeben.

Modus	Funktion
1	Statische Frequenz kann eingestellt werden
2	Linearer Frequenzlauf
3	Logarithmischer Frequenzlauf
4	Testmodus, um die Übertragung zu testen

Die Validierungsmethoden

In der Klasse `MainController` werden die einzelnen Eingaben validiert, ein Beispiel für so eine Validierungsmethode ist die Methode `isValidNetwork`. An diese Methode kann ein Textfeld für die Eingabe einer IP und ein Textfeld für die Eingabe eines Ports übergeben werden. als Rückgabewert liefert die Methode bei validen Daten ein „true“ und bei invaliden Daten ein „false“.

```
189 @  public boolean isValidNetwork(TextField ip, TextField port){
```

Zuerst wird nun überprüft, ob die Textfelder nicht leer sind und somit zumindest erst einmal Daten beinhalten.

```
194      if((!ip.getText().isEmpty()) && (!port.getText().isEmpty())){
```

Ist dies der Fall, wird als nächstes die IP-Adresse über eine so genannte „Regular expression“(Regex) [16] überprüft.

```
"^((0|1\\d?\\d?|2[0-4]?\\d?|25[0-5]?|[3-9]\\d?)\\.){3}(0|1\\d?\\d?|2[0-4]?\\d?|25[0-5]?|[3-9]\\d?)$";
```

```
196      if(ip.getText().matches(ipPattern))
```

Übersetzung der Regex: Ein String besteht aus 4 Blöcken, welche mit einem Punkt getrennt sind und bei dem jeder Block einen maximalen Wert nach den Vorgaben der IPv4 hat (z.B. hat Block 2-4 als Maximalwert 255)

Als letztes wird nun überprüft, ob der Port eine Eingabe ist, die sich in einen Integer umwandeln lässt und ob sich dieser im passenden Wertebereich befindet. Ist dies der Fall, ist die Eingabe valide und es kann „true“ übergeben werden.

```
197      try{
198          int portInput = Integer.parseInt(port.getText());
199          //check if port is greater than 0
200          return portInput > 0;
201      }
```

Ist eine der Abfragen oder wird eine Exception mit „catch“ gefangen, wird dem Benutzer mit einer Alertbox angezeigt, welche Eingabe falsch war und wie er diesen Fehler beheben kann.

Auf eine ähnliche Art und Weise funktionieren auch die Methoden isValidFrequency, isValidSteps, isValidDecades und isValidTime.

Über die Methode showAbout kann eine About Seite geöffnet werden, welche Hintergrundinformationen über das Programm anzeigt. (Siehe: Abbildung 16)



Abbildung 16: About Seite des Programms

Connection

Diese Klasse ist für den Aufbau der TCP Verbindung zum Arduino sowie für das Übertragen der Daten zuständig. Hierfür wird die Klasse „net“ aus dem Java Paket „java.net“ verwendet. [17]

Die Klasse besitzt die Methode `sendToServer`, diese versucht zunächst eine Verbindung mit der angegebenen IP-Adresse und dem angegebenen Port aufzubauen.

```
22      try (Socket socket = new Socket(IP, port)) {
```

Ist dies erfolgreich gewesen, wird nun der übergebene String gesendet.

```
24      OutputStream output = socket.getOutputStream();  
25      PrintWriter writer = new PrintWriter(output, autoFlush: true);  
26      writer.println(Message);
```

Alertbox

Diese Klasse wird verwendet, um dem Benutzer eine dialogbox auszugeben, wenn es zu einem Fehler kommt. Dies kann entweder passieren, indem die Daten nicht korrekt über TCP gesendet werden können, oder indem der Benutzer eine Eingabe macht, die sich in der Validierung als invalide herausstellt.

Ist dies der Fall, so wird eine Warnung sowie eine Handlungsempfehlung angezeigt. (Siehe: Abbildung 17)

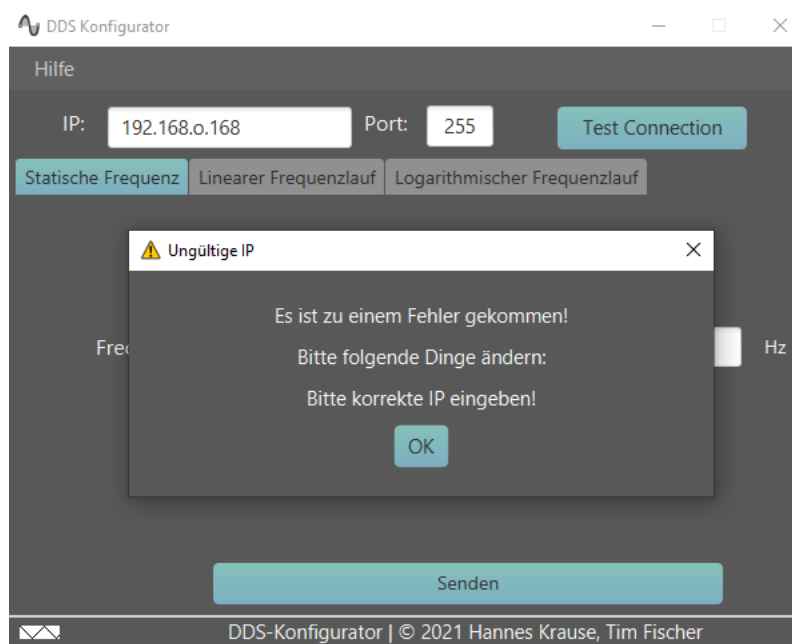


Abbildung 17: Beispiel einer Alertbox (hier wurde in der IP ein o anstatt einer 0 eingetragen)

Messung und Fehleranalyse (Tim)

Um zu betrachten, wie groß der Fehler der ausgegebenen Frequenz sowie die Dämpfung der Amplitude des verwendeten AD9850 ist, wurden mit dem fertigen Aufbau einige Messungen durchgeführt. (Siehe: „Messung der Frequenzen und Amplituden bei verschiedenen Frequenzen“ und „Fast-Fourier-Transformation (FFT) von ausgewählten Signalen“). Alle Messungen wurden mit einem Digitalen Speicheroszilloskop „Rigol DZ1054Z“ mit einer Bandbreite von 100MHz und einer Eingangsimpedanz von 1 M Ω durchgeführt.

