Fachhochschule Nordwestschweiz
Hochschule für Technik

digitales Theremin
Fachbericht

Projekt 6
20. August 2020

Auftraggeber:	Prof. Dr. Hanspeter Schmid
Betreuung:	Prof. Dr. Hanspeter Schmid Herr Prof. Karl Schenk
Team:	Andreas Frei Dennis Aeschbacher
Studiengang:	Elektro- und Informationstechnik
Semester:	Frühlingssemester 2020

Abstract

In this Project a Theremin was built that mainly operates on digital hardware unlike the original device that solely used analog electronics. The device is supposed to be used in presentations for trade fairs by the Institute for Sensors and Electronics ISE. As such the device should be built in a appealing housing. Moreover the device should have other additional functionality such as soundeffects or the ability to record sound. The digital hardware was implemented in VHDL on the developer board DE1-SoC from terasIC with a Cyclone V FPGA from Intel. The sole analog component implemented was the oscillator that controls the pitch. The pitch of the device can be changed well, but the sound itself has a flaw at the moment, because there is an audible crackle. This is due to a communication problem with the codec that was used for the audio output. This problem will not be corrected during this project, because the communication will be implemented differently in the finished product. The work in this project served as a platform for the continuation in project 6. The next steps would be to implement the volume control and redesign the pcb for two antennas and oscillators.

Inhaltsverzeichnis

1 Einleitung

Das Theremin kennen heutzutage nur wenige Leute, obwohl es das erste elektronische Instrument war. Es wurde 1920 von dem Russen Lev Sergejewitsch Termen, welcher sich später zu Leon Theremin umbenennen liess, erfunden [1]. Personen die regelmässig Filme schauen, haben die Musik welche mit einem Theremin gemacht wird bestimmt schon einmal gehört. Ein Beispiel dafür ist Ghostbusters, wo das Theremin oft im Hintergrund zu hören ist. Zudem ist das Theremin in einigen Science-Fiction-Filmen und Horrorfilmen zu hören [2]. Das Theremin wird ohne es zu berühren gespielt, indem man mit den Händen die Distanz zu zwei Antennen ändert. Dies führt zu Veränderung der Tonhöhe und Lautstärke.

Im Projekt 5 und 6 soll nun ein solches Instrument entwickelt werden. Mit dem Unterschied, dass das sonst analoge Instrument digital aufgebaut werden soll. Dabei soll es auf einem Field Programmable Gate Array (FPGA) implementiert werden. Später soll das Theremin als Messeobjekt für das Institut für Sensorik und Elektronik ISE verwendet werden. Im Rahmen des Projekt 5 wurde die Tonhöhenantenne des Theremin realisiert. Dazu wurde die Antenne zusammen mit dem Antennenoszillator analog beibehalten. Die restlichen Komponenten wurden in VHDL realisiert. Das Resultat wurde auf dem DE1-SoC Board von terasIC mit einem Cyclone V FPGA von Intel getestet.

Der folgende Fachbericht beginnt mit dem Kapitel 2 Technische Grundlagen.In der ersten Hälfte des Kapitel wird erklärt wie ein analoges Theremin funktioniert und welche Komponenten ein Theremin ausmachen. In der zweiten Hälfte werden digitale Lösungsansätze besprochen. Anschliessend wird im Kapitel 4 Realisierung beschrieben wie die Komponenten realisiert wurden. Im Kapitel 7 Validierung wird als erstes auf die Inbetriebnahme des Antennenoszillators eingegangen. Als nächstes werden die Simulationen des VHDL Codes erläutert. Im letzten Abschnitt wird auf die Inbetriebnahme des VHDL Codes auf dem DE1-SoC Board Bezug genommen.

Erklärung zu Glissando und Anzeige Spielgenauigkeit

2 Technische Grundlagen

In diesem Kapitel ist als erstes erklärt wie ein analoges Theremin funktioniert um ein Verständnis für dieses zu gewinnen. Anschliessend folgt eine kleiner Abschnitt zur Musiktheorie. Zuletzt werden verschiedene Algorithmen erklärt, welche im digitalen Theremin eingesetzt wurden.

2.1 analoges Theremin

Das klassische Theremin besitzt zwei Antennen. Der Spieler kann über die senkrecht angebrachte Antenne die Tonhöhe beeinflussen. Mit der waagrechten Antenne beeinflusst der Spieler die Lautstärke. Eine typische Eigenschaft des Theremin ist, dass der Ton des Theremin in einem weiten Frequenzbereich kontinuierlich veränderbar ist. Das Theremin ist daher nicht auf die Tonleiter beschränkt.

Das Theremin erzeugt Töne durch das verstimmen des an der Antenne angebrachten Oszillators. Die Hand des Spielers, der durch seine eigene Masse als Erdung fungiert, verändert über die jeweilige Antenne den LC-Schwingkreis des Tonhöhen und Lautstärke Oszillators. Dabei wird der Kapazitive Anteil des Schwingkreis beeinflusst, was eine Änderung der Schwingfrequenz zur Folge hat. Die Frequenz dieser Oszillatoren ist jedoch weit über dem hörbaren Bereich (zwischen 100 kHz bis 1 MHz). Mit Hilfe eines Mischers und einem Referenzoszillator wird die Frequenzdifferenz hörbar gemacht und danach verstärkt[3]. Die einzelnen Schaltungsteile sind im folgenden Teil genauer erklärt.

Antennenoszillator und Tonhöhenantenne

Die Tonhöhenantenne ist ein Metallrohr welches mit dem Antennenoszillator verbunden ist. Durch die Hand des Spielers wird über die Tonhöhenantenne die Frequenz des Antennenoszillators verändert. Die Kapazitätsänderung welche über die Antenne erreicht werden kann ist sehr gering. Diese liegt im Picofarad Bereich [4]. Um eine genug grosse Frequenzänderung zu erzeugen muss demnach die Grundfrequenz des Antennenoszillators weit über dem hörbaren Frequenzbereich gewählt werden.

Antennenoszillator und Lautstärkeantenne

Die Lautstärkeantenne ist wie die Tonhöhenantenne ein Metallrohr, welches mit dem Antennenoszillator verbunden ist. Die durch den Spieler beeinflusste Frequenzänderung wandelt ein Hüllkurvendetektor in eine Spannungsänderung um. Diese Spannungsänderung dient dem Verstärker als Steuergrösse um das Audio Signal zu verstärken. [3].

wikipedia

Mischer und Referenzoszillator

Die erzeugte Frequenz der Tonhöhenantenne ist weit über dem vom Menschen hörbaren Bereich. Deswegen wird das Antennen Signal mit einem Referenzoszillator mit fester Frequenz gemischt. Dies bewirkt eine Frequenzdifferenz im hörbaren Bereich. Um diese Differenz zu erhalten multipliziert der Mischer die zwei Signale des Referenzoszillator $A_1 \sin(\omega_1)$ und des Antennenoszillator $A_2 \sin(\omega_2)$ wie folgt:

$$V_{out} = A_1 A_2 \sin(\omega_1 t) \sin(\omega_2 t) \tag{2.1}$$

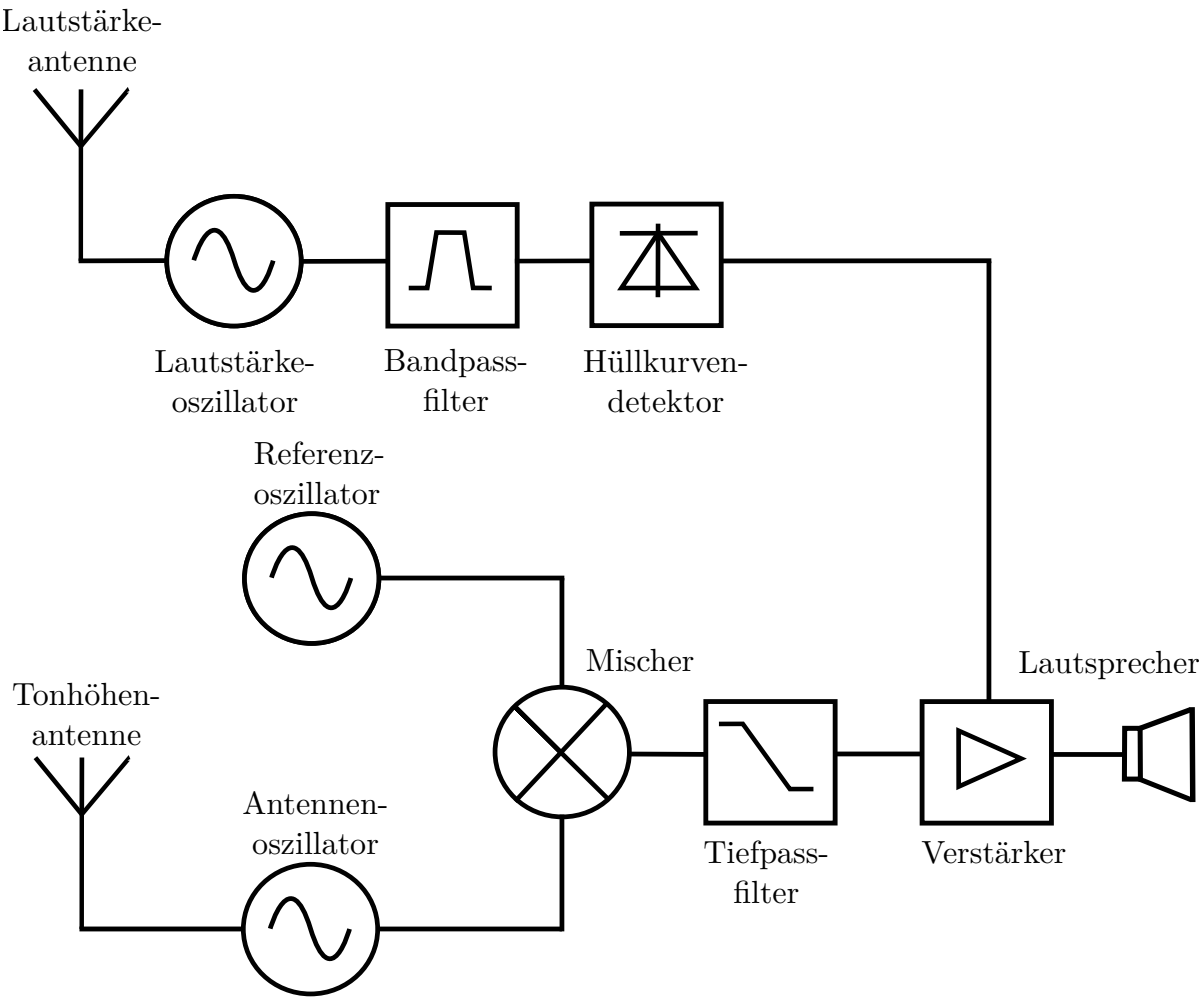


Abbildung 2.1: Blockschaltbild eines analogen Theremins

V_{out} kann durch Additionstheoreme umgeformt werden. Dabei erhält man folgenden Ausdruck:

$$V_{out} = A/2[\cos((\omega_1 - \omega_2)t) - \cos((\omega_1 + \omega_2)t)] \tag{2.2}$$

Das Ausgangssignal V_{out} hat zwei Frequenzkomponenten. Zum einen die Differenz der beiden Frequenzen zum anderen die Summe der Frequenzen. Dabei ist bei dem Theremin nur die Differenz der Frequenzen von Interesse [4].

Das Theremin muss vor jedem Gebrauch kalibriert werden. Es könnte beispielsweise sein, dass die Differenz der Frequenz ausserhalb des hörbaren Bereiches ist. Dazu wird mit Hilfe eines Trimmkondensators am Referenzoszillator die Differenzfrequenz auf 0 Hz gestellt.

Tiefpassfilter

Mit Hilfe eines Tiefpassfilters wird das Signal mit der Summe der Oszillator Frequenzen vollständig abgeschwächt. Übrig bleibt die Differenz der Oszillator Frequenzen. Dieser ist der Anteil des Mischprozesses, welcher von Intresse ist, da er im hörbaren Bereich ist.

$$V_{out} = A/2cos((\omega_1 - \omega_2)t) \tag{2.3}$$

Verstärker und Lautsprecher

Die hörbare Differenz wird verstärkt und über einen Lautsprecher ausgegeben.

2.2 Musiktheorie

Um besser an einem Musikinstrument arbeiten zu können ist es wichtig ein wenig Musiktheorie zu kennen. Der wichtigste Fakt ist wohl, dass der Schallpegel logarithmisch wahrgenommen wird und in Dezibel (dB) gemessen wird. Auch die Frequenz der Tonhöhe hören wir nicht linear. Ein Ton mit 400 Hz nehmen wir nicht als doppelt so hoch wahr als ein Ton mit 200 Hz. Dies ist sehr schön ersichtlich in Tabelle 2.1. Je höher die Töne werden, desto grösser werden die Frequenzunterschiede. Für einfacheres Rechnen von diesen Unterschieden wird die Masseinheit Cent gebraucht. Dabei ist definiert, dass zwei Töne 100 Cent auseinanderliegen und dass zwei Töne mit einer Oktave Unterschied 1200 Cent Frequenzunterschied haben. Diese Cent-Werte kann man mithilfe von Formel 2.4 in einen Faktor umrechnen [5].

$$x = \sqrt[1200]{2}^{n_{cent}} \tag{2.4}$$

Dabei ist n_{cent} der Unterschied in Cent und x als Faktor.
Ist nun die “akustische” Mitte zwischen Zwei Tönen gesucht ist die Berechnung mit Cent nützlich. Diese Mitte liegt nicht linear zwischen den beiden Tönen sondern 50 Cent entfernt von beiden Tönen. Werden diese 50 Cent in einen Faktor umgerechnet und mit dem tieferen Ton multipliziert erhält man die Mitte.

Um nun zu sagen, wann ein Unterschied in der Frequenz vom Gehör wahrgenommen wird, kommt es ganz auf die Person drauf an. Als Faustformel kann gesagt werden, dass zwei aufeinanderfolgende Töne mit etwa 6 Cent unterschied vom Gehör registriert wird. Jedoch wann ein Ton als ”nicht getroffen“ empfunden wird ist schwierig zu sagen.[5]

Ein weiteres interessantes Thema ist die Pentatonik. Dabei handelt es sich um ein Tonsystem mit nur 5 Tönen. Als Beispiel kann das Klavier genommen werden. Benützt der Spieler nur die Schwarzen Tasten des Klavier, so würde er in einem pentatonischen Tonsystem spielen. In Abbildung 2.1 entspricht dies allen Tönen mit einem # in der Notation. Ein Merkmal der Pentatonik ist, dass es sehr einfach ist eine Melodie zu spielen, die ansprechend klingt, ohne grossen Aufwand.[6]

Tabelle 2.1: Töne aus vier Oktaven und deren Frequenzen [7]

Ton	**Frequenz[Hz]**	**Ton**	**Frequenz[Hz]**	**Ton**	**Frequenz[Hz]**
C3	130.813	F4	349.228	A#5	932.328
C#3	138.591	F#4	369.994	B5	987.767
D3	146.832	G4	391.995	C6	1046.5
D#3	155.563	G#4	415.305	C#6	1108.73
E3	164.814	A4	440	D6	1174.66
F3	174.614	A#4	466.164	D#6	1244.51
F#3	184.997	B4	493.883	E6	1318.51
G3	195.998	C5	523.251	F6	1396.91
G#3	207.652	C#5	554.365	F#6	1479.98
A3	220	D5	587.33	G6	1567.98
A#3	233.082	D#5	622.254	G#6	1661.22
B3	246.942	E5	659.255	A6	1760
C4	261.626	F5	698.456	A#6	1864.66
C#4	277.183	F#5	739.989	B6	1975.53
D4	293.665	G5	783.991	C7	2093
D#4	311.127	G#5	830.609		
E4	329.628	A5	880		

2.3 Cordic Algorithmus

Um in einem FPGA aufwendigere Rechenoperationen wie die Berechnung eines Sinus zu implementieren ist eine zusätzliche Hardware notwendig. Der Cordic Algorithmus kann für diesen Zweck eingesetzt werden. Nebst anderen diversen Rechenoperationen ist der Einsatz als Sinusgenerator möglich, was in späteren Kapiteln genauer besprochen ist.
Der Cordic Algorithmus ist ein iterativer Algorithmus welcher praktisch nur Additionen und Verschiebungen von Bits benötigt. Er kann in zwei Modi betrieben werden. Zum einen der Vektor Modus, in welchem die Berechnung eines Winkels aus einem gegebenen Vektor möglich ist. Zum anderen der Rotationsmodus, mit welchem aus einem gegebenen Winkel die Elemente des zugehörigen Vektors berechnet werden können. Die folgenden Formeln sind für diese Berechnung notwendig [8]:

$$x_{i+1} = x_i - y_i d_i 2^{-i} \tag{2.5}$$

$$y_{i+1} = y_i + x_i d_i 2^{-i} \tag{2.6}$$

$$z_{i+1} = z_i - d_i \arctan 2^{-i} \tag{2.7}$$

Dabei ist die Berechnung von d_i im Rotationsmodus wie folgt:

$$d_i = \begin{cases} -1 & z_i < 0 \\ 1 & \text{otherwise} \end{cases} \tag{2.8}$$

Formeln 2.5 bis 4.1 zeigen schön den iterativen Ablauf des Algorithmus auf. Um nun einen Sinuswert zu berechnen sind folgende Initialwerte notwendig:

$$\begin{aligned} x_0 &= 1 \\ y_0 &= 0 \\ z_0 &= \varphi \end{aligned} \tag{2.9}$$

φ ist der gegebene Winkel, welcher zwischen $-\pi/2$ und $\pi/2$ sein muss, damit der Algorithmus konvergiert.

Daraus ergeben sich nach n Iterationen der Sinus und Kosinus Wert wie folgt:

$$\begin{aligned} x_n &= \frac{\cos\varphi}{A} \\ y_n &= \frac{\sin\varphi}{A} \end{aligned} \tag{2.10}$$

Schlussendlich ist es notwendig die Resultate um den Faktor $A = 0.60725294$ zu korrigieren um die richtigen Werte zu erhalten.

2.4 CIC Filter

Ein CIC-Filter oder Cascaded-Integrator-Comb-Filter ist ein digitales Filter, welches nebst der Filterung eines Signals zusätzlich deren Abtastfrequenz verändert. Abbildung 2.2 zeigt ein Decimation-CIC-Filter. Dieses verkleinert die Abtastfrequenz am Ausgang um den Faktor R. Die zweite Form ist ein Interpolation-CIC-Filter. Dieses vergrössert die Abtastfrequenz um den Faktor R. Jedoch wird hier nur auf das Decimation-CIC-Filter eingegangen.

