Fachhochschule Nordwestschweiz
Hochschule für Technik

digitales Theremin
Fachbericht

Projekt 6
21. 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

This project is a continuation on the work on a Theremin that mainly operates on digital hardware. Unlike the original device, which 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 an appealing housing. Moreover the device should have other additional functionality that helps the player to use the instrument more easily. 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 were the oscillators that control the pitch and volume by use of two antennae and the codec, wich converts the audio data for use on a loudspeaker. The pitch and volume of the instrument can be changed by the two antennae like a usual theremin. To adjust settings the theremin has a touchscreen display that is controlled by the Nios system, that was implemented. The theremin has two functionalities, that help the player use the instrument more easily. First the glissando effect, wich corrects the pitch to the closest musical note. Second the pitch display, wich displays the closest musical note the player ist playing at and the deviation that the player is causing. It has two minor flaws. At higher frequencies aliasing is audible. And with certain volume settings a bit of noise is perceptible. In conclusion it can be said, that the theremin is ready for any trade fairs that will come.

Inhaltsverzeichnis

1 Einleitung **1**

2 Technische Grundlagen **2**

2.1 analoges Theremin 2

2.2 Musiktheorie 4

2.3 Cordic Algorithmus 5

2.4 CIC Filter 6

2.5 Goldschmidt Algorithmus 8

3 Konzept **9**

4 Realisierung **11**

4.1 Tonhöhen- und Lautstärkenoszillator 13

4.2 Clock 14

4.3 CPU 14

4.4 Tonhöhenverarbeitung 16

4.5 Lautstärkeverarbeitung 23

4.6 Audioserialisierer 26

5 Realisierung Software **27**

5.1 Hauptprogramm State Machine 27

5.2 Treiber 28

5.3 Audio 30

5.4 Touch 31

5.5 GUI 31

6 Realisierung Gehäuse **33**

7 Validierung **35**

7.1 Antennenoszillator PCB 35

7.2 Frequenzmessung 36

7.3 Glissando Effekt 37

7.4 Gesamttest 39

8 Schlusswort **40**

9 Danksagung **42**

10 Ehrlichkeitserklärung **43**

Literatur **44**

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, wegen seines unheimlichen Klangs [2].
Das Theremin wird ohne es zu berühren gespielt, indem der Spieler mit den Händen die Distanz zu zwei Antennen ändert. Dies führt zu Veränderung der Tonhöhe mit der einen Antenne und der Lautstärke mit der anderen Antenne. Die Antennen und der Spieler wirken dabei wie Kondensatoren, welche zwei Oszillatoren ganz leicht verstimmen. Da diese Oszillatoren mit etwa 500 kHz schwingen machen diese kleinen Änderungen eine Frequenzänderung im Bereich von mehreren kHz aus. Weitere Verarbeitung bringt diese Änderungen in den hörbaren Bereich.
Dass das Theremin über kontinuierliche Veränderungen der Tonhöhe gespielt wird ist eine typische Eigenschaft des Instruments.

Im Projekt 5 wurden die Grundlagen für ein solches Instrument entwickelt. Die Implementation des sonst analogen Instruments ist nun jedoch digital. Dabei ist es auf einem Field Programmable Gate Array (FPGA) implementiert. Das Theremin soll als Messeobjekt für das Institut für Sensorik und Elektronik ISE dienen, um die Möglichkeiten welche FPGAs bieten aufzuzeigen. Nach Ende des Projekt 5 war die Implementation, welche das Spielen über die Tonhöhenantenne ermöglicht weitestgehend abgeschlossen. Der Antennenoszillator war der einzige analoge Teil des Gerätes. Die Realisierung der restlichen Komponenten war in VHDL. Das Resultat wurde auf dem DE1-SoC Board von Terasic mit einem Cyclone V FPGA von Intel getestet. Quartus Prime diente als Entwicklungsumgebung.

Das Projekt 6 umfasste die abschliessende Entwicklung des digitalen Theremin. Die Lautstärke kann nun über eine zweite Antenne eingestellt werden. Es ist nun somit möglich das digitale Theremin wie ein normales Theremin zu spielen.
Weiter ist eine Bedienung des Theremin über ein LCD mit Touchscreen möglich. Der implementierte Nios II Prozessor ist dabei für diese Bedienung und die Steuerung der restlichen Komponenten zuständig.
Über das Display kann der Spieler eine Kalibration des Theremin durchführen, welche automatisch die Tonhöhe und Lautstärke richtig einstellt.
Für Spieler mit etwas weniger musikalischem Talent können zwei Spielhilfen aktiviert werden. Einerseits der Glissando-Effekt und andererseits die Anzeige der Spielgenauigkeit mit zwei verschiedenen Tonleitern. Der Glissando-Effekt hilft dem Spieler, indem er während dem Spielen auf den nächstgelegenen Ton korrigiert. Die Anzeige der Spielgenauigkeit zeigt dem Spieler wie genau der Ton getroffen ist und bei gewissen Einstellungen, wie nahe er an dem richtigen Ton ist. Das Gehäuse ist als 3D-Druck entstanden, um ein ausgefalleneres Design zu ermöglichen.

Die folgende Dokumentation beginnt mit den Technischen Grundlagen, wo als erstes beschrieben ist, wie ein Theremin funktioniert und anschliessend diverse Grundlagen welche für die Implementation nötig waren. Als nächstes folgen Kapitel zu der Realisierung der Hardware, Software und dem Gehäuse. Abschliessend folgt die Validierung des Theremin und eine Dokumentation der Ergebnisse in einem Schlusswort.