Hier kann erkannt werden, dass es ab einer Frequenz von 10 MHz zu einer Dämpfung des Ausgangssignals um ca. 5,5dBu im Vergleich zur Amplitude bei 10Hz kommt. (Siehe: Abbildung 21) (Gemessen wurde hier mit einem Tastkopf direkt hinter dem Ausgangswiderstand des DDS, um das Tiefpassverhalten von Messaufbau gering zu halten)

Bei der maximalen Frequenz des AD9850 (40MHz) kommt es zu einer Dämpfung des Ausgangssignals um ca. 15dBu im Vergleich zur Amplitude bei 10Hz. Gleichzeitig tritt außerdem ein relativer Fehler von 0,75% bei der Frequenz auf. (Siehe: Abbildung 22) Wird die maximal mögliche Frequenz nicht vollständig angereizt, stimmt hingegen die am DDS-Generator eingestellte Frequenz ausgezeichnet mit der gemessenen Frequenz überein, eine Abweichung ist mit der gegebenen Auflösung des Oszilloskops auch im 10MHz-Bereich nicht festzustellen.

Zur Beurteilung der Qualität des erzeugten Sinus-Signals wurden exemplarisch zwei FFT² durchgeführt. Hier zeigt sich, dass der erzeugte Sinus im niedrigen bis mittleren Frequenzbereich ein sehr schmales Spektrum besitzt und kaum weitere Frequenzanteile enthält. Der Rauschanteil bei diesem Signal ist somit gering und es kann auf einen sehr sauberen Sinus geschlossen werden. (Siehe: Abbildung 24)

² FFT: Fast-Fourier-Transformation (Schnelle Fourier Transformation)

Erweiterungs- und Verbesserungsmöglichkeiten(zusammen)

Aufgrund der begrenzten Dauer des Projektes musste auch der umzusetzende Funktionsumfang begrenzt werden. Aus diesem Grund gibt es weitere Ideen für Funktionen des Programms die wir nun noch einmal aufführen möchten:

Arduino Programm:

- Auslagerung von bestimmten Zeichenketten und Werten in den EEPROM des Arduino, um weiteren dynamischen Speicher zu sparen und das Programm somit erweiterbarer und robuster zu machen.
- Anlegen einer eigenen Header-Datei, um das Hauptprogramm durch Auslagerung der Initialisierung kürzer und damit übersichtlicher gestalten zu können
- Optimieren des Eingabekomforts für Frequenzwerte, sodass unterschiedliche Schrittweiten schneller gewählt werden können und beliebige Frequenzen „treffsicherer“ gewählt werden können
- Überarbeiten der Funktion setHighValue zur Reduzierung von Code-Duplikation bei der Einstellung von Frequenz-/Zeitwerten
- Vermehrtes Nutzen von Interrupts zur Reaktion auf Eingaben während der Ausführung einer Funktion
- Erweitern der Funktion logSweep um die Möglichkeit, einen logarithmischen Frequenzlauf von höherer zu niedriger Frequenz auszuführen
- Entwerfen von Funktionen zur Erzeugung weiterer Signalformen, z.B. Sägezahn und Dreieck

Remote Programm:

- Eine Möglichkeit die von Benutzer eingestellten Werte wie z.B. IP-Adresse oder Port Nummer in eine Konfigurationsdatei zu speichern sowie diese Konfigurationsdatei in das Programm zu laden.
- Eine Rückkommunikation vom Frequenzgenerator, um z.B. eine Kommunikation zu unterbinden, wenn vom Benutzer manuell eine Funktion gestartet wurde.
- Um bestimmte Schaltungen testen zu können soll es möglich sein eigene Frequenzgänge per Datei (CSV etc.) zu laden. Beispiel {5 Sekunden sweep bis zur Grenzfrequenz der Schaltung; 5 Sekunden statische Frequenz; 5 Sekunden sweep über Grenzfrequenz hinweg}
- Generelle Analyse des Programmes ob die Zugriffsmodifikatoren von Methoden richtig gewählt sind oder ob Methoden in andere Klassen ausgelagert werden sollten.

Aufbau:

- Eine Platine, die die Verdrahtung auf dem Steckbrett ersetzt den Aufbau somit kompakter und geschützter gegen äußere Einflüsse macht sowie die Hochfrequenzeigenschaften des Aufbaus verbessert
- Ein Gehäuse, welches den DDS umschließt und mit entsprechenden Anschlüssen zur Stromversorgung und zum Datenaustausch versehen ist
 - Ggf. eingebauter Akku, sodass der Signalgenerator ohne Netzanschluss vollständig mobil betrieben werden kann
- Erweiterung um zwei weitere BNC-Buchsen, sodass auch das vom DDS bereitgestellte phasenverschiebbare Signal für Messzwecke verwendet werden kann
- Nutzen eines größeren Displays, um eine übersichtlichere Bedienung zu ermöglichen

Fazit (zusammen)

Mit dem durchgeführten Projekt konnte gezeigt werden, dass mit einem einfachen und kostengünstigen Aufbau (Gesamtkosten der Bauteile: ca. 100€) der Funktionsumfang und der Bedienkomfort von kommerziell erhältlichen Einsteiger-DDS-Signalgeneratoren erreicht und in Teilen sogar übertroffen werden kann.

Die erzeugte Sinusspannung lässt sich sehr zuverlässig exakt aufs Hertz genau einstellen und überzeugt bis zu einem gewissen Anteil der maximal möglichen Frequenz durch einen sehr sauberen Verlauf mit einem geringen Anteil an Störeinflüssen.

Die freie Wahl der Parameter beim linearen und logarithmischen Frequenzlauf eröffnen ein breites Einsatzspektrum in verschiedensten Bereichen der Messtechnik. Einsteigern und Elektronikinteressierten wird somit ermöglicht, z.B. eigene Schaltungen auf das Verhalten bei verschiedenen Frequenzen zu untersuchen und z.B. Grenzfrequenzen von Filterschaltungen messtechnisch zu bestimmen.