In Abbildung 2.2 ist zu sehen, dass das Filter in drei Stufen unterteilt ist. Links sieht man ein Integrator-Filter, welches einen anliegenden Wert mit einem verzögerten Wert aufaddiert oder einfach das Signal integriert. Anschliessend folgt ein Dezimierer, welcher das Signal um den Faktor R unter-abtastet. Zuletzt folgt ein Combfilter. Dieser nimmt den aktuellen Wert und subtrahiert den alten Wert. Durch Tauschen von Integrator-Filtern mit Comb-Filtern und umgekehrt kann der zuvor genannte Interpolation-CIC-Filter gebaut werden. Es ist möglich mehrere Integrator- und Combfilterpaare zusammen zu schalten um eine grössere Dämpfung höherer Frequenzen zu bewirken. Das CIC-Filter benötigt jedoch um zu funktionieren eine gewisse Anzahl Bits in den Speichern der Verzögerungselemente, welche grösser als die Eingangsbits ist. Dieser Effekt nennt sich Bit-Growth und die zusätzliche Anzahl Bits lässt sich wie folgt berechnen[9]**cic_b**:

$$B_+ = \lceil N log_2 RM \rceil \tag{2.11}$$

Dabei entspricht N der Ordnung des Filters oder wie viele Integrator-Comb-Filterpaare das Filter hat. R ist der zuvor genannte Dezimationsfaktor und M ist die Verzögerung der Speicherelemente.

Das CIC-Filter verstärkt das Eingangssignal zudem um einen Faktor G. Dieser lässt sich wie folgt berechnen:

$$G = (R \cdot M)^N \tag{2.12}$$

Um nun den Faktor zu berechnen, um welchen man den Ausgang eines CIC-Filter multiplizieren muss um den Zahlenbereich voll auszunutzen, kann folgende Formel genutzt werden:

$$G_+ = G/2^{B_+} \tag{2.13}$$

Ein Problem, welches die CIC-Filter mit sich bringen, ist dass sie in bestimmten Situationen Aliasing erzeugen. Dieses entsteht wenn sich Signalkomponenten zu nahe an den Nullstellen des Filters befinden. In Abbildung 2.3 sieht man sehr schön, dass sehr auf die Frequenz des Signals im Vergleich zum verwendeten CIC-Filter geachtet werden muss.

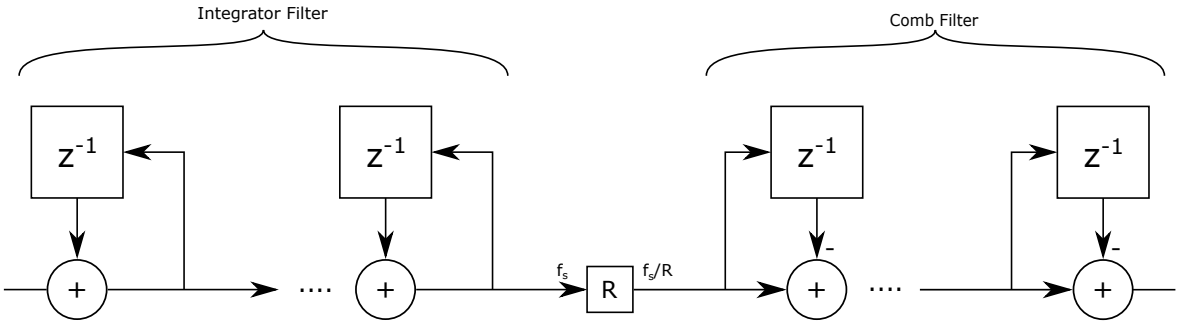


Abbildung 2.2: Aufbau eines CIC-Filters N-ter Ordnung mit Eingang mix_out und Ausgang filt_out

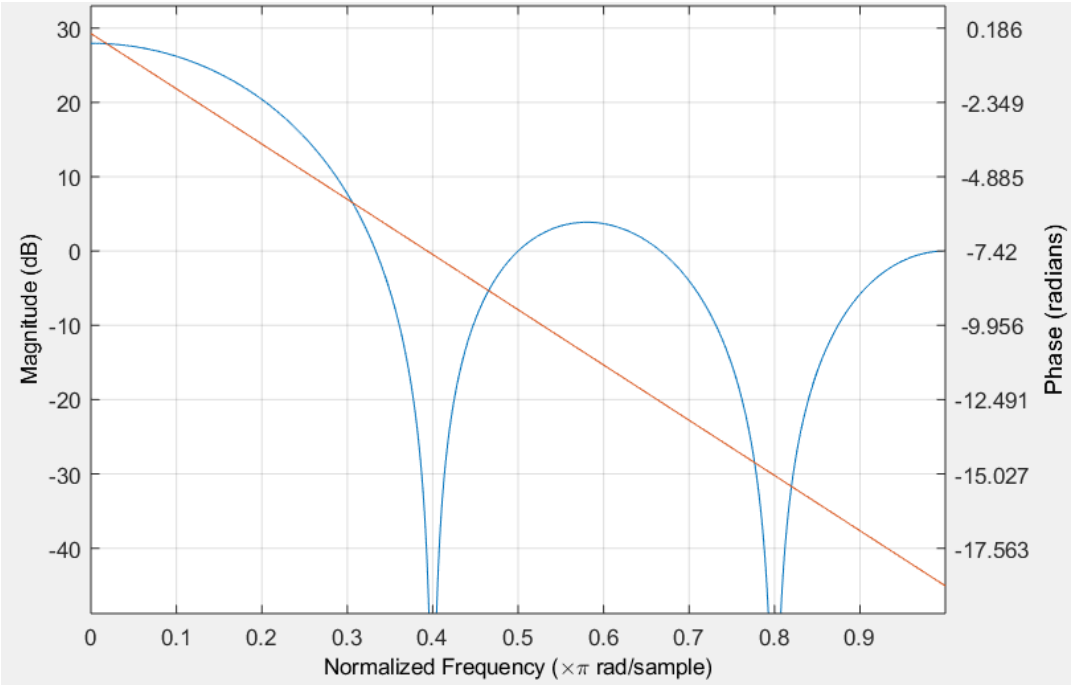


Abbildung 2.3: Amplitudengang und Phasengang eines CIC-Filters (M = 1, R = 5; N = 2)

2.5 Goldschmidt Algorithmus

Um in FPGAs dividieren zu können ist es nötig selber eine solche Operation zu implementieren. Dafür gibt es für verschiedene Anforderungen diverse Algorithmen. Einer dieser ist der Goldschmidt Algorithmus. Er ermöglicht es iterativ eine Division zweier Zahlen durchzuführen welche

als Resultat auch Nachkommazahlen enthält. Für die Berechnung multipliziert der Algrithmus den Nenner und Zähler wie in Formel 2.14 iterativ mit den Faktoren F_n.

$$Q = \frac{Z}{N}\frac{F_1}{F_1}\frac{F_2}{F_2}\frac{F_3}{F_3}\frac{F_{...}}{F_{...}} \tag{2.14}$$

Offensichtlich verändert dies nicht das Verhältnis des Zählers und Nenners. Für die Berechnung einer Iteration ergeben sich folgende Formeln:

$$F_{i+1} = 2 - N_i \tag{2.15}$$

$$Z_{i+1} = F_{i+1} \cdot Z_i \tag{2.16}$$

$$N_{i+1} = N_{i+1} \cdot Z_i \tag{2.17}$$

Damit der Algorithmus richtig funktioniert muss ist eine Skalierung des Zählers und Nenners notwendig. Dies, da die Werte nur konvergieren, wenn der Nenner zwischen 0 und 1 ist. Will man beispielsweise 2 durch 3 teilen ist vorgängig eine Skalierung auf 0.5 respektive 0.75 notwendig. Dies ist in Hardware durch eine einfache Schiebung nach rechts zu bewerkstelligen.

3 Konzept

Der Aufbau des digitalen Theremin ist sehr ähnlich wie das des Analogen, mit einigen Änderungen um es besser digital aufzubauen. Abbildung 3.1 zeigt, dass der Lautstärke- und Tonhöhenoszillator nicht mehr einen Sinus sondern einen Rechteck generieren. Wir haben uns deshalb für diese Änderung entschieden, da es so einfacher ist das Signal in das FPGA einzulesen. Dies da kein Analog-Digital-Wandler nötig ist. Da der Referenzoszillator weiterhin ein Sinus ist, ergibt die Mischung mit dem Rechteck auch Mischprodukte mit dessen Oberwellen. Da diese aber eine höhere Frequenz haben, können diese später weggefiltert werden.
Weiter sind die Referenzoszillatoren neu digital. Um nun einen Sinus zu generieren, haben wir uns entschieden den in Kapitel 2.3 behandelten Cordic Algorithmus zu verwenden. Dieser ist besser um verschiedene Frequenzen zu generieren als eine einfache Lookup-Table und bietet einen grösseren Lerngewinn. Diese Komponente stammt aus dem Projekt 5.
Der Mischer multipliziert die den Sinus des Referenzoszillators mit dem Rechteck des analogen Oszillators.
Für das Tiefpassfilter haben wir uns entschieden mehrere CIC-Filter und ein FIR-Filter einzusetzen. Das CIC-Filter stammt ebenfalls aus dem Projekt 5. CIC-Filter haben den Vorteil, dass sie Ressourcensparender sind als äquivalente FIR-Filter.
Wie man sieht ist die Signalverarbeitung für den Lautstärketeil bei diesem Aufbau gleich wie der Tonhöhen Teil. Dies haben wir so entschieden, um dieselben Komponenten nochmals nutzen zu können.
Um das Audiosignal zu verstärken, wird die Höhe der Frequenz des Lautstärketeils benötigt. Diese wird durch den Block Frequenzmessung gemacht und an den Verstärker weitergegeben. Da die Frequenz des Lautstärkeoszillators bei Veränderung der Distanz zu der Antenne logarithmisch ändert, ist keinerlei Umrechnung nötig um eine entsprechende Lautstärkeänderung zu erzielen. Anschliessend konvertiert der Digital-Analog-Wandler das verstärkte Audiosignal und gibt es am Lautsprecher aus.
Wir entschieden zudem ein Nios System einzusetzen um das Theremin zu bedienen und zu steuern. Dies hauptsächlich, um einen Einblick in den Nios zu gewinnen. Für die Interagierung mit dem Theremin entschieden wir uns für ein Touch Display. Der Bedienungs & Steuerungs Block (Nios System) wurde in Abbildung 3.1 nicht mit anderen Komponenten verbunden um die Zeichnung übersichtlicher zu gestalten.

Über die Steuerung soll zudem eine automatische Kalibration des Theremins möglich sein. Schlussendlich soll nämlich wenn der Spieler sich den Antennen nähert die Tonhöhe grösser und die Lautstärke lauter werden. Aus diesem Grund müssen die Referenzoszillatoren auf die analogen Oszillatoren abgestimmt werden.

Es soll zudem möglich sein den in Kapitel 1 erwähnten Glissando-Effekt zu aktivieren und auf dem Display die Spielgenauigkeit anzuzeigen. Diese beiden Features werden über den Nios und das Display gesteuert. Die Übergangszeit des Glissando-Effekt soll zudem Einstellbar sein und es soll nebst der normalen Tonleiter auch die pentatonische Tonleiter spielbar sein.

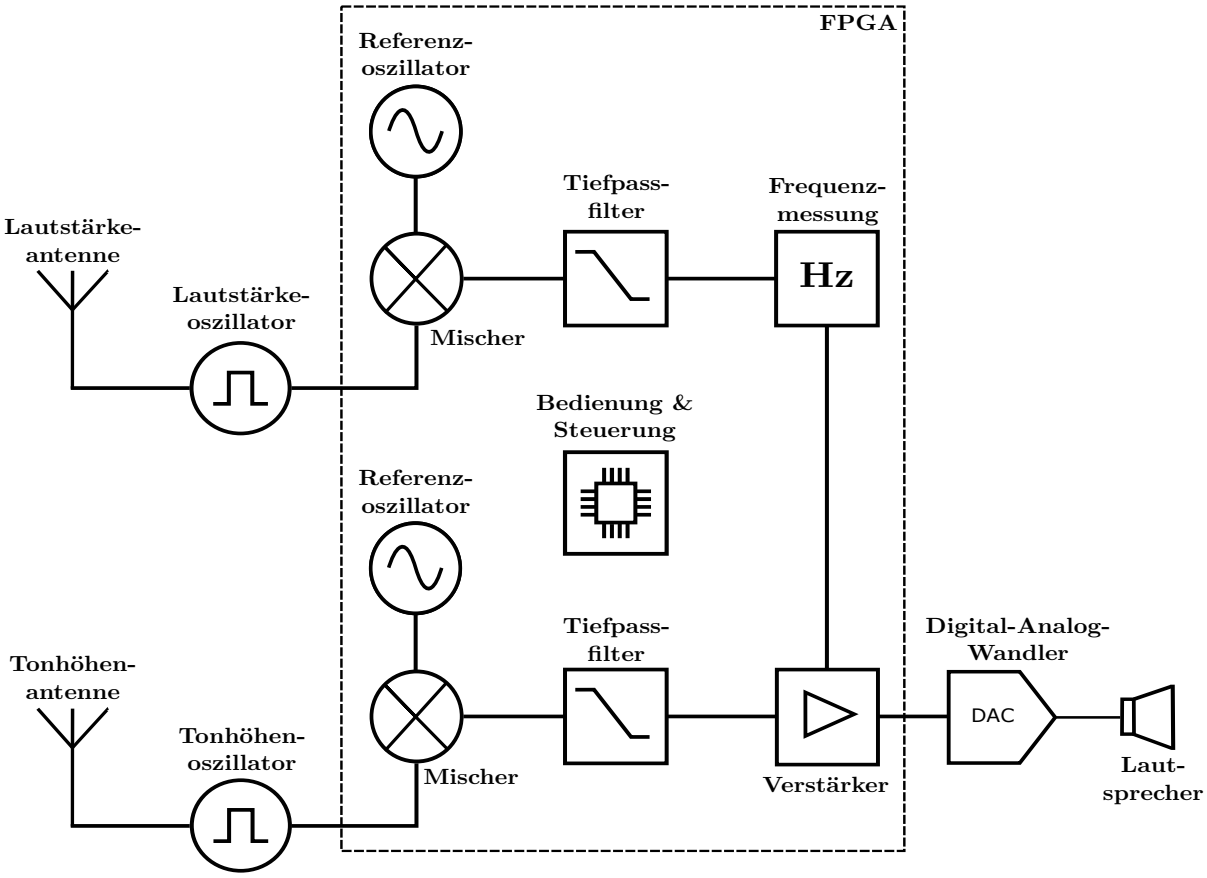


Abbildung 3.1: Blockschaltbild des digitalen Theremins

4 Realisierung

Das digitale Theremin ist auf dem Entwicklungsboard DE1-SOC von terasIC aufgebaut. Dieses enthält ein Cyclone V 5CSEMA5 FPGA von Intel. Weiter befindet sich auf dem Board der Audio Codec WM8731 von Wolfson für die Ausgabe an einem Lautsprecher. In Abbildung 4.1 ist der Aufbau des Digitalen Theremin aufgezeigt inklusive der Peripherie ausserhalb des FPGA.
Das Theremin, welches im FPGA aufgebaut ist, besteht aus zwei Bereichen. Einerseits der Signalverarbeitung und Übermittlung an den Codec. Dieser besteht aus den Komponenten *Volume* und *Pitch Generation*, *DC-FIFO* und dem *Audio Serializer*. Der zweite Bereich ist Das Nios System. Dieses besteht aus dem Prozessor und diversen IP Cores, welche die Kommunikation mit den Peripherien ermöglicht. Ausserhalb des FPGA ist zudem das entwickelte PCB, welches die beiden Antennenoszillatoren enthält, mit welchen das Theremin gespielt werden kann.
Die Kommunikation zwischen dem Nios Prozessor und den anderen Komponenten geschieht über das *Avalon Memory Mapped Interface*. Der Prozessor ist in dieser Kommunikation Master und die restlichen Komponenten Slaves. Die Übertragung der Audioinformation in der Signalverarbeitung geschieht über das *Avalon Streaming Interface*. Wobei Sender als Streaming Source und Empfänger als Streaming Sink deklariert sind. Das Streaming Interface ist notwendig für den Einsatz des Dual-Clock-FIFO (DC-FIFO). Dieses übernimmt den Übergang verschiedener Clockregionen zwischen den Komponenten *Pitch Generation* und *Audio Serializer*.
Die Clocks, welche zu den verschiedenen Komponenten gehen sind in Abbildung 4.1 für eine bessere Überschaulichkeit weggelassen worden. Für eine Liste aller Clock Frequenzen und deren Ziel siehe Kapitel 4.2.

4.1 Antennenoszillator

Für die Antennenoszillator Schaltung übernahmen wir die bereits im Projekt 5 eingesetzte Colpitts-Oszillator Schaltung. Die aus dem Projekt 5 übernommene Schaltung ist in Abbildung 4.2 gezeigt.

Es handelt sich dabei um eine Colpitts-Oszillator Schaltung mit einem JFET. Diese Schaltung ist von dem Bauset "Theremin selber bauen"von Franzis übernommen. Da der im Bauset verwendete JFET nicht mehr bestellbar ist, verwendeten wir den J113 N-Channel JFET. Die mit LTspice simulierten Werte des J113 glichen stark der original Schaltung, weshalb dieser gewählt wurde. Damit das Sinussignal des Antennenoszillator nicht A/D gewandelt werden muss, haben wir entschieden das Sinussignal in ein Rechtecksignal mit gleicher Frequenz zu wandeln. Dies geschieht mithilfe einer Komparatorschaltung. Die Komparatorschaltung wird mit 3.3 V betrieben da die Logikeingänge des FPGA auf diese Spannung ausgelegt sind. Als Referenzspannung der Komperatorschaltung dient die halbe Versorgungsspannung, welche mit den Widerständen R7 und R8 gebildet wird.

Als Antenne wird ein Messingrohr verwendet. Dieses ist zwischen der Spule L1 und dem Kondensator C5 verbunden.

Die Ausgangsspannung des Colpitts-Oszillator ist über den Kondensator C11 entkoppelt. Dies entfernt den DC-Anteil. Der Kondensator C11 und die Widerstände R3 und R4 bilden zusammen einen Hochpass. Damit die Oszillator Frequenz von ca 562 kHz das Filter passieren kann ist C11 so gewählt das die Grenzfrequenz des Filters bei ca 265 kHz liegt.

Der Antennenoszillator im Projekt 6 verwendet für die Lautstärke und die Tonhöhenantenne jeweils eine Colpitts-Oszillator Schaltung wie in Abbildung 4.2 gezeigt. Der Antennenoszillator ist mit einem 12 VDC Schaltnetzteil gespiessen. Der MC7809 Spannungsregler generiert die 9 VDC für die Colpitts-Oszillator Schaltungen. Die 3.3 VDC für den Komperator erzeugt der

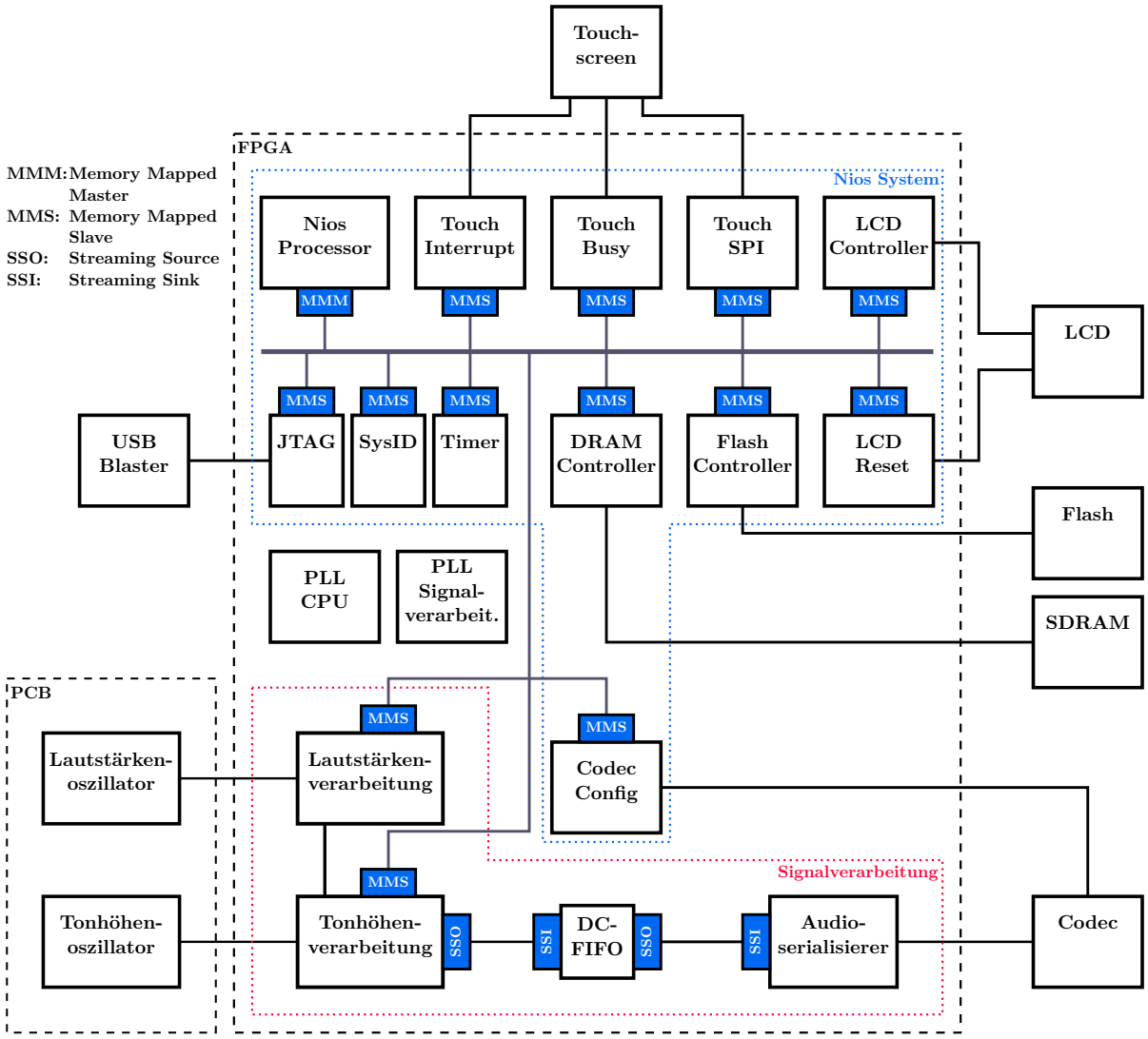


Abbildung 4.1: Blockschaltbild gesammtes Theremin

LT1117 Spannungsregler. Bei der Wahl der Spannungsregel ist darauf geachtet worden das die erzeugten Spannungen möglichst Störungsfrei und wenig Rippel aufwiesen. Das gesamte Schema der Schaltung ist im Anhang enthalten.

4.2 Clock

Die verschiednen Clocks für die Hardwarekomponenten und die CPU werden in zwei Phase-Locked-Loop (PLL) Blöcken generiert. Ein Block für die Signalverarbeitung und einer für das Nios System. In Tabelle 7.4 sind alle Frequenzen aufgelistet.
Alle Frequenzen welche nicht 50MHz sind ergaben sich daraus, dass die externen Peripherien, welche mit den entsprechenden IP Cores verbunden sind, diese Frequenzen als Vorgabe haben. Weiter benötigen die Komponenten Pitch- und Volume Generation die Frequenz 54Mhz, da deren Frequenz ein vielfaches von 48kHz sein muss.

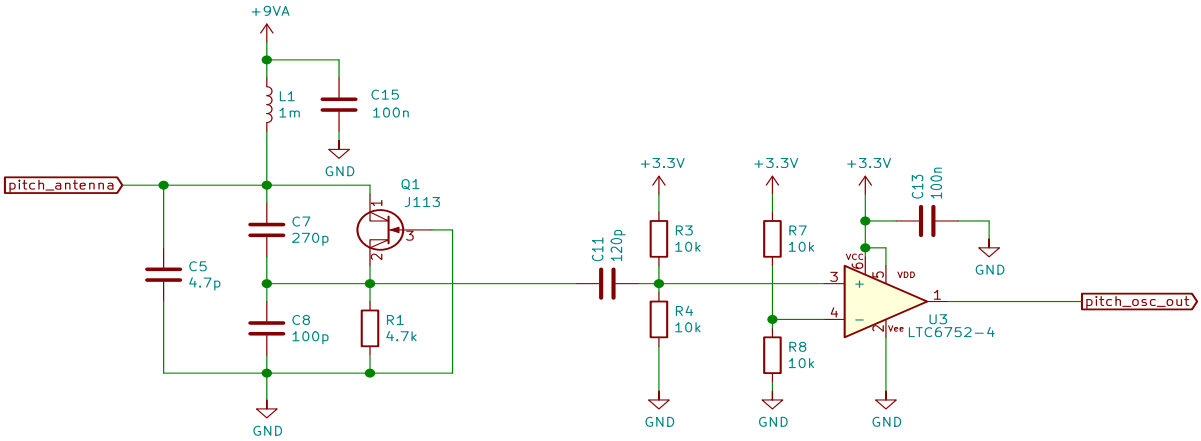


Abbildung 4.2: Schema Antennenoszillator. Links Collpitts-Oszillator,rechts Komparatorschaltung

Tabelle 4.1: Clockfrequenzen der verschiedenen Komponenten

Komponente	Frequenz	PLL Core
Nios Processor	50 MHz	PLL CPU
JTAG Controller	50 MHz	PLL CPU
Timer	50 MHz	PLL CPU
SysID	50 MHz	PLL CPU
DRAM Controller	50 MHz	PLL CPU
SDRAM	50 MHz	PLL CPU
LCD Controller	15 MHz	PLL CPU
LCD Reset	15 MHz	PLL CPU
Touch Interrupt	15 MHz	PLL CPU
Touch Busy	15 MHz	PLL CPU
Touch SPI	15 MHz	PLL CPU
Audio Config	12 MHz	PLL CPU
Flash Controller	25 MHz	PLL CPU
Pitch Generation	54 MHz	PLL Signal-Processing
Volume Generation	54 MHz	PLL Signal-Processing
DC-FIFO Input	54 MHz	PLL Signal-Processing
DC-FIFO Output	24 MHz	PLL Signal-Processing
Audio Serializer	24 MHz	PLL Signal-Processing
Codec	12 MHz	PLL Signal-Processing

4.3 CPU

Der eingesetzte Nios Prozessor ist für die Bedienung des Theremin und die Steuerung der Signalverarbeitungshardware zuständig. Die diversen eingesetzten IP Cores sind in den unten stehenden Kapiteln beschrieben.

JTAG, Timer und System ID

Der JTAG IP Core ermöglicht das flüchtige Programmieren des Nios wie auch das Kommunizieren mit selbem für Debugging Zwecke. Durch den Einsatz des Timer IP Cores erhält der

Nios einen Interval Timer um beispielsweise periodisch Interrupts zu generieren. In dem System ID IP Core ist die Systemidentifikationsnummer gespeichert. Diese wird benötigt um beim laden der Software sicherzustellen, dass das passende Hardware Image vorhanden ist. Alle drei Komponenten sind mit Standardeinstellungen in das Nios System eingefügt worden.

Speicher

Der Arbeitsspeicher ist ein externer 64MB SDRAM Chip IS42S16320D von ISSI. Für die Kommunikation mit dem Nios Prozessor ist der SDRAM Controller IP Core zuständig. Der Nios Prozessor kann über das Memory Mapped Interface mit dem Core Kommunizieren und so auf das SDRAM zugreifen. Da dieser Chip auf dem Entwicklungsboard sowieso vorhanden ist, haben wir uns gegen Onchip Speicher entschieden um Ressourcen zu sparen.

Das Hardware Image und der Programmcode ist auf dem Board enthaltenen Flash Speicher gespeichert. Dabei wird anders als bei dem nicht flüchtigen Programmieren nicht das SRAM Object File (.sof) geladen sondern ein JTAG Indirect Configuration File (.jic). Dieses kann in Quartus aus dem SRAM Object File und dem in Eclipse generierten HEX File erstellt werden. Anschliessend kann es über ein Serial Flash Loader Image über den USB Blaster auf den Flash geladen werden. Beim Einschalten des Gerätes wird zuerst das Hardwareimage ins FPGA geladen und anschliessend der Programmcode geholt. Auf Empfehlung von Dokumentationen von Intel haben wir uns dafür entschieden den Programmcode durch einen Bootcopier ins SDRAM zu kopieren. Abbildung 4.3 zeigt das Layout des Flash Speichers nach dem Programmieren. [10]

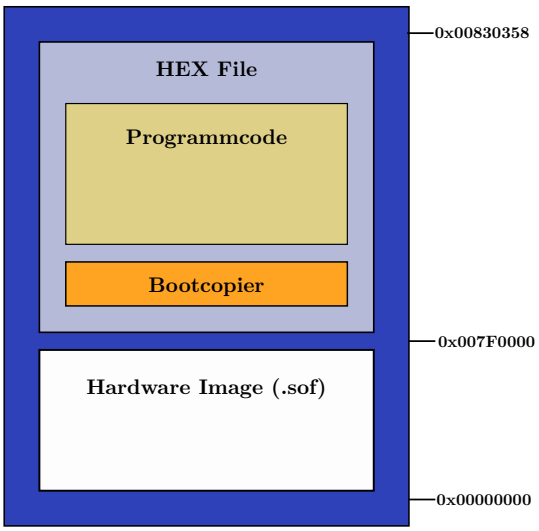


Abbildung 4.3: Layout des Flash Speichers

LCD Controller & Reset

Für das beschreiben des LCD ist die von terasIC bereitgestellte VHDL Komponente LT24_Controller zuständig. Dieser kann über das Memory Mapped Interface von dem Nios Prozessor gesteuert werden. Das verwendete Display LT24 von terasIC enthält für das Schreiben des LCD den LCD Treiber ILI9341 von ILITEK. Dieser Chip wird durch den LT24_Controller über das parallele

16 Bit Interface gesteuert. Weiter kann der LCD Chip über den PIO Core LCD Reset zurückgesetzt werden. Wie diese beiden Komponenten in Software angesteuert werden ist in Kapitel **??** genauer beschrieben. [11]

Touchscreen
Der Touch Screen Digitizer AD7843 von Analog Devices misst den resistiven Touchscreen des LCD aus und übermittelt die digitalisierten Koordinaten über SPI an den Prozessor. Der Nios Kommuniziert dabei über drei verschiedene IP Cores mit diesem Chip. Der SPI Core *Touch SPI* für die Datenübertragung, der PIO Core *Touch Busy* um den Beschäftigungsstatus des Chips zu wissen und den zweiten PIO Core *Touch Interrupt*, welcher den Nios über eine Betätigung des Touchscreens informert. Bei einer Berührung des Touchscreens löst *Touch Interrupt* beim Nios Prozessor einen Interrupt aus, welcher sofort die Koordinaten über SPI anfordert. [12]

4.4 Tonhöhenverarbeitung

Die Hauptaufgabe der Komponente *Tonhöhenverarbeitung* ist es das Audiosignal aus dem Rechtecksignal des Tonhöhenoszillators zu generieren. Die *Tonhöhenverarbeitung* nimmt zudem eine Frequenzmessung des Audiosignals vor um diverse Funktionalitäten zu gewährleisten. In Abbildung 4.4 ist der Grobe aufbau der Komponente aufgezeigt. Die genaue Erklärung zu den einzelnen Komponenten ist in den folgenden Abschnitten zu finden.

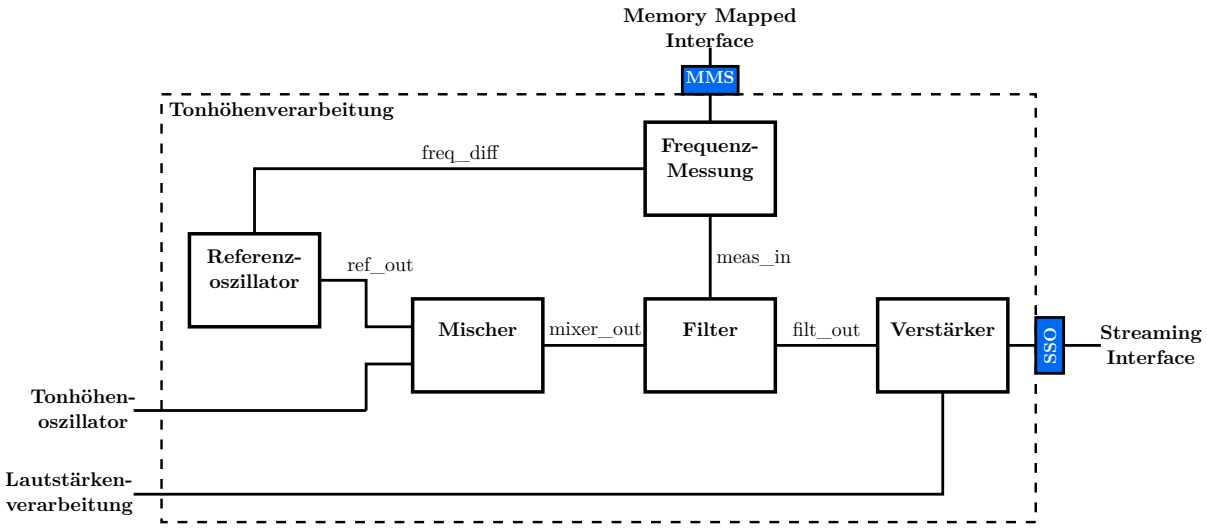


Abbildung 4.4: Blockschaltbild der Custom IP *Tonhöhenverarbeitung*

Referenzoszillator

Der *Referenzoszillator* ist wie in der analogen Version aus Kapitel 2.1 dafür zuständig ein Sinussignal mit einer Frequenz nahe der des Tonhöhenoszillator zu generieren und an *ref_out*

auszugeben. Er generiert diesen wie schon erwähnt mithilfe des Cordic Algorithmus. Er ist aufgeteilt in zwei Komponenten: der *Cordic Prozessor* und der *Cordic Controller*. Wie diese beiden Komponenten miteinander verbunden sind ist in Abbildung 4.5 ersichtlich. Beide Komponenten stammen aus dem Projekt 5. Änderungen, welche im Projekt 6 stattfanden, sind entsprechend gekennzeichnet.

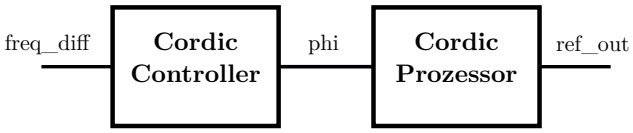


Abbildung 4.5: Aufbau des Referenzoszillators

Der *Cordic Prozessor* ist die eigentliche Implementierung des Cordic Algorithmus wie er in Kapitel 2.3 beschrieben wurde. Er muss jedoch für den Einsatz im FPGA leicht angepasst werden. Die Berechnung der Werte für $\arctan 2^{-i}$ konnte vorgängig gemacht werden und ist in eine Lookup Table gespeichert. Dies spart Ressourcen für diese komplizierte Berechnung ein. Wir haben uns entschieden den Algorithmus in einer Pipeline zu implementieren. Dies führt einerseits zu einer höheren maximalen Clockfrequenz, andererseits aber auch zu einer grösseren Signallatenz. Dies ist jedoch nicht problematisch für die gewählte Anwendung, da die Latenz im Nanosekundenbereich ist und später nicht hörbar auffällt. Zuletzt ist eine Multiplikation mit 2^{-i} ganz einfach durch eine Verschiebung um i Bits nach rechts ersetzbar. Dies spart wiederum Ressourcen ein.
Das berechnete Resultat ist als signed Zahl definiert um die Berechnung von negativen Zahlen zu ermöglichen. Sie sind im fixed-point Format und haben 16 Bit Länge. Dabei sind von den 16 Bit 1 Bit Vorzeichen und 15 Bit Nachkommastellen. Dies entspricht einem Zahlenbereich von -1 bis 0.999969. Die berechneten Werte gibt der *Cordic Prozessor* an *ref_out* aus.