Die Möglichkeit der drahtlosen Einstellung aller Parameter und das Starten sämtlicher Funktionen über eine komfortable Benutzeroberfläche am PC dürfte in dieser Preisklasse definitiv ein Alleinstellungsmerkmal sein, wodurch sich dieser DDS-Generator von den käuflichen Geräten abhebt.

Des Weiteren ist es durch den Aufbau (physischer Aufbau wie auch Aufbau der Programme) gut möglich, einzelne Punkte nach den eigenen Wünschen oder bestehenden Anforderungen zu ändern oder zu erweitern (z.B. ein anderes Display zu verwenden oder eine andere Möglichkeit der Eingabe zu verwenden).

Anhang

Messung der Frequenzen und Amplituden bei verschiedenen Frequenzen

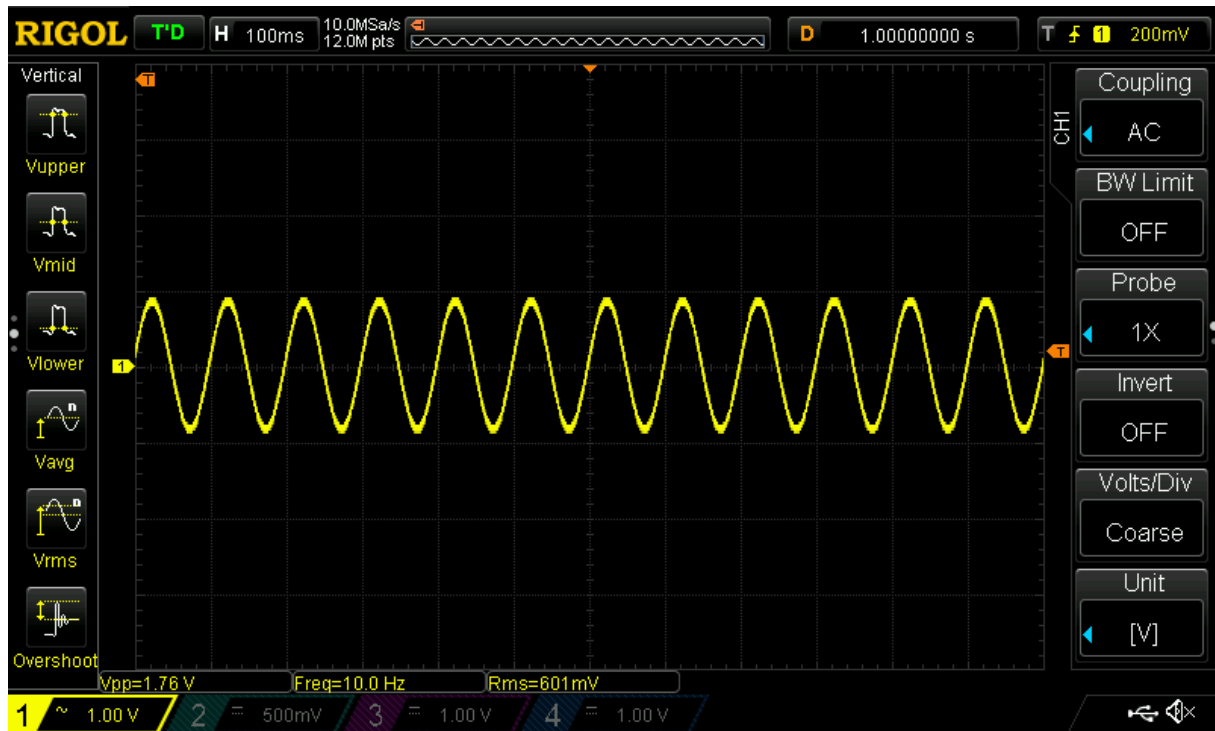


Abbildung 18: Das Signal bei einer Frequenz von 10Hz

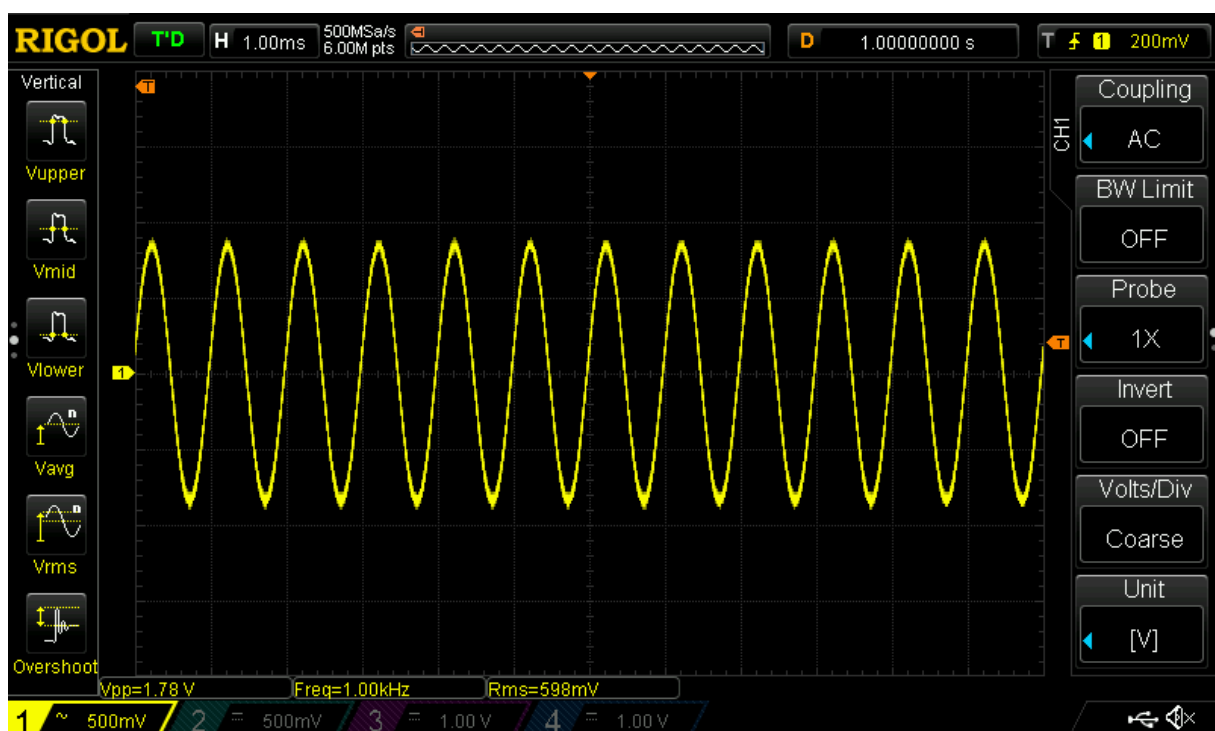