Für die Berechnung der Sinuswerte muss ein Winkelwert berechnet werden, der mit der Zeit so ändert, dass sich am Ausgang des Cordic Prozessor ein Sinus mit der gewünschten Frequenz ergibt. Der berechnete Winkelwert wird an *phi* ausgegeben. Für diese Aufgabe ist der *Cordic Controller* zuständig. Bei mit der Zeit linear ansteigendem Winkelwert ergibt sich am Ausgang die gewünschte Sinusform. Wichtig ist jedoch, dass der Cordic Algorithmus nur für Winkelwerte zwischen $-\pi/2$ und $\pi/2$ oder anders für Werte im ersten und zweiten Quadranten konvergiert. Die Lösung für dieses Problem ist in Abbildung 4.6 ersichtlich. Als erstes wird der Sägezahn Winkel berechnet. Für den linearen Anstieg des Winkels zählt der *Cordic Controller* einen Zähler mit einer bestimmten Schrittweite jeden Clockzyklus hoch. Der wrap-around des Zählers ist dabei sogar erwünscht um den Sprung zwischen dem II und III Quadranten zu erzielen. Der Schrittwert ergibt sich wie folgt:

$$step = \frac{2^{n+1} f_{sig}}{f_{clk}} \tag{4.1}$$

Wobei n die Anzahl Bits des Wertebereichs des Dreiecks Winkels ist, f_{sig} die gewünschte Frequenz des generierten Signals und f_{clk} die Clock Frequenz des FPGA.

Nun kommt die bereits erwähnte Einschränkung des Cordic Algorithmus ins Spiel. Die berechneten Werte des Sägezahnwinkels zwischen dem II und III Quadranten konvergieren nicht. Aus diesem Grund konvergiert der *Cordic Controller* diesen Winkel in den Dreieckswinkel. Sind die beiden vordersten Bits entweder 01 oder 10 befindet sich der Winkelwert im II respektive III Quadranten. Um in diesem Fall den Dreieckswinkel zu erhalten invertiert der *Cordic Controller*

alle Bits ausser dem most-significant Bit. Wie man sich leicht davon überzeugen kann ergibt der Dreieckswinkel denselben Sinusverlauf wie der Sägezahnwinkel bei einer Sinusrechnung ohne die erwähnten Einschränkungen. [8]

Um die Kalibrierung und den Glissandoeffekt für das Theremin zu ermöglichen waren im Projekt 6 kleine Anpassungen am Cordic Controller nötig. Wie zuvor hat der Controller eine fixe Frequenz implementiert, welche in der Grössenordnung 550 kHz liegt. Jedoch legt die Frequenzmessungskomponente nun eine Differenz über einen Eingang an den Cordic Controller an um die zuvor genannten Features über den Referenzoszillator zu ermöglichen.

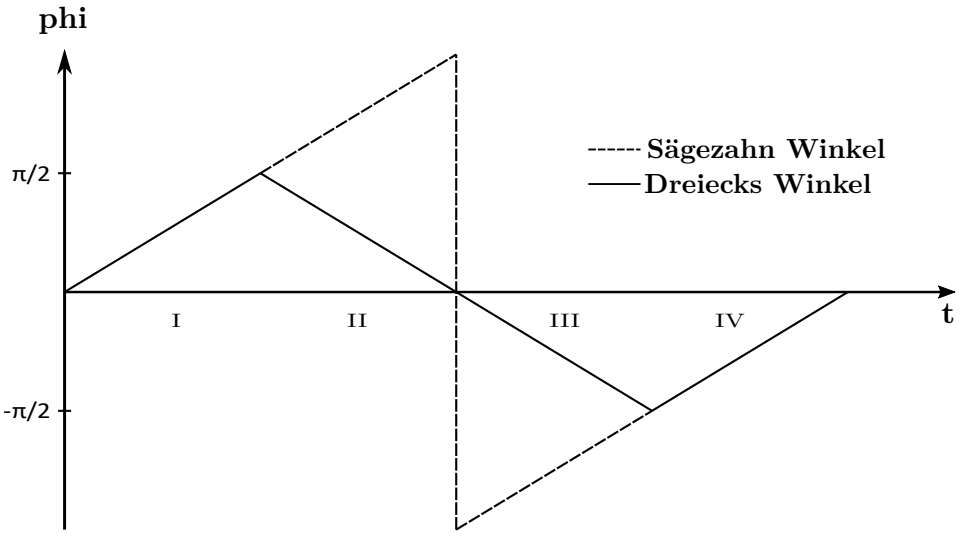


Abbildung 4.6: berechneter Winkel *phi* des Cordic Controllers in Funktion der Zeit

Mischer

Die Implementation des *Mischers* ist dank der Entscheidung für die Rechteckform des Tonhöhenoszillatorsignals sehr einfach. Über einen GPIO liest der Mischer das Signal des Tonhöhenoszillators ein und verrechnet es mit dem generierten Sinus *ref_out*. Eine 1 des Rechtecks wird dabei als die Zahl 1 und eine 0 als die Zahl -1 interpretiert. Die Multiplikation zwischen der 1 und *ref_out* ist dabei nicht nötig und eine Multiplikation mit -1 erzielt der Mischer durch das Bilden des Zweierkomplement von *ref_out*.

Filter

Im Projekt 5 war das *Filter* ein einzelnes CIC-Filter mit dem Dezimationsfaktor 1000. Dies hatte zur Folge, dass viel Aliasing entstand. Um dieses zu verringern haben wir uns im Projekt 6 für den Aufbau aus Abbildung 4.7 entschieden. Die drei Filter CIC 1 bis CIC 3 sind Instanzen einer CIC-Filter Komponente. Diese Komponente ist unverändert geblieben seit Projekt 5. Die Parameter der drei Instanzen sind in Tabelle 4.2 ersichtlich. Da es bei CIC-Filtern wie in Kapitel 2.4 beschrieben, um deren Nullstellen zu Aliasing kommt, sind diese Filter so eingestellt, dass sich die Oberwellen des Rechteck möglichst nicht in deren Nähe befinden. Bei *CIC 1* wurde eine höhere Ordnung gewählt um am Anfang eine stärkere Dämpfung zu erzielen. Die Anzahl Ausgangsbits wurde nach Formel 2.11 in Kapitel 2.4 berechnet.
Zuletzt haben wir noch ein FIR-Filter implementiert um das Signal auf 48kHz unter abzutasten. Das Filter hat eine Passfrequenz von 2 kHz und eine Stopfrequenz von 24 kHz mit einer Dämpfung von 55dB. Wir entschieden uns die Koeffizienten mit dem *filterDesigner Tool* von Matlab

zu berechnen und als 27 Bit signed Zahlen in einer Lookup Table zu speichern. Wir wählten deshalb 27 Bit, da im FPGA eine solche Multiplikation noch knapp in einen DSP Block integriert werden kann. [13] Weswegen der Ausgang Frequenzmessung nach dem zweiten CIC-Filter gewählt wurde ist im nächsten Abschnitt beschrieben.

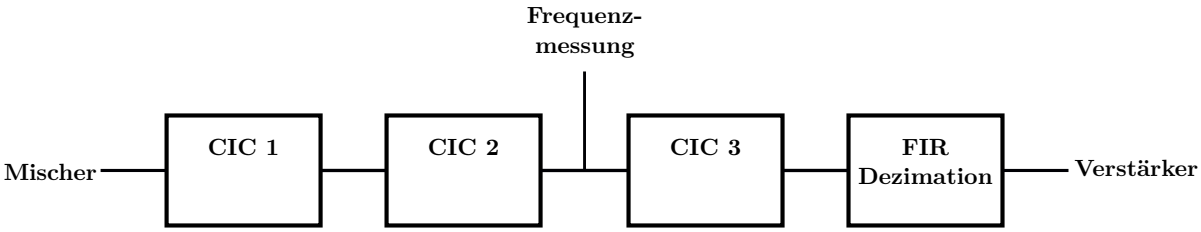


Abbildung 4.7: Aufbau des Filters in der Komponente Pitch Generation

Tabelle 4.2: Parameter der drei CIC-Filter

Komponente	Dezim.Fakt.	Ordnung	Ausgangsfreq.	Ausgangsbits
CIC 1	5	2	10.8 MHz	21 Bits
CIC 2	9	1	12 MHz	25 Bits
CIC 3	5	1	240 kHz	28 Bits

Verstärker

Die Komponente *Verstärker* ist dafür zuständig beim Signal *filt_out* den Gain der CIC-Filter zu kompensieren und anschliessend dieses mit der Dämpfung, welche die Lautstärkeverarbeitung liefert, zu multiplizieren. In Abbildung 4.8 ist eine Problematik aufgezeigt, welche auftritt, wenn man die Dämpfung zu beliebigen Zeiten wechselt. Links sieht man, was passiert, wenn die Dämpfung beim höchsten Wert der positiven Halbwelle ändert. Diese Sprünge im Signal treten dann als hörbares Knacksen auf. Um dies zu verhindern ist eine Erkennung von Nulldurchgängen implementiert, dass wie in der Abbildung rechts die Dämpfungen nur bei diesen Nullstellen geändert werden. Da der Codec ein Offsetbehaftetes Signal verlangt muss das most-significant Bit des Signal getoggelt werden um dies zu bewerkstelligen. Diese Komponente enthält zudem die Kommunikation mit dem Streaming Interface.
Die Dämpfung des Audiosignals könnte auch über den Codec gemacht werden. Weshalb dies nicht möglich ist, ist in Kapitel 5.3 beschrieben.

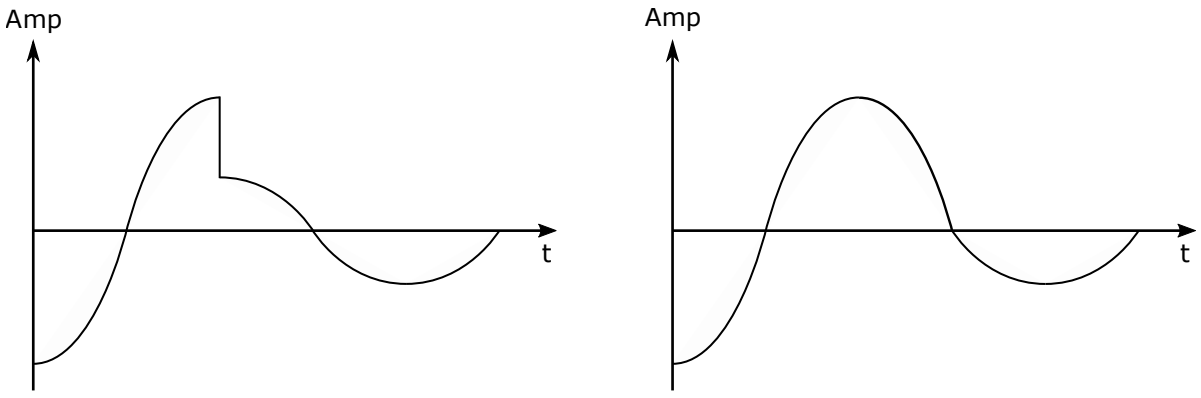


Abbildung 4.8: Unterschied Dämpfungswechsel (links ohne und rechts mit Nullstellenerkennung)

Frequenzmessung, Kalibration & Glissandoeffekt

Die Komponente *Frequenzmessung* hat mehrere Aufgaben und ist diejenige Komponente mit welcher über den Nios Prozessor die gesammte *Tonhöhenverabreitung* gesteuert werden kann. Der Aufbau dieser Komponente ist in Abbildung 4.9 aufgezeigt.
Zum einen wird hier die Frequenzmessung durchgeführt. Dies geschieht über die drei Komponenten FIR, Periodenzähler und Goldschmidtdividierer. FIR ist wie der Name sagt ein FIR Filter. Dieses ist nötig um das Signal aus dem Filter, welches noch Hochfrequente Signale enthält zu Filtern. Das Filter hat eine Passfrequenz von 2 kHz eine Stopfrequenz von 40 kHz und eine Dämpfung von 30dB. Das Filter wurde mit dem *filterDesigner* Tool in Matlab berechnet. Wir haben entschieden die Filterkoeffizienten als fixed-point signed Zahlen in einem Array mit 18 bit länge abzuspeichern. Auf dieser Koeffizientenlänge kann Quartus die DSP-Blöcke so nutzen, dass nach der Multiplikation des Signals mit den Koeffizienten die Resultate gleich in den Blöcken addiert wird. Dies ermöglicht längere Multiplikationsketten. [13]
Anschliessend wird das Signal *fir_out* im *Periodenzähler* ausgemessen. Diese zählt von Nulldurchgang zu Nulldurchgang einen Zähler hoch. Bei einem Nulldurchgang wird der Wert dieses Zählers am Signal *per_cnt* ausgegeben. Der Zählerwert entspricht der Anzahl Abtastwerte des Signals in einer Signalperiode.
Das Signal hat eine Abtastfrequenz von 1.2 MHz. Dividiert man diese Abtastfrequenz durch die zuvor gezählte Anzahl Abtastperioden erhält man die Frequenz des Signals.
Um diese Division zu berechnen haben wir uns entschieden den Goldschmidt Algorithmus aus Kapitel 2.5 einzusetzen. Dieser hat den Vorteil, dass er auch Nachkommastellen berechnen kann um die nötige Genauigkeit bei den tiefen Frequenzen zu erreichen wie in Kapitel 2.2 beschrieben. Dass die Berechnungsdauer des Algorithmus nicht für alle Zahlen gleich ist, ist nicht Problematisch, da diese Zeiten im Nanosekundenbereich liegen und nicht hörbar sind. Der Messbereich der Frequenzmessung fängt bei 100 Hz an und geht bis 10 kHz. Alle Frequenzen darunter oder darüber zeigen 100 Hz respektive 10 kHz an.

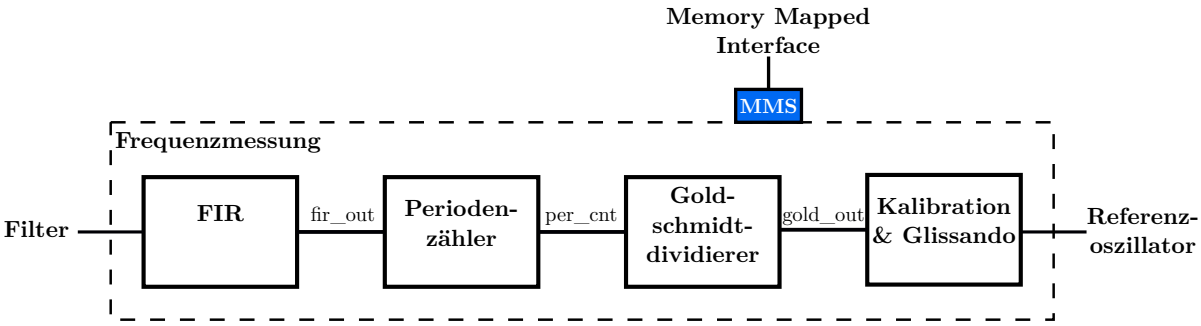


Abbildung 4.9: Aufbau der Frequenzmessung, Kalibration und Glissandoeffekt in der Komponente Pitch Generation

Bild überarbeiten und namen an Signalen anbringen wie in Text

Die gemessene Frequenz wird anschliessend in der Komponente *Calibration & Glissando* benötigt. Diese ist dafür zuständig einerseits die Tonhöhe zu kalibrieren und andererseits den Glissando-Effekt zu steuern. Um die Frequenz des Audiosignals für diese Funktionen zu verändern wird der Referenzoszillator entsprechenden verstimmt. Um die Frequenz des Referenzoszillators einzustellen wird die Differenz welche den Glissandoeffekt hervorruft und die Änderung welche während der Kalibration berechnet wurde addiert und dem Referenzoszillator übergeben. Diese wird von nun an als Frequenzdifferenz bezeichnet. Die Steuerung dieser Funktionen ist als State-Machine aufgebaut wie in Abbildung 4.10 zu sehen ist. Dabei sind die States reset, check, sign und diff für die Kalibration zuständig und freq range, step und step count für den Glissando-Effekt zuständig. Es folgt eine Erklärung, was in den einzelnen Zuständen geschieht:

Idle: Der Idle State setzt die Frequenzdifferenz auf die berechnete Differenz der Kalibrierung. Der Anteil des Glissando-Effekts wird auf 0Hz gesetzt.

reset: Der alte Kalibrationswert wird gelöscht und gewartet bis eine neue Frequenzmessung durchgeführt wurde.

check: Da die Messung Frequenzen unter 100 Hz immer als 100 Hz angibt, müssen diese erkannt werden. Bei einer Messung von 100 Hz inkrementiert die Komponente Calibration & Glissando die Frequenzdifferenz um 400 Hz. Dies führt dazu, dass bei der nächsten Messung das Signal nicht mehr in dem Bereich liegt, in dem die Messung immer 100 Hz misst.

sign: Für die in Kapitel 3 beschriebene Spielweise, dass mit kleinerer Distanz zur Antenne die Tonhöhe steigt, ist der State sign zuständig. Ist der Referenzoszillator so eingestellt, dass dessen Frequenz kleiner ist als die des Tonhöhenoszillators, so würde bei Annäherung des Spielers an die Antenne die Frequenz des Audiosignals zuerst sinken, bis die beiden Oszillatorfrequenzen gleich sind. Anschliessend würde sie wieder steigen. Um zu erkennen ob dies der Fall ist, subtrahiert die Komponente 100 Hz von der Frequenzdifferenz. Ist die nachfolgende Messung grösser geworden, bedeutet dies, dass der Referenzoszillator kleiner war als der Tonhöhenoszillator. Nun kann ganz einfach die gemessene Frequenz verdoppelt werden und zu der Frequenzdifferenz addiert werden, damit der Referenzoszillator die grössere Frequenz hat.

diff: Nach den letzten drei States ist nun sichergestellt, dass der Referenzoszillator grösser ist als der Tonhöhenoszillator. Jedoch sollen ja diese beiden Oszillatoren aufeinander abgestimmt sein. Dazu wird abwechslungsweise die Frequenzdifferent um einen kleinen Schritt dekrementiert und danach die Frequenz gemessen. Wir haben uns dafür entschieden, dass wenn die Messung 120 Hz unterschreitet die Kalibration abgeschlossen ist. Dies da Töne unterhalb dieser Frequenz sehr merkwürdig klingen.

freq range: Der State freq range ist dafür zuständig herauszufinden, welcher diskrete Ton der gewählten Tonleiter (normal oder pentatonisch) am nächsten zur gemessenen Frequenz ist. Die Frequenz wird dabei mit einer Lookup-Table mit allen Grenzen zwischen den Tönen verglichen. ist die Frequenz ausserhalb des gewählten Frequenzbereich wird hier abgebrochen und in den State idle gewechselt.