Abbildung 19: Das Signal bei einer Frequenz von 1 kHz

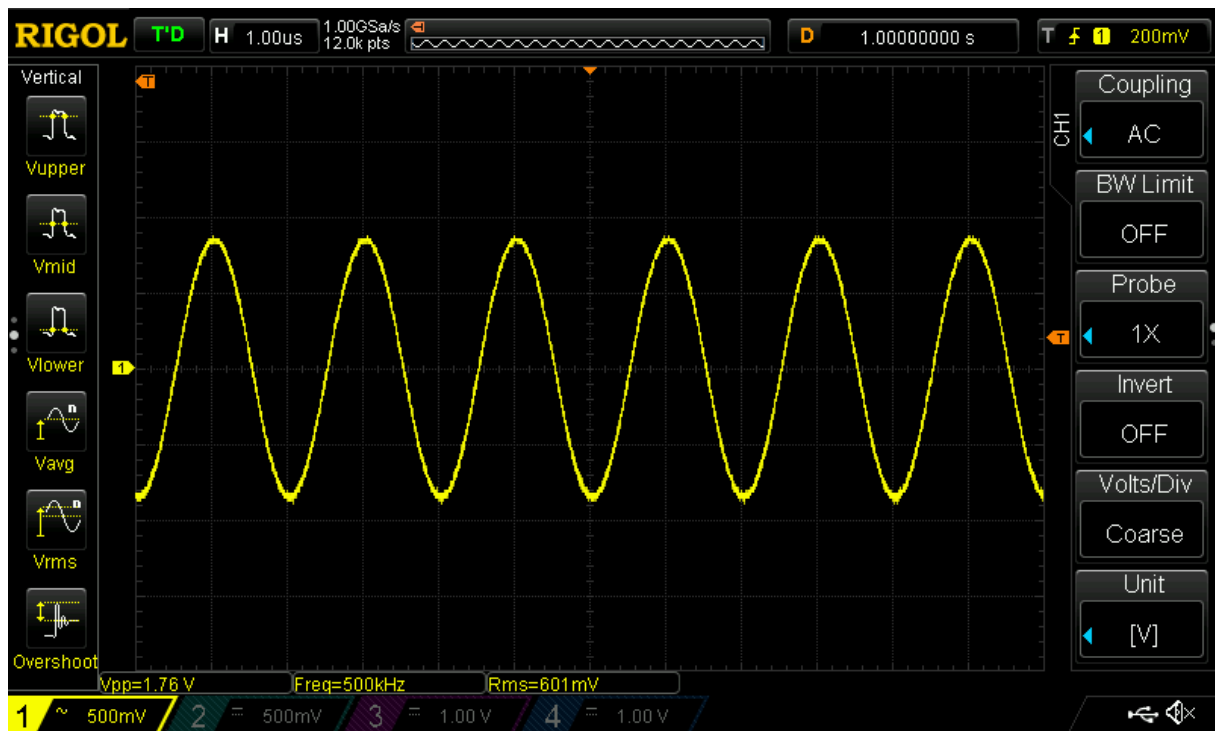


Abbildung 20: Das Signal bei einer Frequenz von 500 kHz

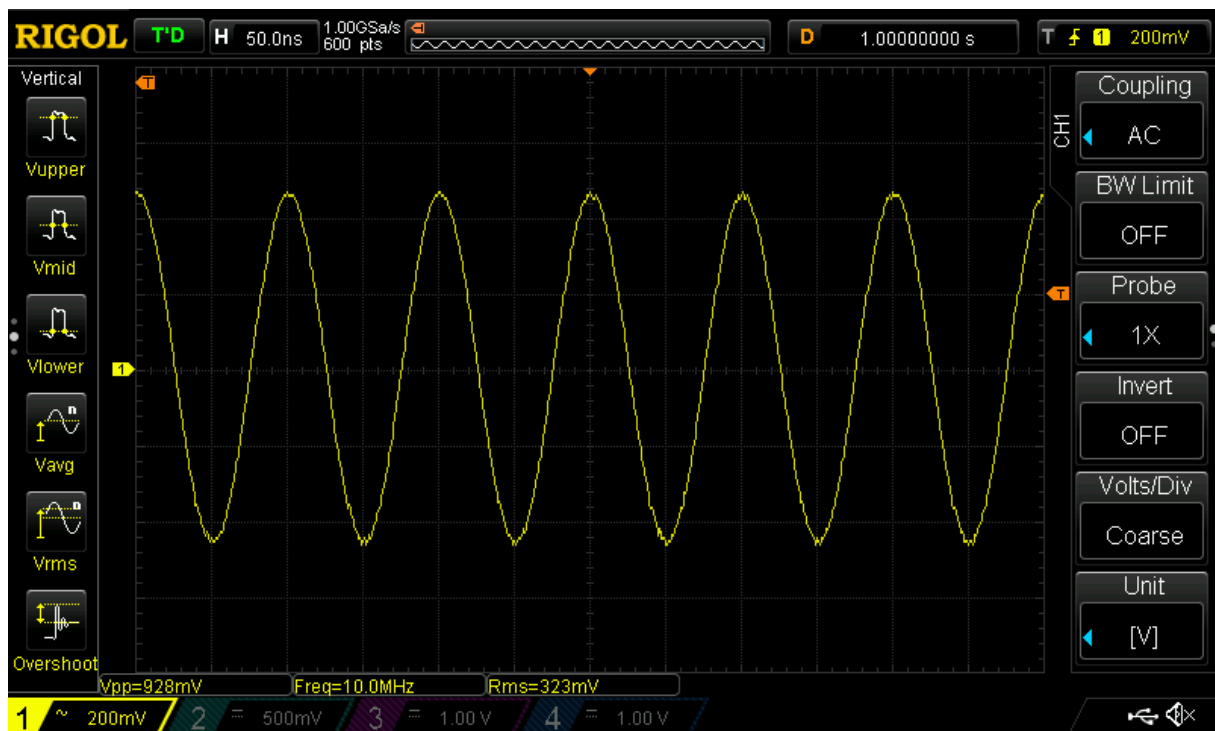


Abbildung 21: Das Signal bei einer Frequenz von 10 MHz

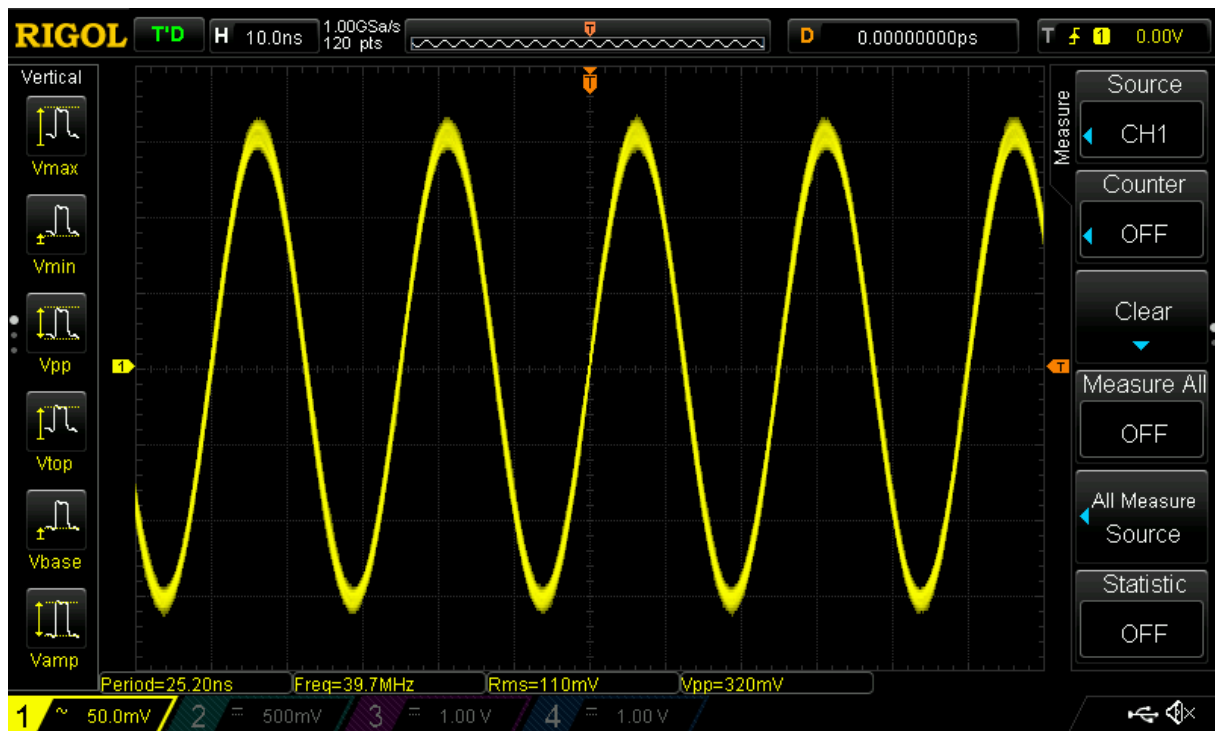


Abbildung 22: Das Signal bei einer Frequenz von 40 MHz [Bandbreite Oszilloskop: 100 MHz]