step: Der State step bestimmt die Schrittgrösse, welcher im nächsten State für die Annäherung nötig ist. Die Schrittgrössen für alle Töne sind im Vorhinein berechnet worden als 1 Cent Differenz zum eigentlichen Ton. Würde überall die gleiche Schrittgrösse genommen, wäre das Aufschliessen bei hohen Tönen extrem langsam und bei tiefen Tönen so schnell, dass kein Übergang hörbar ist. Wenn die gemessene Frequenz grösser als der anzunähernde Ton ist, muss der Schritt zudem negativ sein.

step count: Der letzte State verrechnet die Frequenzdifferenz in vorgegebenen Intervallen mit dem zuvor bestimmten Step, bis auf den Ton aufgeschlossen ist. Hat die Frequenzmessung jedoch während dem Zählen eine neue Frequenz gemessen wird in den State freq range gewechselt. Erreicht die Messung bevor eine neue Messung stattfand einen Unterschied von unter 6 Cent zu der anzunähernden Frequenz, stoppt das Zählen bis zu einer neuen Messung.

step cnt zu step count ändern

Register

Um mit der Komponente *Tonhöhenverarbeitung* über das Memory-Mapped Interface kommunizieren zu können wurden folgende Register definiert:

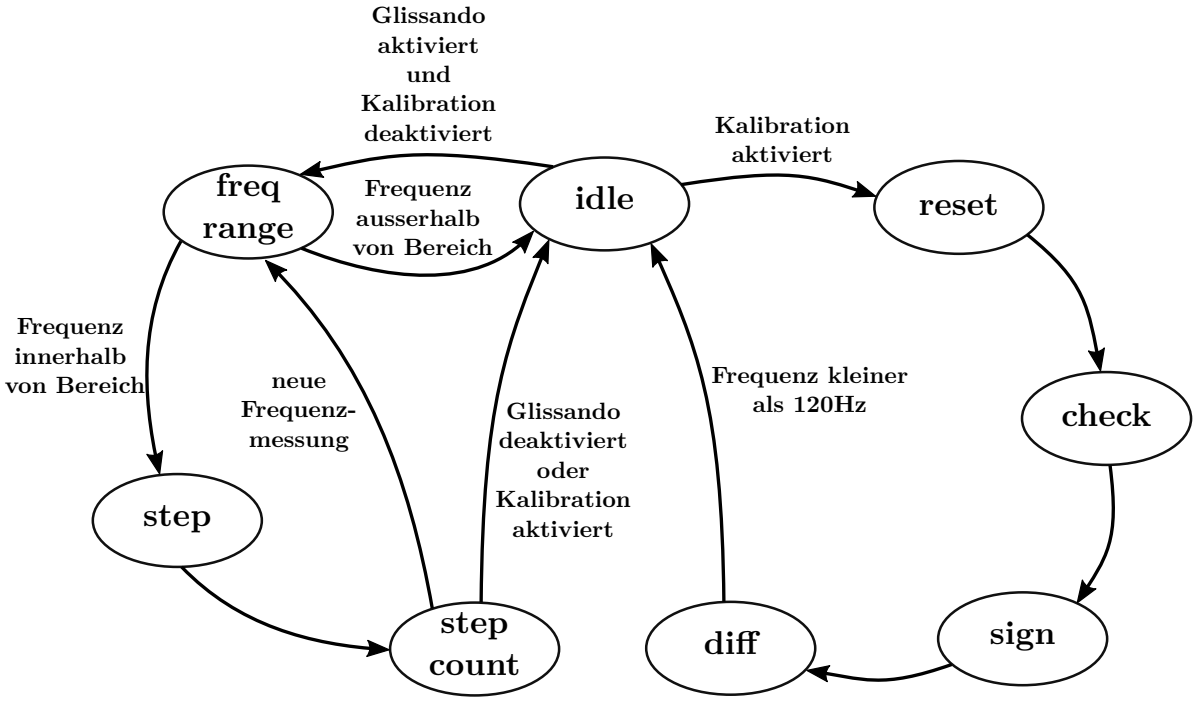


Abbildung 4.10: State Event Diagramm der Kalibration & Glissando Komponente

Tabelle 4.3: Zusammenfassung der Register

Register	**Adresse**	**R/W**	**Bits**			
			31-3	**2**	**1**	**0**
cntrl_reg	00	R/W	X	scale	cal	glis
freq_data_reg	01	R	Frequenz			
delay_data_reg	10	W	Verzögerung			

Tabelle 4.4: Control Register Flags

Bits	**Kürzel**	**R/W**	**Beschreibung**
0	glis	W	1 = Glissando-Effekt aktiviert 0 = Glissando-Effekt deaktiviert
1	cal	R/W	1 = Kalibration aktiviert 0 = Kalibration beendet
2	scale	W	1 = Pentatonische Tonleiter 0 = Normale Tonleiter

Das Register *freq_data_reg* enthält die Frequenz für die Anzeige der Spielgenauigkeit. Mehr dazu in Kapitel 5.3.
Das Register *delay_data_reg* enthält die Einstellung für die Zeit, die der Glissando-Effekt benötigt um die Töne zu korrigieren. Es können Werte von 0 bis 9 geschrieben werden um diese Zeit zu erhöhen.

4.5 Lautstärkeverarbeitung

Die Aufgabe der *Lautstärkenverarbeitung* ist es aus dem Signal des Lautstärkenoszillators einen Dämpfungsfaktor zu berechnen, mit welchem das Signal der Tonhöhenverarbeitung multipliziert wird. So kann der Spieler die Lautstärke während dem Spielen wie die Tonhöhe über eine Antenne einstellen. Wie Abbildung 4.11 zeigt sind die beiden Verarbeitungskomponenten auch sehr ähnlich. Der Referenzoszillator und der Mischer sind gleich wie in der Tonverarbeitung. In den nächsten Abschnitten sind die Unterschiede zu den Komponenten der Tonverarbeitung aufgezeigt.

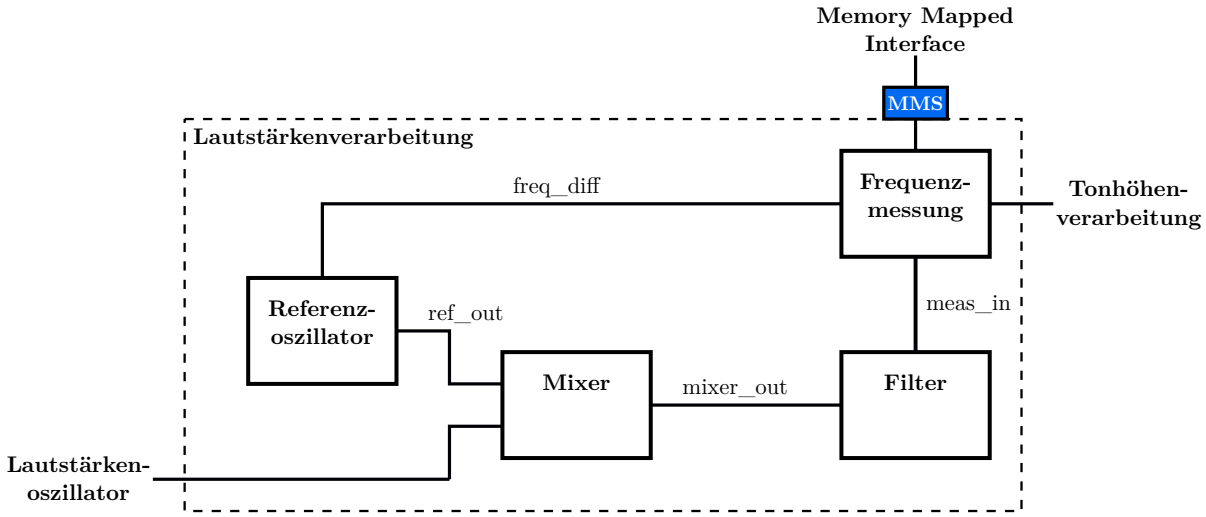


Abbildung 4.11: Blockschaltbild der Custom IP Volume Generation

Filter

Beim *Filter* der Laustsärkeverarbeitung wurde die Frequenzmessung erst nach dem dritten CIC Filter angeschlossen. Wir haben dies so gemacht, um beim FIR-Filter der Frequenzmessungskomponente weniger Koeffizienten und somit weniger Ressourcen zu benötigen. Das FIR-Filter benötigt deshalb weniger Koeffizienten, da nach CIC 3 die Abtastfrequenz des Signals um den Faktor 45 kleiner ist, nämlich 240kHz. Zudem ist das FIR-Filter in dieser Komponente deshalb nicht mehr nötig.

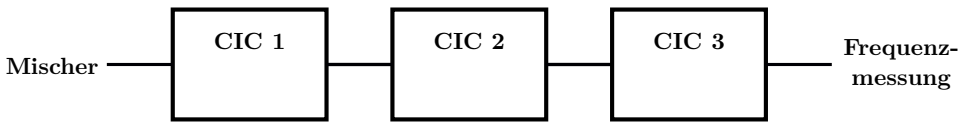


Abbildung 4.12: Aufbau des Filters in der Komponente Volume Generation

Frequenzmessung & Kalibration

Auch bei der Komponente *Frequenzmessung* waren Anpassungen nötig. Die Abtastfrequenz des Signals aus dem Filter haben wir wie im vorherigen Abschnitt besprochen auf 240kHz gesetzt. Dies bedeutet, dass die Frequenzmessung weniger genau ist, was jedoch für die Einstellung der Lautstärke nicht so problematisch ist wie für die Tonhöhe. Der Vorteil von einer tieferen Abtastfrequenz ist, dass das FIR-Filter weniger Filterkoeffizienten hat und somit Ressourcen einspart. Der Periodenzähler ist gleich geblieben wie in der Tonhöhenverarbeitung.

Anders als zuvor sind nun zwei Goldschmidtdividierer verbaut. Dies um am Ausgang zu der Tonhöhenverarbeitung die in Kapitel 4.4 besprochenen Dämpfungsfaktor zu berechnen. Wie zuvor berechnet der *Goldschmidtdividierer 1* aus der Anzahl Abtastperioden aus per_cnt die entsprechende Frequenz. *Goldschmidtdividierer 2* berechnet aus der Frequenz den erwähnten Dämpfungsfaktor. Wir haben uns dafür entschieden diese Berechnung so einzustellen, dass alle Messungen unter 300 Hz einen Dämpfungsfaktor 0 oder kein Ton ergibt. Das Maximum oder 1 haben wir auf 3500 Hz eingestellt. Da die Frequenz des Signals welches vom Filter kommt exponential Ansteigt, muss keine Umrechnung stattfinden um die Eigenschaft des logarithmischen Hörens zu kompensieren.

Anders als bei der der gleichen Komponente in der Tonhöhenverarbeitung ist die Funktionalität des Glissando-Effekts in der Lautstärkeverarbeitung nicht nötig. Die Kalibration funktioniert jedoch gleich wie in der Tonhöhenverarbeitung.

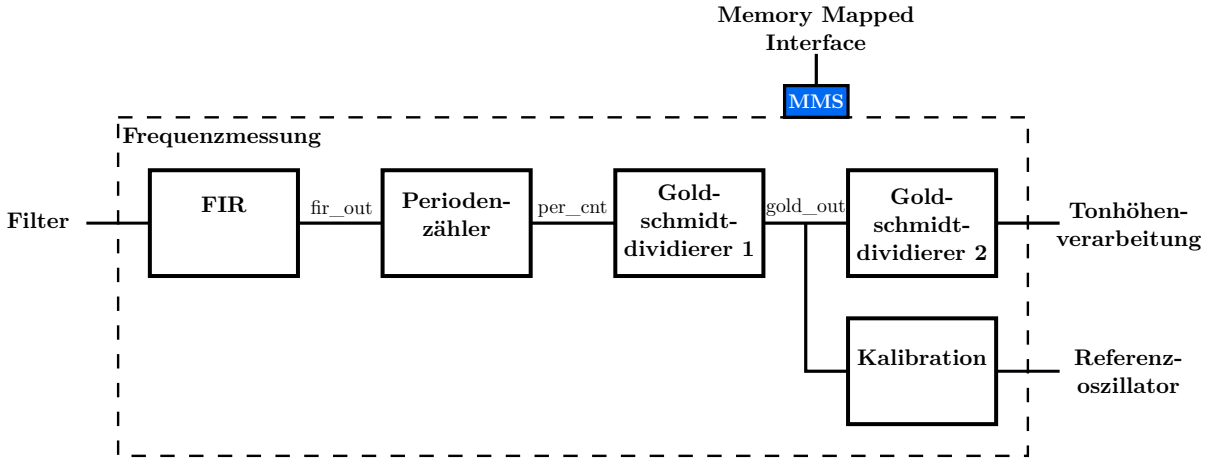


Abbildung 4.13: Aufbau der Frequenzmessung und Kalibration in der Komponente Volume Generation

Register

Um mit der Komponente *Lautstärkenverarbeitung* über das Memory-Mapped Interface kommunizieren zu können wurde das folgende Register definiert:

Tabelle 4.5: Control Register Flags

Bits	Kürzel	R/W	Beschreibung
0	ant_on	W	1 = Lautstärkeantenne aktiviert 0 = Lautstärkenantenne deaktiviert
1	cal	R/W	1 = Kalibration aktiviert 0 = Kalibration beendet

Ist das Flag ant_on auf 0 gesetzt, lässt sich das Theremin nur über die Tonhöhenantenne spielen. Die Lautstärke ist dabei immer konstant.

4.6 Audioserialisierer

Für die Übertragung der Audiodaten zum Codec ist der *Audioserialisierer* zuständig. Obwohl es von Intel bereits eine IP gäbe, um diese Übertragung zu übernehmen, mussten wir diese Komponente nochmals selber schreiben. Dies, da wir sicherstellen mussten, dass die Clocks dieser

Komponente und der Tonhöhenverarbeitung vom selben PLL generiert werden, da ansonsten Werte übersprungen oder verloren gehen. Leider kann die IP von Intel nur mit 12.288 MHz betrieben werden. Der IP Core kann jedoch nicht eine solche Frequenz und die gewünschten 54 MHz für die Tonhöhenverarbeitung in einem PLL generieren.
Wie der Name der Komponente schon sagt serialisiert sie die Parallelen Daten für die Übertragung. Dabei sieht der geforderte Signalverlauf des Codec wie in Abbildung 4.14 dargestellt aus. Der Audioserialisierer gibt jeweils für den linken und rechten Kanal dieselben Daten aus. Die Signale DACLRC und BCLK werden vom Codec nach der Konfiguration beschrieben in Kapitel 5.3 vom Codec generiert und sind Eingänge des Audioserialisierers. Bei jeder positiven Flanke lädt der Serialisierer einen neuen Audiosignalwert ins Schieberegister und schiebt bei jeder negativen Flanke des BCLK ein neues Bit ins Signal DACDAT.

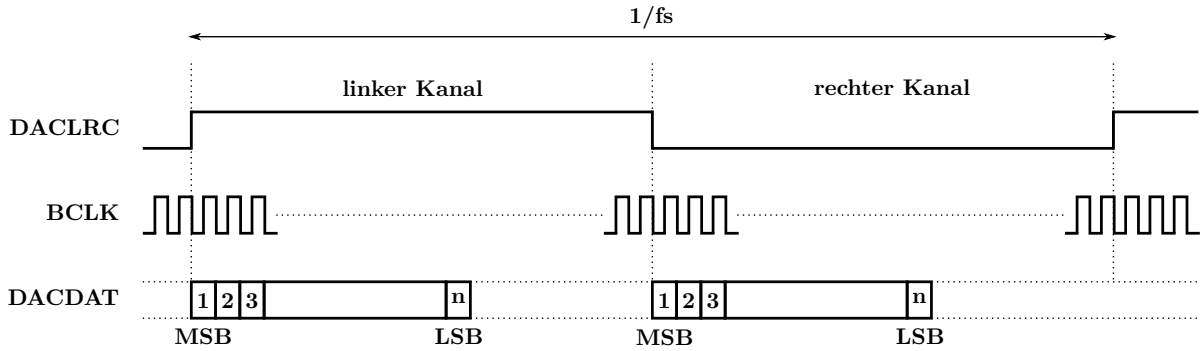


Abbildung 4.14: Signalverlauf des Codec Interface

5 Realisierung Software

Im folgenden Teil des Fachberichtes ist der Aufbau der Software beschrieben. Es folgt zu Beginn eine Gesamtübersicht.

Die Software ermöglicht die Bedienung des Theremin. Zur Steuerung des Theremin ist auf dem NIOS II eine in C programmierten Statemachine realisiert. Die Bedienung geschieht über das LT24 LCD Touch Module von Terasic. Die von Terasic zur Verfügung gestellten C Dateien zur Ansteuerung des LCD und des Touch, sowie die Dateien zur Darstellung des GUI, stellten sich als wenig brauchbar heraus. Die darin enthaltenen Funktionen sind oft zu ineffizient. Daher entschieden wir uns diese Funktionen selber zu schreiben.

5.1 Hauptprogramm State Machine

Abbildung 5.1 zeigt die Initialisierung und den Hauptprogrammfluss der Software.