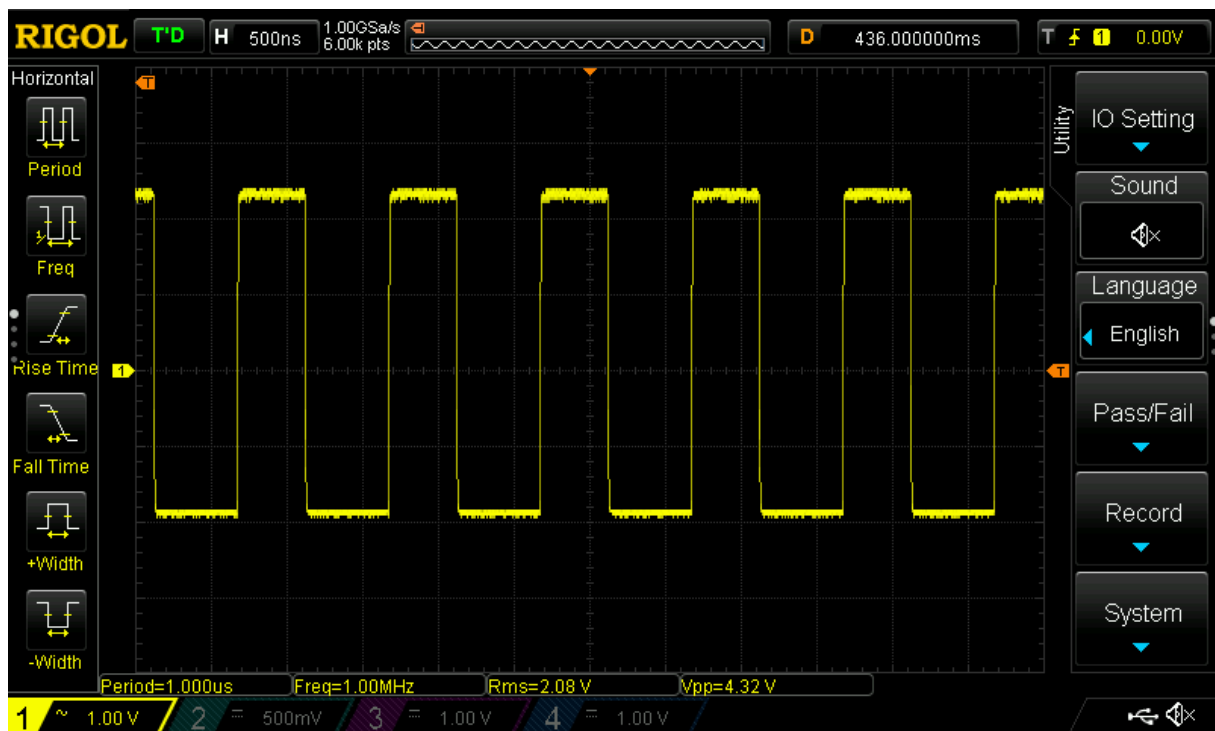


Abbildung 23: Rechtecksignal bei einer Frequenz von 1MHz

Fast-Fourier-Transformation (FFT) von ausgewählten Signalen

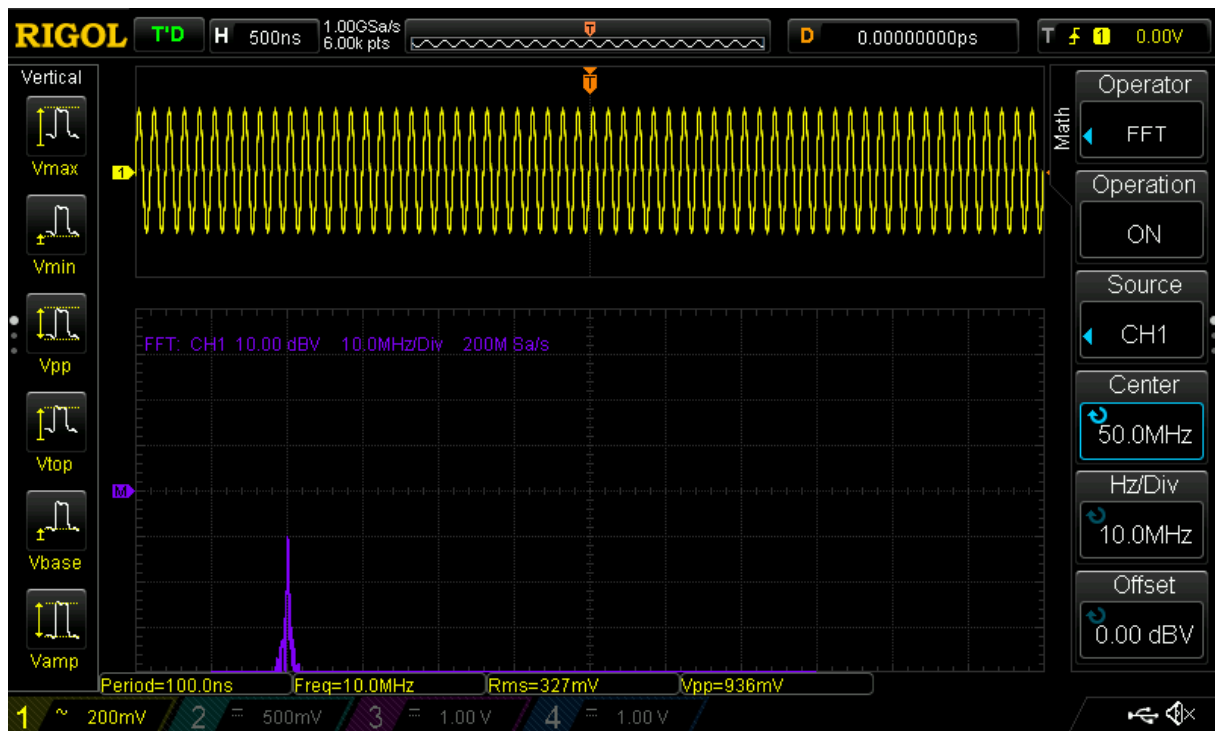


Abbildung 24: FFT bei einem Signal $f=10\text{MHz}$ [Abtastfrequenz: 200MHz]