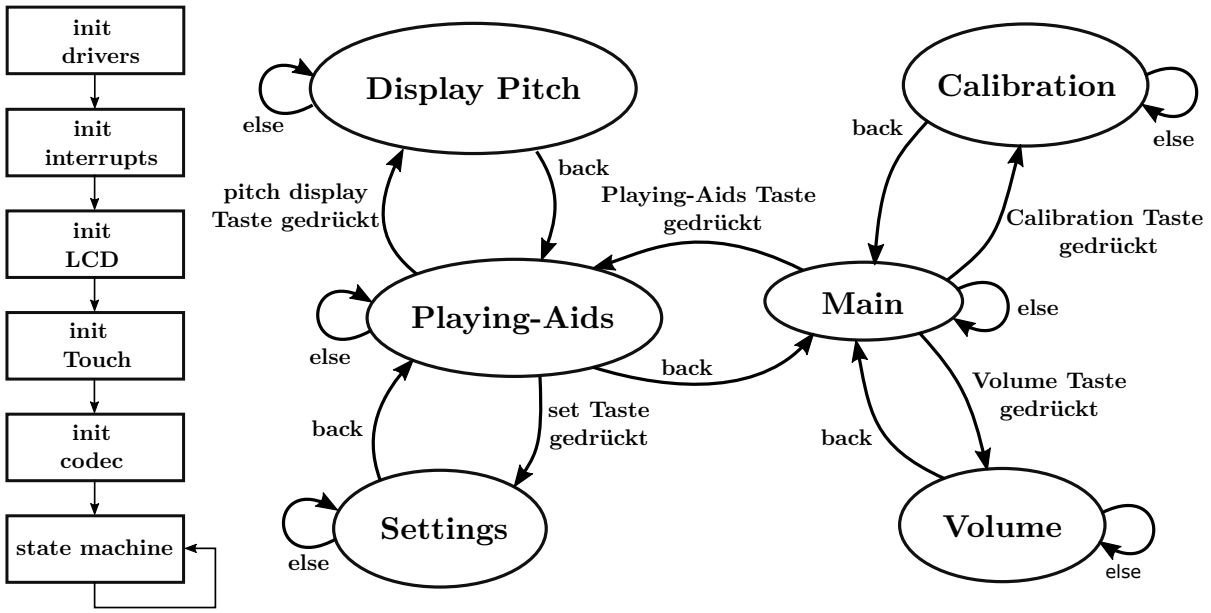


Abbildung 5.1: State Machine und Initailisierung.

Als erstes initialisiert das Programm alle Treiber, den Touch, das LCD, und den Codec. Anschlissend führt das Programm in einer Endlosschlaufe die state machine durch.

Main: Dies ist der erste State nachdem Initialisierungsvorgang. Der Main State wartet auf eine Betätigung der drei in Abbildung 5.2a gezeigten Tastern. Eine Betätigung der drei Tastern führt zu einem Wechsel in den entsprechenden State.

Calibration: Der Calibration State fordert den Benutzer auf seine rechte Hand in die Nähe der rechten Antenne zu halten. Nach zwei Sekunden startet die Kalibrierung. Sobald die Kalibrierung abgeschlossen ist, kann der Benutzer über den ***back*** Taster zurück in den Main State gelangen. Abbildung 5.2b zeigt die Darstellung des LCD nach erfolgreicher Kalibrierung.

Volume: In diesem State kann der Benutzer die Lautstärke ändern und die Lautstärkenantenne aktivieren und deaktivieren. Die Lautstärke kann in 10 verschiedene Pegeln eingestellt werden. Dies geschieht mithilfe der in Abbildung 5.2c gezeigten + und - Tasten. Beim betätigen der

vol antenna Taste wird die Lautstärkenantenne je nach aktuellem Zustand deaktiviert oder aktiviert. Mit dem betätigen der ***back*** Taste gelangt der Benutzer in den Main State.

Playing-Aids: Im State Playing Aids kann der Benutzer mit der ***Glissando on*** Taste der Glissando Effekt akieviert und deaktiviert. Die Abbildungen 5.2d und 5.2e zeigen die grafische Realisierung dieser Taste. Über die ***Set*** Taste gelangt der Benutzer in den State Settings. Durch betätigen des ***display pitch*** Taste wird in den State Pitch Display gewechselt. Zurück in den Main State gelangt man über die ***back*** Taste.

Settings: Im Settings State können Einstellungen zum Glissando Effekt gemacht werden. Das Delay des Glissando Effekts ist in 10 Stufen einstellbar. Zudem kann der Anwender mit dem Taster ***penta*** zwischen der Pentatonischer und der normalen Tonleiter wechseln. Mit dem betätigen der ***back*** Taste gelangt der Benutzer in den Playing-Aids State. Abbildung 5.2f zeigt wie der Setting State grafisch auf dem LCD aussieht.

Pitch Display: Dieser State unterstützt den Theremin Spieler dabei Töne der Tonleiter zu treffen. Der Spieler bekommt über das Display eine optische Rückmeldung in welcher Region der Tonleiter sich der gespielte Ton befindet, so muss der Spieler nicht alleine auf sein Gehör vertrauen. Abbildung 5.2g zeigt die graphische Darstellung dieses States. Der Schriftzug oberhalb des LCD zeigt den Ton an, der sich in der Region der gespielten Frequenz befindet. Der kleine vertikale Strich zeigt dem Spieler graphisch an wie weit weg die aktuell gespielte Frequenz von dem Ton der Tonleiter ist. Diese graphische Unterstützung kann jedoch nur bei der Pentatonischen Tonleiter angewendet werden. Da die Antenne sehr empfindlich auf Änderungen ist, ist es mit der normalen Tonleiter nicht möglich den Vertikalen Strich anzuzeigen. Daher wird dieses Anzeige nur bei der Pentatonischen Tonleiter verwendet.

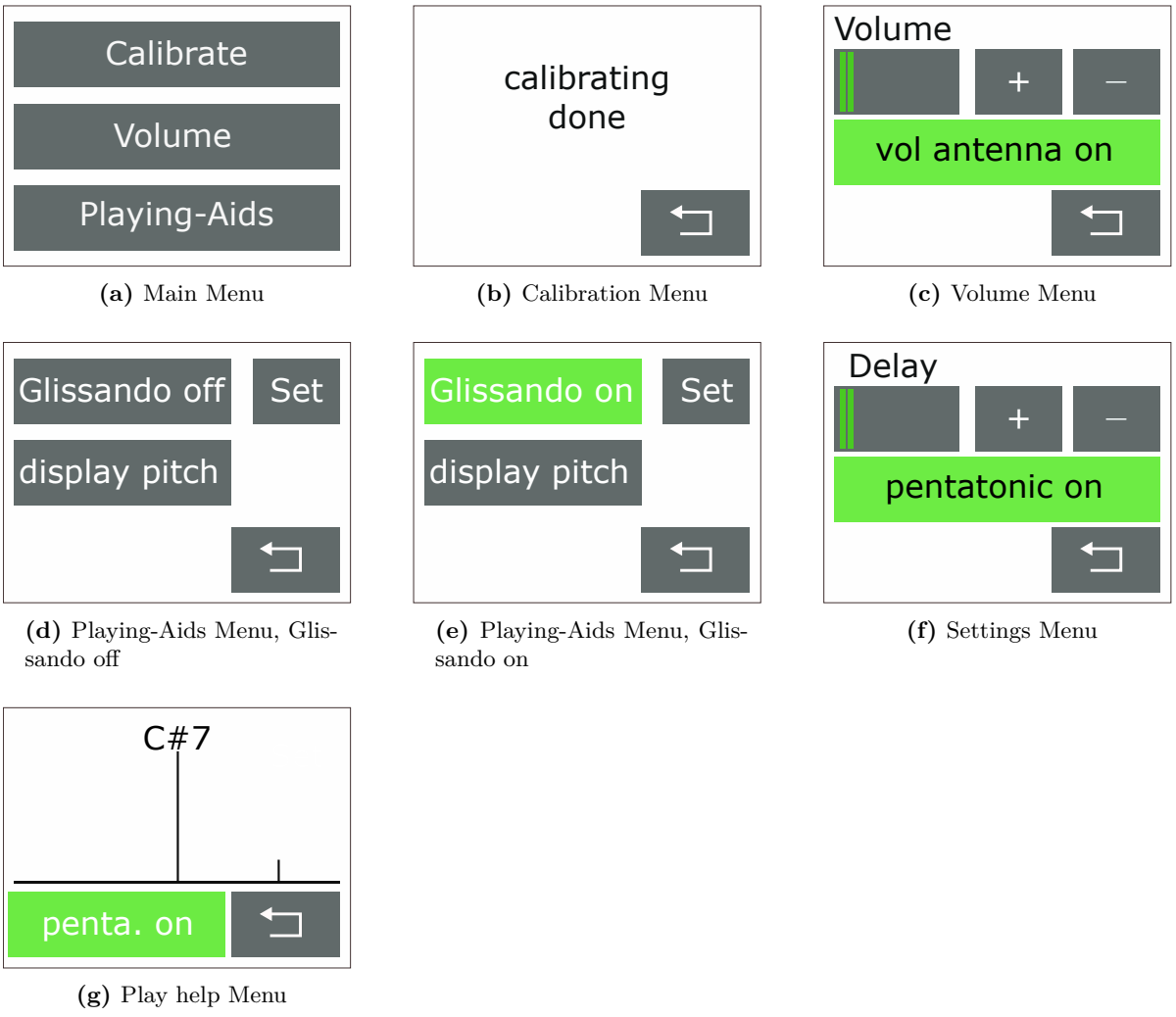
5.2 Treiber

In diesem Unterkapitel sind die verschiedenen Funktionen der selbst geschriebenen Treiber für die Custom IP componenten Tonhöhenverarbeitung und Lautstärkeverarbeitung beschrieben. Zusätzlich war die Erstellung eines Treiber für den LT24 Controller nötig, da Terasic den zur Verfügung gestellte Treiber nicht nach dem Konventionen von Intel erstellt hat.

Um ein selbst erstellten Treiber dem Board Support Package (BSP) hinzuzufügen, müssen die Benennungen und Ablage Orte der Files einige Bedingungen einhalten. Die Treiber Dateien müssen in einem Ordner IP abgelegt sein, welcher sich im quartus Ordner befinden muss. Darin muss ein Skript mit der Endung sw.tcl abgelegt sein. In diesem Skript muss ein eindeutiger Name für den Treiber angegeben sein. Zudem muss der Pfad zu den Treiber Daten angegeben sein. Intel empfiehlt drei Treiber Daten zu erstellen:

- inc
  - custom_ip_regs.h
- HAL
  - custom_ip_.h
  - custom_ip_.c

Das file mit der Endung regs.h definiert hardware interface spezifische Abläufe. Dieses wird im Ordner inc abgelegt. Im HAL Ordner sind ein c und ein h file erstellt, welche die Integration mit dem hardware abstracten layer(HAL) ermöglichen [14]. Die folgenden Paragraphe zeigen die in den c Dateien realisierten Funktionen für die drei Treiber.



(a) Main Menu

(b) Calibration Menu

(c) Volume Menu

(d) Playing-Aids Menu, Glissando off

(e) Playing-Aids Menu, Glissando on

(f) Settings Menu

(g) Play help Menu

Abbildung 5.2: Dummy figure

LT24 Controller Wie in der Einleitung dieses Unterkapitels erwähnt erfüllt der von Terasic mitgelieferte Treiber nicht die Konventionen welche Intel verlangt. Zudem sind die meisten Funktionen des Treibers sehr ineffizient gestaltet. Darum haben wir uns entschieden den Treiber für den LT24 Controller selbst zu schreiben. Im folgenden Teil ist beschrieben welche Funktionen wir für die Steuerung des LCD erstellt haben.

Das Mudul **LT24_Controller.c** ermöglicht es auf dem LCD einzelne Pixel und Rechteckflächen zu zeichnen und zu löschen. Die Funktion *LCD_DrawPoint(x,y,color)* setzt ein Cursor an die gewünschte Stelle auf dem LCD und zeichnet ein Pixel in der entsprechenden Farbe. Auf diese Art ein Rechteck zu zeichnen ist sehr ineffizient. Da der Treiber von Terasic Rechtecke auf diese Art zeichnet, haben wir entschieden eine bessere Lösung zu finden. Mit der Funktion *LCD_DrawRect(xs,ys,xe,ye,color)* werden Rechtecke effizienter gezeichnet. Es wird dabei nur einmal für das ganze Rechteck ein Cursor Feld gesetzt und danach alle Pixel eingefärbt. Da durch das setzen des Cursor Feld nicht für jeden einzelnen Pixel ein Cursor gesetzt werden muss, wird Zeit eingespart. Dadurch ist es für das Menschliche Auge nicht mehr ersichtlich wie einzelne Pixel gezeichnet werden, wie dies bei der Funktion von Terasic der Fall ist.

Tonhöhenverarbeitung Das Modul **Pitch_generation.c** ermöglicht auf das Kontroll und das Glissando Delay Register zu schreiben und diese zu lesen. Das Register freq data ist nur lesbar. Dieses liefert die aktuelle Frequenz der Tonhöhenantenne und der Index des Tones bei dem die Frequenz am nächsten liegt. Die Funktion *get_pixel_pitch_accuracy(penta_on_off,pitch_freq)* liesst das freq data Register aus. Für den Modus Pitch Display, zu sehen in Abbildung 5.2g, ist es nötig anzuzeigen auf welcher Seite sich der Benutzer von einem Ton befindet. Daher wird in dieser Funktion das Verhältnis der aktuellen Tonhöhenantenne Frequenz mit dem Abstand zum nächsten Ton gebildet. Mit diesem Verhältnis wird ein Pixel Wert berechnet. Dieser wird gebraucht um denn vertikalen Strich auf dem LCD zu zeichnen.

Lautstärkeverarbeitung Die einzige Kominikation welche die Lautstärkeverarbeitung Komponente mit dem NIOS II hat, ist das schreiben und lesen des Kontroll Registers um die Kalibrierung zu starten. Das Module *Volume_generation.c* ermöglicht dies.

5.3 Audio

Die Kommunikation mit WM8731 Codec geschieht über die IP Componente Audi_Video Configuration von Intel. Diese besitzt die Funktion *alt_up_av_config_write_audio_cfg_register* um die Kontrollregister des Codec über den Hardware Abstraction Layer (HAL) anzusprechen[15]. Das Module **audio.c** beinhaltet die beiden Funktionen, *codec_wm8731_init()* und *set_vol(vol_gain)*.

In der Initialisierung wird der Codec als Master konfiguriert. Der Clock des Codec wird auf 12 MHz gesetzt und die Sampling Rate auf 48 kHz. Die Input Audio Data Bit Länge wird auf 24 Bit gesetzt und der Übertragungsmodus der Daten auf left justified gesetzt. Mit diesen Einstellungen ist eine Kominikation mit der audio serializer Komponente möglich . Um Störungen zu vermeiden sind die Line In Eingänge des Codec gemutet, da wir nur den Line Out brauchen. Der Linke und Rechte Kanal sind so eingestellt das beide Kanäle die selben Lautstärken haben[16].

Verweiss auf Dennis

Die Gesamtlautstärke des Theremin ist auf 10 unterschiedliche Pegel einbestellbar, dies geschieht mit dem Aufruf der Funktion *set_vol(vol_gain)*. Die leiseste Stufe dämpft den Audio Signal Pegel um −76 db. Jeweils eine Stufe grösser reduziert die Dämpfung um 7 db. Die höchste Stufe dämpft den Audio Signal Pegel um −7 db. Der Codec könnte gemäss Datenblatt das Signal bis +6dB verstärken. Nach Labor Tests haben wir uns entschieden das Audio Signal nur zu dämpfen, da eine Verstärkung zu laut ist.

Ursprünglich war geplant die Lautstärke Einstellung der Volume Antenne über den Codec zu steuern. Nicht wie in Kapitel 4.5 beschrieben über die VHDL Komponente. Diese Methode konnte jedoch nicht realisier werden, da die Zero Cross detection des Codec nicht funktionierte. Mit der Zero Cross Dedection sollte der Codec erst bei einem Nulldurchgang die Lautstärke vermindern oder erhöhen. In unseren Versuchen hat sich die Lautstärke jedoch nie geändert, obwohl ein Sinus Audio Signal angelegt wurde. Wir mussten feststellen das diese Methode für den DAC nicht funktioniert. Mit dieser Methode hätte die Lautstärke, verglichen zur realisierten Methode, eine bessere Dynamik gehabt. 20. August 2020genauer erklären

5.4 Touch

Die von Terasic zur Verfügung gestellte Datei zur auslesen des Touch ist leider sehr unübersichtlich aufgebaut. Es ist sehr schwierig die darin enthaltenen Funktionen auf unser Projekt anzuwenden. Daher entschieden wir uns selbst ein Touch Interrupt Routine zu erstellen.

Der resistive Touch Display des LT24 LCD touch module ist mit dem AD783 Analoge Devices Chip verbunden. Sobald der Touch Display berührt wird lösst der Chip am Pen Interrupt Pin ein Interrupt aus welches vom NIOS II detektiert wird. Der AD783 Chip speicher die X und Y Koordinaten zum Zeitpunkt des Interrupts ab. Diese können über den SPI Bus ausgelesen werden[17].

Für unser Projekt sind wir daran interessiert den Pen Interrupt zu detektieren und die X und Y Koordinaten auszulesen, damit wir sagen können welcher Taster gedrückt worden ist.

Das ganze ist im Modul **touch_isr.c** realisiert. Darin befindet sich eine Funktion *touch_init(void*context)*. Diese aktiviert das Touch Pen Interrupt und registriert die Funktion welche durch das Interrupt aufgerufen wird. In der Interrupt Funktion *touch_isr(void*context)* wird zuerst das Touch Pen Interrupt deaktiviert. So das in dieser Zeit kein weiteres Interrupt auftreten kann. Nach dem deaktivieren des Interrupts liesst der *alt_avalon_spi_command* die X und Y Koordinaten aus.

5.5 GUI

Die Funktionen welche von Terasic für die Gestaltung des GUI zur Verfügung gestellt werden sind für unser Projekt zu ineffizient. Um ein Text anzuzeigen verwendet Terasic eine Funktion die ein Alpha Blending durchführt. Dieses wird immer an den Rändern der Buchstaben durchgeführt. Dabei wird die Schriftfarbe mit der Hintergrundfarbe gemischt. Dieser Prozess nimmt viel Zeit in Anspruch. Wodurch bei jedem neuen zeichnen eines Texte Strings zugeschaut werden kann wie die Pixel gezeichnet werden. Dies ist für unser Projekt unbrauchbar. Zudem hat uns die von Terasic verwendete Schriftart nicht gefallen. Daher entschieden wir uns eine eigene Funktion zu schreiben welche Text stringe zeichnet. Diese ist im Modul **simple_text.c** realisiert.

simple_text.c Da das LCD nur in der Lage ist Pixel zu setzen gestaltet sich das zeichnen einer Schrift eher mühsam. Mit dem open source tool "The Dot Factory"generierten wir uns eine Bitmap für die Schrifftart Arial welche 22points gross gewählt ist. Das tool generiert zwei Arrays. Im Character Bitmap Array sind die Character in einer Bitmap gespeichert. Das Character Descriptor Array enthält Informationen über die Breite jedes Characters und den Offset in der Character Bitmap. Um den Ort eines Characters zu bestimmen wird der Character minus des ersten Character in der Bitmap gerechnet. Dies ergibt den Index für das Descriptor Array in welchem der Offset für die Bitmap gespeichert ist. In Abbildung 5.3 ist dieser Prozess veranschaulicht. In diesem Beispiel besteht die Bitmap aus den Buchstaben abc. Für den Buchstaben b wird der Offset 1 ausgerechnet. Im Descriptor Arrey ist auf Position 1 die Breite 5 gespeichert und der Offset 10 für die Bitmap. Mit diesen Informationen kann die Funktion *print_string(x,y,color,font,font_descriptor,string)* einen Text String zeichnen. Dabei werden nur die Pixel gezeichnet welche in der Bitmap auf 1 gesetzt sind.

gui.c In diesem Modul sind mit der Funktion *print_string(x,y,color,font,font_descriptor,string)* aus dem Modul **simple_text.c** und der Funktion *LCD_DrawRect(xs,ys,xe,ye,color)* aus dem Modul **LT24_controller.c** die einzelnen Menüs dargestellt.

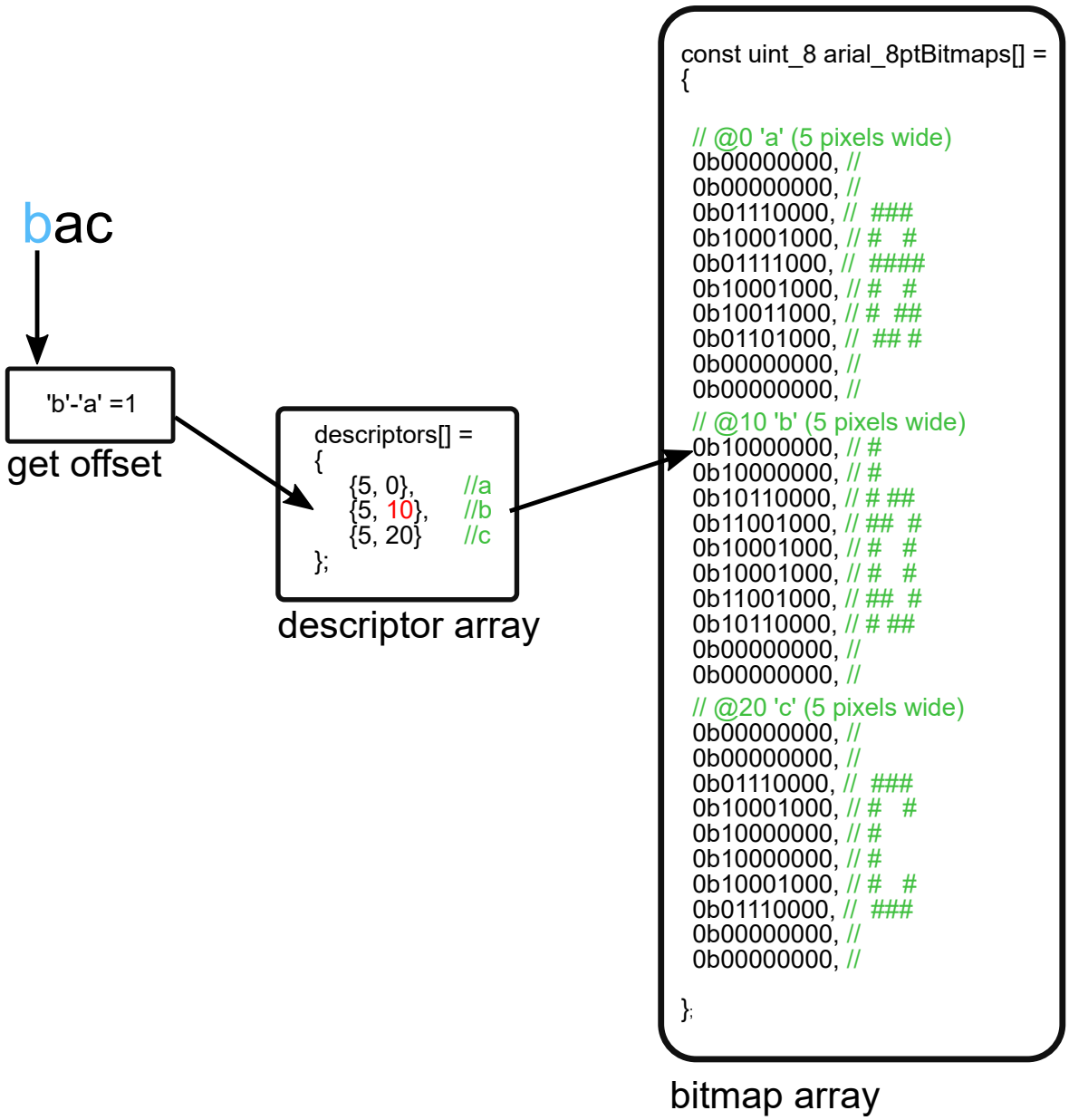


Abbildung 5.3: Bitmap und Discriptor Array.

6 Realisierung Gehäuse

Im folgenden Teil des Fachberichtes ist die Realisierung des Gehäuse beschrieben.

Die Anforderungen an das Gehäuse des digitalen Theremin sind:

- Die Volume und die Pitch Antenne müssen genügend Abstand zueinander haben.
- Das Antennenoszillator PCB und das DE1-SoC Board müssen für den Spieler ersichtlich sein.
- Für die Bedienung muss das LT24 Touch Module für den Benutzer gut sichtbar platziert sein.
- Um den Preis des Gehäuses tief zu halten und für die Gestaltung des Gehäuses möglichst viel Freiheit zu haben, wird das Gehäuse 3D-gedruckt.

Um diese Anforderungen zu erfüllen entschiedenen wir uns das Gehäuse mit dem 3D-CAD-System Inventor zu konstruieren. Inventor ist eine sehr umfangreiche Konstruktion Software die einige nützliche Funktionen enthält um 3D Modelle zu erstellen. Zudem können Studenten für ein Jahr lang die Software für nicht kommerzielle Zwecke frei nutzen[18]. Um das Gehäuse zu drucken hatten wir uns entschieden den S5 Ultimaker 3D Drucker zu verwenden, da dieser sehr benutzerfreundlich ist und ein grosses Druckvolumen hat. Der S5 Drucker hat eine Druckvolumen von $330\,\text{mm} \times 240\,\text{mm} \times 300\,\text{mm}$. Im Verlauf der Realisierung des Gehäuses stellte sich jedoch heraus das, dass Gehäuse bedeuten grösser wird als das Druckvolumen des Druckers. Da wir kein 3D Drucker mit grösserem Druckvolumen zur Verfügung hatten, ist das Gehäuse in vier Einzelteilen konstruiert.

Wir entschieden uns die Form des Theremin oval zu gestalten, da der 3D-Drucker diese Geometrie erlaubt. Die vier Einzelteile des Gehäuses sind alle gleich aufgebaut. Abbildung 6.1 zeigt die Grundstruktur eines Einzelteil. Die Funktion Wandung ermöglichte es die Grundstruktur oval auszuhöhlen. Jeweils zwei Einzelteile bilden zusammen den Deckel und den Boden. Daraus resultierte das in Abbildung 6.2 gezeigte Gehäuse.

Die vier Einzelteile sind aus schwarzem Polylactide (PLA) gedruckt. Da das Gehäuse eine ovale Geometrie hat, braucht es Stützstrukturen für den Herstellungsprozess. Das eingesetzte Polyvinylalkohol (PVAL) hat die nützliche Eigenschaft das es wasserlössliche ist. Nachdem das Bauteil einigen Stunden im Wasser eingelegt war, verschwanden die Stützstrukturen sehr gründlich. Das drucken der vier Einzelteilen dauerte insgesamt sechs Tage.

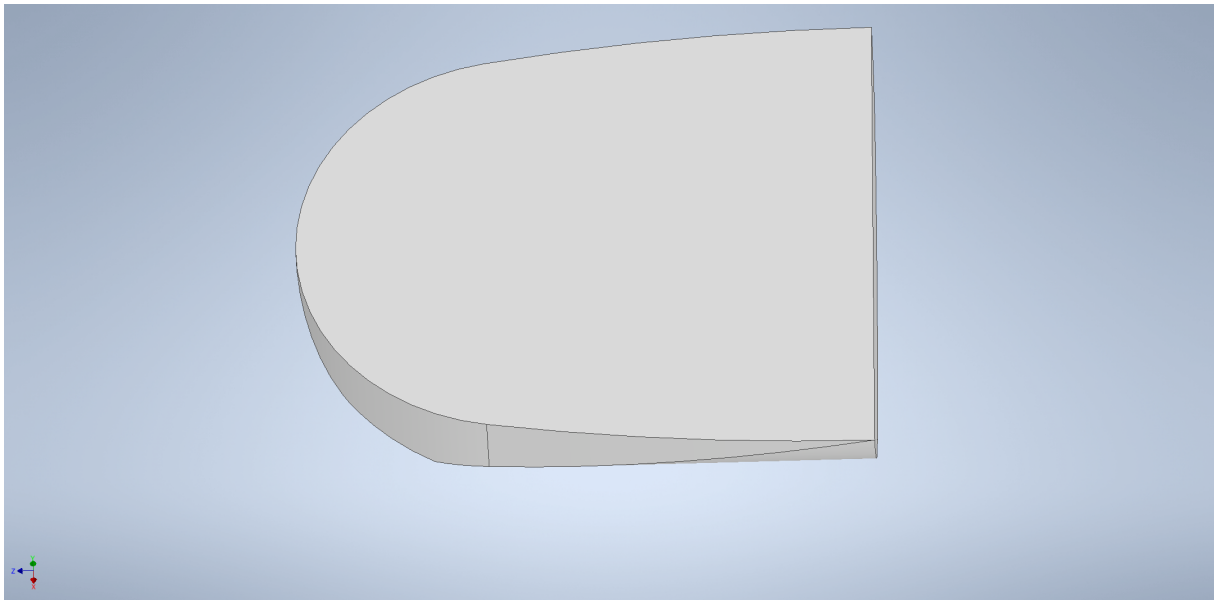


Abbildung 6.1: Grundstruktur eines Einzelteils.

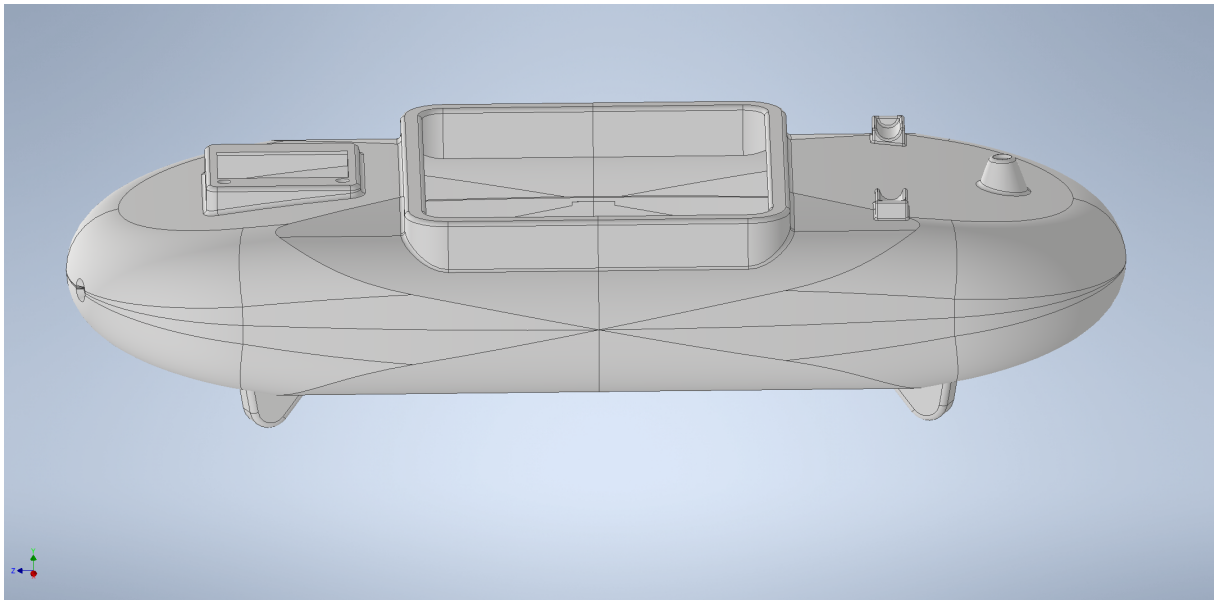


Abbildung 6.2: Theremin Gehäuse.

7 Validierung

In diesem Kapitel sind die Resultate der Messungen des PCB, der Frequenzmessung, des Glissando Effekts und des Ton Display aufgeführt. Dabei wird jeweils der Messaufbau beschrieben und die Resultate dargestellt und kommentiert.

7.1 Antennenoszillator PCB

Auf dem Antennenoszillator PCB haben wir die Betriebsspannungen und die Ausgänge der Komperatoren getestet. Die dazu verwendeten Messmittel sind in Tabelle 7.1 angegeben. Die Signale haben wir mit dem Lecroy wavesurfer 3054 Oszilloskop gemessen.

Die 9 V Betriebsspannung hat eine Rippelspannung von 7.3 mV. Somit ist die Colpitts Oszillator Schaltung stabil gespiessen. Die 3.3 V Betriebsspannung weisst einen Rippelspannung von 133 mV auf. Dies ist jedoch noch vertretbar, da die Spannung für den digitalen Teil der Schaltung gebraucht wird.

Tabelle 7.1: Gemessene Spannungen PCB

Spannung	**soll [VDC]**	**ist [VDC]**	**Ripple [mVDC]**
Betriebsspannung 9V	9	9.3	7.3
Betriebsspannung 3.3V	3.3	3.4	133
Schaltnetzteil 12V	12	12.5	150

Bei den Messungen der Ausgängen der Komperatoren waren auf den Rechtecken bei der Steigenden und fallender Flanke Überschwinger ersichtlich. In einem Paper von Analog Device haben wir erfahren, das die Verursacher der Überschwinger die langen Ausgangsleitungen sind. Diese wirken wie nicht abgeschlossene Übertragungsleitungen und lösen Reflexionen aus [19]. Um dieses Problem zu lösen, schlossen wir die Aussgangsleitung mit einem 300 Ohm Widerstand ab. Danach waren keine Reflexionen mehr ersichtlich.

Beim ersten Austesten der Oszillatoren mit angeschlossenen Antennen wiesen die Oszillatoren ähnliche Frequenzen auf. Bei einer Veränderung der Lautstärkeantenne Frequenz wurde ebenfalls die Tonhöhenantennenfrequenz verändert. Dies kann durch Übersprechen oder gegenseitige Beeinflussung der Oszillatoren hervorgerufen werden. Um dieses Problem zu umgehen haben wir entschieden die Frequenzen der beiden Oszillatoren um 30 kHz versetzt zu wählen. Dies führte dazu das die beiden Probleme verschwanden. Jedoch können wir nicht sagen welches der beiden Problem es ist.

Durch den häufigen gebrauch des PCB ist uns bewusst worden, dass der verwendete JFet sehr empfindlich gegenüber Elektrostatische Entladung ist.

Bei einer weiter Entwicklung des Theremin muss das Antennenoszillator PCB überarbeitet werden. Es ist eine Schutzbeschaltung für den JFet notwendig. Zudem sollten die Oszillatoren auf zwei seperaten PCB realisiert werden um Übersprechen und gegenseitige Beeinflussung zu vermeiden.

Tabelle 7.2: Verwendete Messmittel

Messgerät	Bezeichnung
Oszilloskop	Lecroy wavesurfer 3054

7.2 Frequenzmessung

Um die Genauigkeit der Frequenzmessung zu bestimmen, testeten wir das Theremin bei den Frequenzen aus der Tabelle 2.1 in Kapitel 2.2. Wir verwendeten dazu anstatt unseres PCB einen Funktionsgenerator, da keine genaue Messung mit dem PCB möglich ist. Um die Frequenz auszulesen, wurde das Tool *SignalTap Logic Analyzer* in Quartus eingesetzt um auf das Register der Frequenzmessung mit den Messresultaten zuzugreifen. Zu beginn der Messung wurde der Offset des Referenzoszillator bestimmt. Dazu wurde der Frequenzgenerator um 120 Hz tiefer eingestellt als der Referenzoszillator. Da 120 Hz die Frequenz ist auf die Kalibriet wird. Daraus ergab sich ein Offset von 6 Hz. Für die Bestimmung der minimal und maximal gemessenen Frequenzwerte haben wir uns dafür entschieden aus dem SignalTap 20 Werte auszulesen. Aus dem maximum und minimum bestimmten wir die grösste Abweichung zum entsprechenden Ton. Aus diesen Abweichungen berechneten wir die Werte aus Abbildung 7.1 .

Alle gemessenen Werte liegen unter einem Fehler von 8 Cent. Wie bereits im Kapitel 2.2 beschrieben, ist es schwer zu sagen, wann ein Ton als "nicht getroffen" empfunden wird. Daher haben wir uns darauf geeinigt, dass Werte unter 8 Cent als nicht hörbare Veränderung angenommen werden. Somit erfüllen alle Frequenzen diese Genauigkeit.

Wie in Abbildung 7.1 ersichtlich ist, steigt die Messung mit zunehmender Frequenz immer mehr an. Dies spricht mit der Simulation aus Abbildung 7.2 überein. Auch die tiefen Cent Werte um 600 Hz stimmen mit der Simulation überein, da gewisse Frequenzen mit der gewählten Abtastfrequenz von 1.2 MHz eine genauere Messung ergeben.