Zusatzdokumentation Arduino

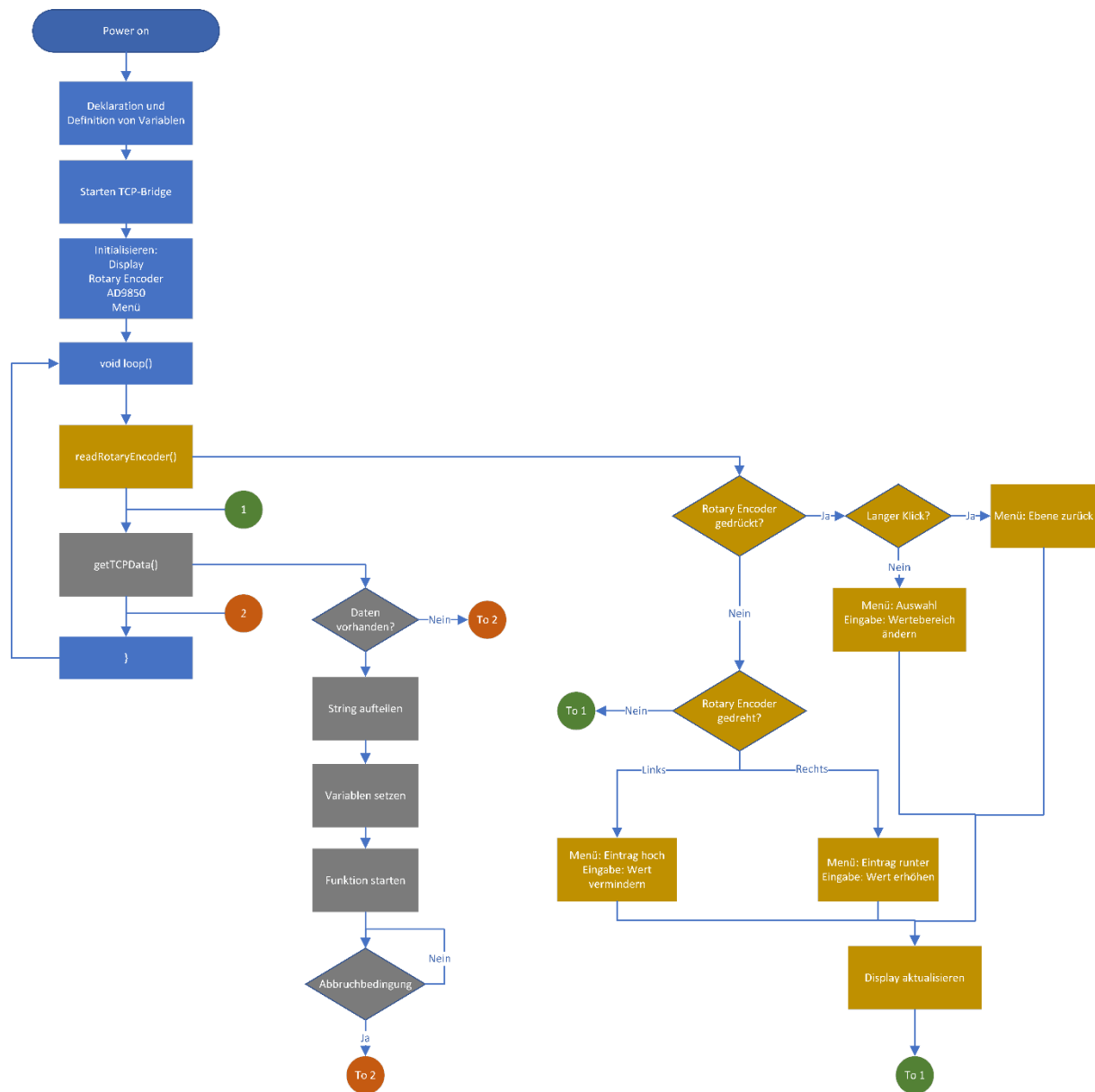


Abbildung 25: Programmablauf Diagramm Arduino Programm

Zusatzdokumentation remote Programm

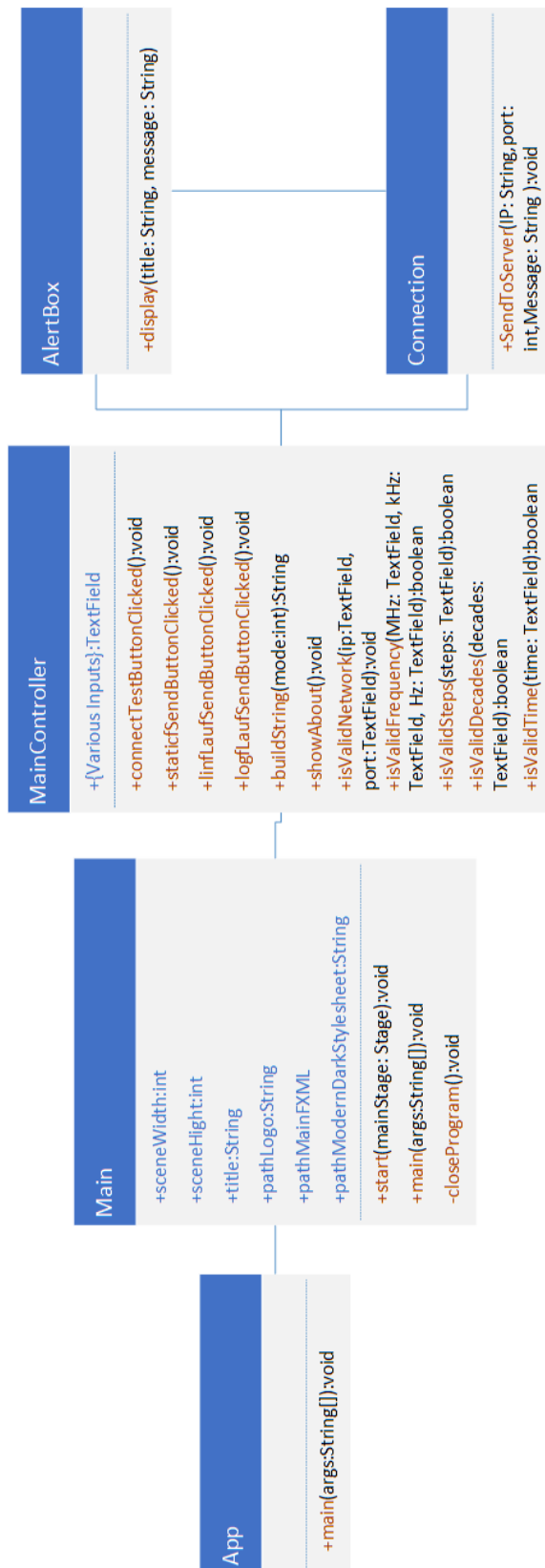


Abbildung 26: Klassendiagramm der remote Steuerung

Abbildungsverzeichnis

ABBILDUNG 1: PRINZIPIELLER AUFBAU EINES DDS SYSTEMS [1] 3	QUELLE: [1]
ABBILDUNG 2: MODELL DES PHASENAKKUMULATORS ALS UMLAUENDER PHASENZEIGER [1] 4	QUELLE: [1]
ABBILDUNG 3: SIGNALFLUSS BEI DER DDS [1] 4	QUELLE: [1]
ABBILDUNG 4: BEISPIELHAFTER ZEITVERLAUF EINES LINEAR GESWEEPTE SINUSSIGNALS 6	QUELLE: EIGENER MATLAB PLOT
ABBILDUNG 5:: BEISPIELHAFTER ZEITVERLAUF EINES LOGARITHMISCH GESWEEPTE SINUSSIGNALS 7	QUELLE: EIGENER MATLAB PLOT
ABBILDUNG 6: ÜBERSICHT ÜBER DIE KOMPONENTEN DES ARDUINO YÚN 8	QUELLE: [6]
ABBILDUNG 7: EIN- UND AUSGÄNGE DES HC-SR08 MODUL 9	QUELLE: [2]
ABBILDUNG 8: DARSTELLUNG DES LCD MIT DEN BEZEICHNUNGEN DER ANSCHLÜSSE 10	QUELLE: EIGENE DARSTELLUNG
ABBILDUNG 9: VERLAUF DER BEIDEN SCHLEIFKONTAKTE A UND B BEIM DREHEN NACH RECHTS 11	QUELLE: EIGENE ZEICHNUNG
ABBILDUNG 10: VERLAUF DER BEIDEN SCHLEIFKONTAKTE A UND B BEIM DREHEN NACH LINKS 11	QUELLE: EIGENE ZEICHNUNG
ABBILDUNG 11: SIGNAL BEIM DREHEN NACH LINKS 11	QUELLE: EIGENE AUFNAHME
ABBILDUNG 12: OSZILLOSKOP BEIM DREHEN NACH RECHTS 11	QUELLE: EIGENE AUFNAHME
ABBILDUNG 13: AUFBAU DES PROJEKTES [ERSTELLT MIT DER SOFTWARE FRITZING] 12	QUELLE: EIGENE SCHALTUNG
ABBILDUNG 14: AUFBAU DES PROJEKTES [EIGENES FOTO] 12	QUELLE: EIGENES FOTO
ABBILDUNG 15: SCREENSHOT DES REMOTE-PROGRAMMS 21	QUELLE: EIGENE AUFNAHME
ABBILDUNG 16: ABOUT SEITE DES PROGRAMMS 23	QUELLE: EIGENE AUFNAHME
ABBILDUNG 17: BEISPIEL EINER ALERTBOX (HIER WURDE IN DER IP EIN O ANSTATT EINER 0 EINGETRAGEN) 24	QUELLE: EIG. AUFNAHME
ABBILDUNG 18: DAS SIGNAL BEI EINER FREQUENZ VON 10Hz I	QUELLE: EIGENE AUFNAHME
ABBILDUNG 19: DAS SIGNAL BEI EINER FREQUENZ VON 1 kHz I	QUELLE: EIGENE AUFNAHME
ABBILDUNG 20: DAS SIGNAL BEI EINER FREQUENZ VON 500 kHz II	QUELLE: EIGENE AUFNAHME
ABBILDUNG 21: DAS SIGNAL BEI EINER FREQUENZ VON 10 MHz II	QUELLE: EIGENE AUFNAHME
ABBILDUNG 22: DAS SIGNAL BEI EINER FREQUENZ VON 40 MHz [BANDBREITE OSZILLOSKOP: 100 MHz] III	QUELLE: EIG. AUFNAHME
ABBILDUNG 23: RECHTECKSIGNAL BEI EINER FREQUENZ VON 1MHz III	QUELLE: EIGENE AUFNAHME
ABBILDUNG 24: FFT BEI EINEM SIGNAL $f=10\text{MHz}$ [ABTASTFREQUENZ: 200MHz] IV	QUELLE: EIGENE AUFNAHME
ABBILDUNG 25: PROGRAMMABLAUF DIAGRAMM ARDUINO PROGRAMM	QUELLE: EIGENE ZEICHNUNG
ABBILDUNG 26: KLASSENDIAGRAMM DER REMOTE STEUERUNG	QUELLE: EIGENE ZEICHNUNG

Literaturverzeichnis

- [1] C. S. Eva Murphy, „All About Direct Digital Synthesis,“ *Analog Dialogue*, August 2004.
- [2] Analog Devices, *CMOS, 125MHZ Complete DDS Synthesizer AD9850*, Norwood, MA 02062-9106, U.S.A.: Analog Devices, 2004.
- [3] Wikipedia, „Direct Digital Synthesis,“ [Online]. Available: https://de.wikipedia.org/wiki/Direct_Digital_Synthesis. [Zugriff am 04 12 2021].
- [4] mikrocontroller.net, „DDS,“ [Online]. Available: <https://www.mikrocontroller.net/articles/DDS>. [Zugriff am 29 11 2021].
- [5] Wikipedia, „Chirp,“ [Online]. Available: <https://en.wikipedia.org/wiki/Chirp>. [Zugriff am 04 12 2021].
- [6] A. Foundation, „Arduino.cc,“ 05 02 2018. [Online]. Available: <https://www.arduino.cc/en/Guide/ArduinoYun>. [Zugriff am 29 11 2021].
- [7] F4GOJ, „github.com,“ 21 07 2018. [Online]. Available: <https://github.com/F4GOJ/AD9850>. [Zugriff am 05 12 2021].
- [8] HITACHI, „Sparkfun,“ [Online]. Available: <http://www.sparkfun.com/datasheets/LCD/HD44780.pdf>. [Zugriff am 01 12 2021].
- [9] v. A. (. Versionsgeschichte), „Mikrocontroller.net - HD44780,“ [Online]. Available: <https://www.mikrocontroller.net/articles/HD44780>. [Zugriff am 01 12 2021].
- [10] Arduino.cc, „LiquidCrystal,“ 24 12 2019. [Online]. Available: <https://www.arduino.cc/en/Reference/LiquidCrystal>. [Zugriff am 05 12 2021].
- [11] Wikipedia, „Inkrementalgeber,“ [Online]. Available: <https://de.wikipedia.org/wiki/Inkrementalgeber>. [Zugriff am 01 12 2021].
- [12] 0xPIT, „ClickEncoder,“ 02 05 2017. [Online]. Available: <https://github.com/0xPIT/encoder>. [Zugriff am 06 12 2021].
- [13] GitHub, „LCD-Simple-Menu-Library,“ [Online]. Available: <https://github.com/tinkersprojects/LCD-Simple-Menu-Library>. [Zugriff am 01 12 2021].
- [14] mikrocontroller.net, „LED-Fading,“ [Online]. Available: <https://www.mikrocontroller.net/articles/LED-Fading>. [Zugriff am 06 12 2021].
- [15] „openjfx.io,“ Gluon, [Online]. Available: <https://openjfx.io/>. [Zugriff am 06 12 2021].
- [16] Wikipedia, „Regular expression (Regex),“ [Online]. Available: https://en.wikipedia.org/wiki/Regular_expression. [Zugriff am 05 12 2021].
- [17] Oracle, „docs.oracle.com,“ [Online]. Available: <https://docs.oracle.com/javase/8/docs/api/java/net/package-summary.html>. [Zugriff am 05 12 2021].