Tabelle 7.3: Verwendete Messmittel

Messgerät	Bezeichnung
Funktionsgenerator	MSZ-M-0051
SignalTap Logic Analyzer	

7.3 Glissando Effekt

Auch für den Glissando-Effekt war es nötig einen Test durchzuführen um die Genauigkeit der Töne festzustellen. Wir verwendeten dazu anstatt unseres PCB einen Funktionsgenerator, da keine genaue Messung mit dem PCB möglich ist. Um die Frequenz einzustellen, wurde das Tool *SignalTap Logic Analyzer* in Quartus eingesetzt um auf das Register der Frequenzmessung mit den Messresultaten zugegriffen. Schlussendlich wurde der Ton mit der App *Tonal Energy Tuner* auf dessen Genauigkeit gemessen. Die App gibt diese Genauigkeit direkt in Cent an. Die Messung umfasst alle Frequenzen aus der Tabelle 2.1 in Kapitel 2.2. In Abbildung 7.3 sind alle diese Abweichungen aufgezeigt. Die blaue Kurve zeigt die Resultate, wenn die Frequenz kleiner eingestellt wurde als der anzunähernde Ton. Die orange Kurve zeigt die Resultate, wenn die Frequenz grösser eingestellt wurde als der anzunähernde Ton. Wie zu sehen ist, ist auch diese

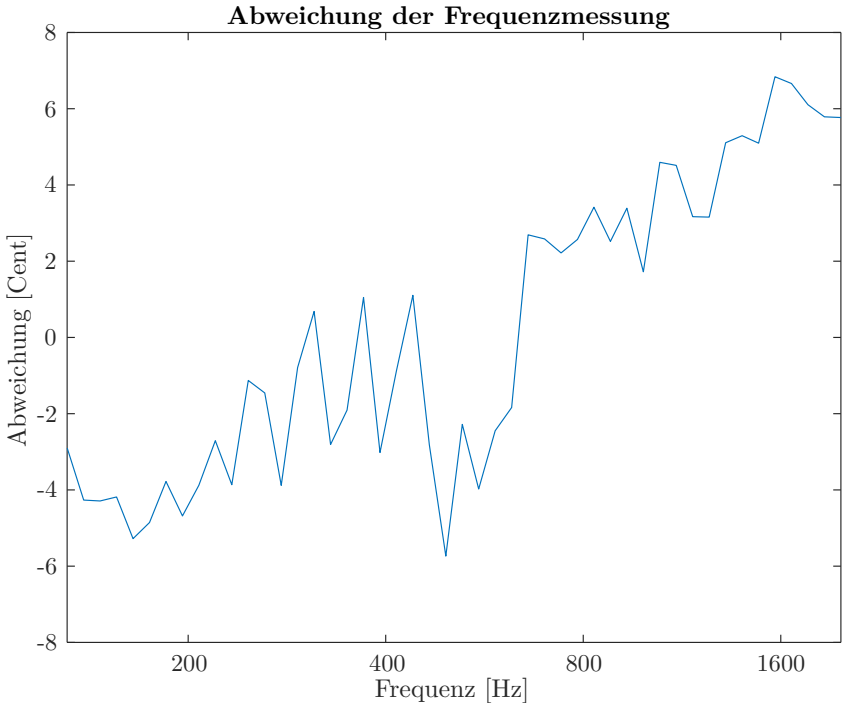


Abbildung 7.1: Maximale Abweichung der Frequenzmessung.

Funktion genauer als 10 Cent. Interessanterweise nimmt bei beiden Einstellungen offenbar die angenäherte Frequenz mit höheren Tönen immer mehr ab. Das Verhalten ist gleich wie das aus Abbildung der Frequenzmessung. Dies ist auch zu erwarten, da der Glissando-Effekt durch diese Messung entsteht und somit die gleiche "Irregularität" hat.

Referenz auf Bild Validierung Frequenz-messung

Tabelle 7.4: Clockfrequenzen der verschiedenen Komponenten

Messmittel	Bezeichnung
Funktionsgenerator	MSZ-M-0051
SignalTap Logic Analyzer	-
Tonal Energy Tuner App	-

7.4 Gesamttest

Zuletzt haben wir das den gesamten Aufbau auf dessen Funktionsfähigkeit getestet. Dabei sind drei Punkte aufgefallen.
Erstens tritt das leichte Aliasing, welches wir erwartet haben bei hoher Lautstärke auf. Auffallend ist, dass dieses nur bei den höheren Tönen (über 1 kHz) auftritt. Dies ist auf die Filterung des Mischersignals durch die CIC-Filter und das verwenden eines Rechtecksignals im Mischer zurückzuführen. Die Oberwellen des Rechtecksignal welche auch Mischprodukte hervorrufen, machen es schwierig durch Filterung mittels CIC-Filter Aliasing ganz zu vermeiden. Diese Problematik ist in Kapitel 2.4 beschrieben.
Zweitens ist bei einer bestimmten Einstellung des Theremin ein Rauschen zu hören. Stellt man die Lautstärke über das Display auf das Maximum und über die Lautstärkenantenne eine tiefe Lautstärke ein, ist ein leichtes Rauschen zu hören. Dies ist auf den Kompromiss, auf welchen wir in Kapitel 5.3 eingegangen sind, zurückzuführen. Da der Digital-Analog-Wandler im Codec

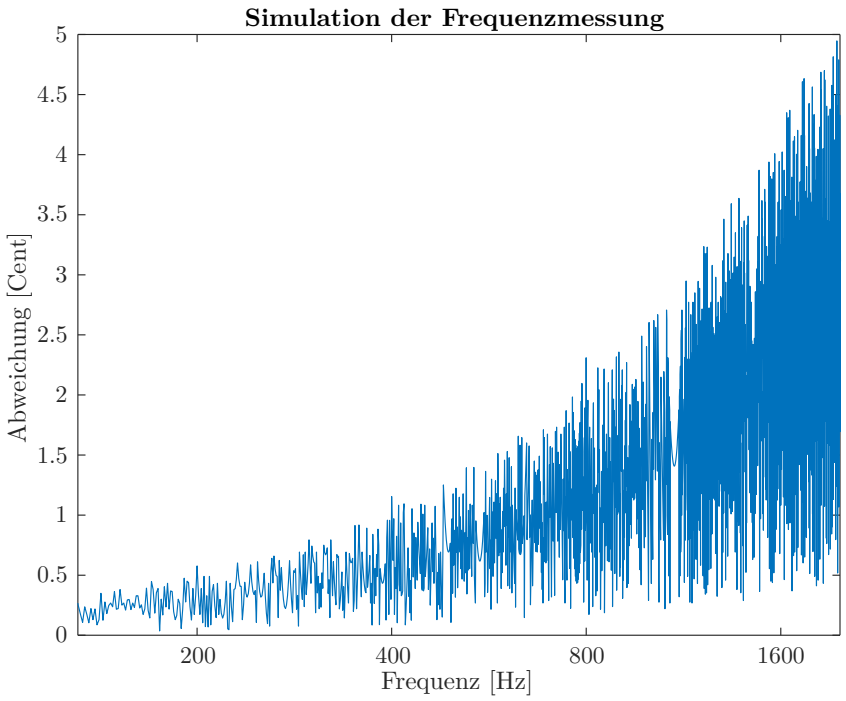


Abbildung 7.2: Abweichung der Frequenzmessung in der Simulation

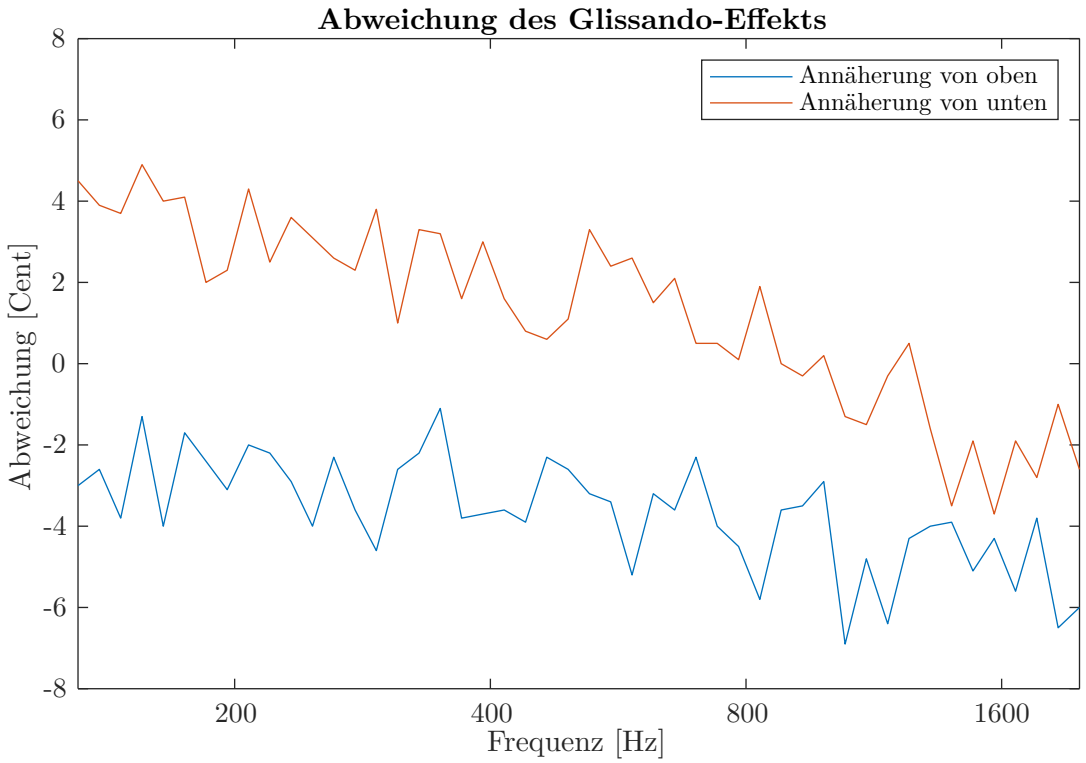


Abbildung 7.3: Resultate der Validierung des Glissando-Effekts

in diesem Fall ein nicht voll ausgesteuertes digitales Signal erhält, fällt dessen Rauschen mehr

ins Gewicht.
Weiter ist uns erst nach der festen

8 Erfahrungen

Über das 5 und 6 Semester haben wir einige Erfahrungen mit Intel FPGA sammeln können. In diesem Abschnitt gehen wir auf unsere Erfahrungen mit den verwendeten Tools ein.

9 Schlusswort

Nach Abschluss der Entwicklung wurde das Produkt validiert. Die Resultate sind in Kapitel **??** dokumentiert. In diesem Kapitel soll aufgelistet werden, welche Teilsysteme funktionieren und welche nicht bzw. welche Zielsetzungen aus dem Pflichtenheft erfüllt wurden. Im letzten Abschnitt wird Verbesserungspotenzial für eine mögliche Weiterentwicklung erfasst.

Die entwickelte Leiterplatte erfüllt einen Grossteil der in Kapitel **??** formulierten Kriterien. Das Produkt umfasst den ersten Prototyp der Steuerleiterplatte und die Mechanik des 3D-Druckers mit diversen Modifikationen. Die Software besteht aus den beiden Firmwares *Marlin* und *ESP3D*. Bei der Steuerung der Motoren funktionieren folgende Punkte einwandfrei:

- Alle vier Motoren (x-, y-, z-Achse und Filamentzuführung) können unabhängig und ohne Schrittverlust angesteuert werden.
- Die Nullpositions-Erkennung der x-, y- und z-Achse funktioniert und diese wird jedes mal automatisch vor jedem 3D-Druck ausgeführt.

Die Bedienung des 3D-Druckers von Hand wie auch auf dem Webinterface funktioniert:

- Die Steuerung des 3D-Druckers erfolgt über Bedienelemente direkt am 3D-Drucker (Encoder, LCD und LEDs).
- Alle Achsen und Motoren können manuell durch Encoder und Display oder das Webinterface gesteuert werden.
- Eine Verwaltung der Druckaufträge kann über das Webinterface durchgeführt werden. Weiter wird auf diesem der Status (Temperatur, Position usw.) angezeigt.

Auch die zusätzlich eingebaute Sensorik funktioniert ohne Probleme.

- Ein Sensor erkennt automatisch eine Erschöpfung des Filaments und meldet dies an das Programm.
- Durch den *BLTouch* Bed-Level-Sensors kann die z-Achse automatisch nivelliert werden. Der Bed-Level-Sensor ist dabei auch für die Nullpositions-Erkennung zuständig.

Die Temperatursteuerung der verschiedenen Komponenten im 3D-Drucker funktioniert ebenfalls:

- Die Temperaturen von Extruder und Heizbett werden erfasst und können über das Webinterface ausgelesen werden.
- Die Leiterplatte, das Filament und der 3D-Druck werden durch Ventilatoren gekühlt.

Leider gibt es auch noch Dinge die noch nicht funktionieren.

- Das senden von G-Code Dateien über das Webinterface auf den 3D-Drucker funktioniert nicht richtig. Überschreitet die Datei eine bestimmte Grösse scheitert das Laden.
- Die Temperaturmessung ist auf $\sim 10^{\circ}$ genau. Somit konnte das Kriterium der genauen Regelung des Heizbettes und des Extruders nur teilweise erreicht werden.

Wie die Validierung zeigt, funktioniert der erste Prototyp. Jedoch gibt es einige Kriterien die zu diesem Zeitpunkt noch nicht erfüllt sind. Diese könnten in einem weiterführenden Projekt noch realisiert werden, bzw. der 3D-Drucker noch weiter verbessert werden.

Da der verwendete $8MHz$ Mikrocontroller viele Probleme mit sich bringt, müsste man bei einer Weiterentwicklung ein anderer Mikroprozessor mit $> 16MHz$ Taktfrequenz verwenden. Damit könnte die Übertragungsrate des Webinterface erhöht werden. Dies würde womöglich auch das Problem des Ladens der G-Code Dateien über Wi-Fi beheben, welches ebenfalls in einer weiterführenden Entwicklung behoben werden müsste.

Es könnte eine Nullpositionserkennung durch Überstromdetektion der Motoren implementiert werden. Dazu müsste lediglich die Firmware angepasst werden, da auf der Leiterplatte diese Funktion schon vorgesehen wurde. Der Vorteil daran wäre, dass weniger mechanische Teile verwendet werden würden (kein Verschleiss, daher weniger Standzeit des 3D-Druckers), sowie das die Verkabelung reduziert werden könnte.

Die Messung der Temperaturen bräuchte eine Überarbeitung, um eine verbesserte Genauigkeit zu erreichen. Dazu müssten andere Komponenten gewählt werden und in der Firmware eine Anpassung getätigt werden.

10 Ehrlichkeitserklärung

Mit der Unterschrift bestätigen die Unterzeichnenden Teammitglieder, dass die vorliegende Projektdokumentation selbstständig im Team und ohne Verwendung anderer, als der angegebenen Hilfsmittel verfasst wurde, sämtliche verwendeten Quellen erwähnt und die gängigen Zitierregeln eingehalten wurden. Eine Überprüfung der Arbeit auf Plagiate mithilfe elektronischer Hilfsmittel darf vorgenommen werden.

Unterschrift:	Ort, Datum:
_____	_____
Unterschrift:	Ort, Datum:
_____	_____

Literatur

[1] WDR. (2010). 05. August 2010 - Vor 90 Jahren: Musikinstrument Theremin wird vorgestellt. (Abrufdatum 17.01.2020), Adresse: https://www1.wdr.de/stichtag/stichtag5844.html.

[2] Garrett Tiedemann. (2016). A history of the theremin in movie music. (Abrufdatum 17.01.2020), Adresse: https://www.classicalmpr.org/story/2016/07/15/theremin-movie-music.

[3] T. Riegler, „Theremin zum selberbauen“, *Franzis*, Jg. 73, Nr. 1, S. 1–73, 2018. DOI: GTIN4019631670519.

[4] Kenneth D. Skeldon, Lindsay M. Reid, Viviene McInally, Brendan Dougan, and Craig Fulton. (1998). Physics of the Theremin. (Abrufdatum 03.01.2020), Adresse: https://pdfs.semanticscholar.org/159b/8f7ab33083fc1b8de584ec338b0ee2f6fd7b.pdf.

[5] Wikipedia. (2020). Cent (Musik). (Abrufdatum 16.08.2020), Adresse: https://de.wikipedia.org/wiki/Cent_(Musik)#:~:text=Das%20Cent%20(von%20lat.,Gr%C3%B6%C3%9Fen%20musikalischer%20Intervalle%20m%C3%B6glich%20ist..

[6] Theorie Musik. (2020). Pentatonik. (Abrufdatum 18.08.2020), Adresse: https://www.theorie-musik.de/tonleiter/pentatonik/.

[7] Wikipedia. (2020). Frequenzen der gleichstufigen Stimmung. (Abrufdatum 16.08.2020), Adresse: https://de.wikipedia.org/wiki/Frequenzen_der_gleichstufigen_Stimmung.

[8] Esteban O. Garcia. (2006). Paper Cordic Generator. (Abrufdatum 17.01.2020), Adresse: https://ccc.inaoep.mx/~rcumplido/papers/2006-Garcia-Pipelined%20CORDIC%20Design.pdf.

[9] Intel. (2007). Understanding CIC Compensation Filters. (Abrufdatum 17.01.2020), Adresse: https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/an/an455.pdf.

[10] ——, (2020). Embedded Design Handbook. (Abrufdatum 12.08.2020), Adresse: https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/hb/nios2/edh_ed_handbook.pdf.

[11] ILITEK. (2010). TFT LCD Single Chip Driver ILI9341. (Abrufdatum 12.08.2020), Adresse: https://cdn-shop.adafruit.com/datasheets/ILI9341.pdf.

[12] Analog Devices. (2004). Touch Screen Digitizer AD7843. (Abrufdatum 12.08.2020), Adresse: https://www.analog.com/media/en/technical-documentation/data-sheets/AD7843.pdf.

[13] Intel. (2020). Cyclone V Handbook. (Abrufdatum 14.08.2020), Adresse: https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/hb/cyclone-v/cv_5v2.pdf.

[14] ——, (2020). Nios® II Software Developer Handbook. (Abrufdatum 14.08.2020), Adresse: https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/hb/nios2/n2sw_nii5v2gen2.pdf.

[15] ——, (2017). Audio/Video Configuration Core. (Abrufdatum 17.01.2020), Adresse: ftp://ftp.intel.com/Pub/fpgaup/pub/Intel_Material/15.0/University_Program_IP_Cores/Audio_Video/Audio_and_Video_Config.pdf.

[16] Wolfson. (2012). Portable Internet Audio CODEC WM8731. (Abrufdatum 17.01.2020), Adresse: https://statics.cirrus.com/pubs/proDatasheet/WM8731_v4.9.pdf.

[17] Analog Devices. (2004). Touch Screen Digitizer. (Abrufdatum 14.08.2020), Adresse: https://www.analog.com/media/en/technical-documentation/data-sheets/AD7843.pdf.

[18] Autodesk Inventor. (2020). Software für mechanische Konstruktion und 3D-CAD. (Abrufdatum 18.08.2020), Adresse: `https://www.autodesk.de/education/edu-software/overview?sorting=featured&page=1`.

[19] Theorie Musik. (1985). High Speed Comparator Techniques. (Abrufdatum 19.08.2020), Adresse: `https://www.analog.com/media/en/technical-documentation/application-notes/an13f.pdf`.