2 Technische Grundlagen

In diesem Kapitel ist als erstes erklärt wie ein analoges Theremin funktioniert um ein Verständnis für dessen Funktionsweise zu gewinnen. Anschliessend folgt ein 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 weitem Frequenzbereich kontinuierlich veränderbar ist. Das Theremin kann daher alle Frequenzen in einem Bereich spielen, im Gegensatz zu den meisten anderen Instrumenten.

Der Spieler spielt das Theremin durch verstimmen der Oszillatoren über die Antennen. Die Hand des Spielers verändert über die jeweilige Antenne die Schwingfrequenz des Tonhöhen und Lautstärkeoszillators. Dabei wird der kapazitive Anteil des LC-Schwingkreises 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 des Tonhöhenoszillators hörbar gemacht und danach verstärkt[3]. Der Lautstärkepegel ergibt sich durch die Verwendung eines Bandpassfilters und nachfolgenden Hüllkurvendetektor. Die Abbildung 2.1 gibt einen Überblick über die Schaltungskomponenten eines Theremins. Die einzelnen Schaltungsteile sind im folgenden Teil genauer erklärt.

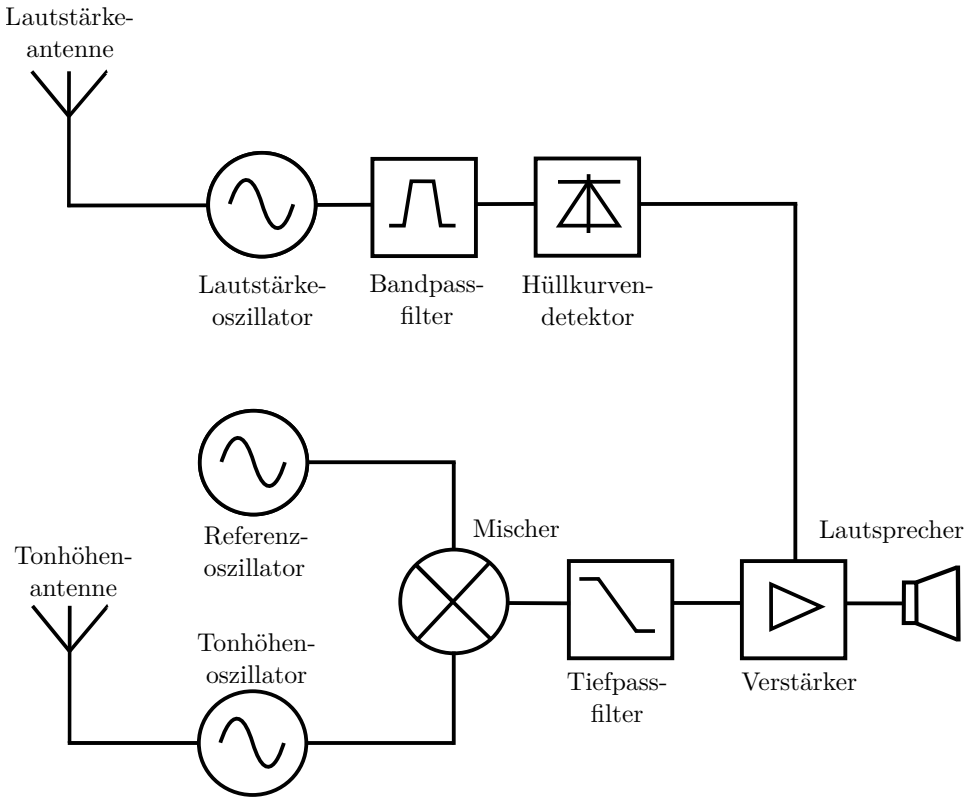


Abbildung 2.1: Blockschaltbild eines analogen Theremins

Tonhöhenoszillator und Tonhöhenantenne

Die Tonhöhenantenne ist ein Metallrohr welches mit dem Tonhöhenoszillator verbunden ist. Der Spieler kann über die Distanz seiner Hand zur Antenne die Frequenz des Tonhöhenoszillator verändern. Die über die Antenne zu erreichende Kapazitätsänderung ist sehr gering. Diese liegt im Picofarad Bereich [4].Die Grundfrequenz des Tonhöhenoszillator muss weit über dem hörbaren Bereich liegen, dass eine genügend grosse Frequenzänderung entsteht.

Lautstärkeoszillator und Lautstärkeantenne

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

Mischer und Referenzoszillator

Die erzeugte Frequenz der Tonhöhenantenne ist weit über dem vom Menschen hörbaren Bereich. Der Mischer multiplizeirt die Signale des Referenzoszillators und des Tonhöhenoszillators wie in Formel 2.1. $A_1 \sin(\omega_1)$ ist das Signal des Referenzoszillators und $A_2 \sin(\omega_2)$ das Signal des Tonhöhenoszillators.

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

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].

Eine Kalibration des Theremins ist vor jedem Gebrauch nötig. Es könnte beispielsweise sein, dass die Differenz der Frequenz ausserhalb des hörbaren Bereiches ist. Dazu stellt der Spieler beim klassischen Theremin mit Hilfe eines Trimmkondensators am Referenzoszillator die Differenzfrequenz auf 0 Hz ein.

Tiefpassfilter

Das Tiefpassfilter filtert die hochfrequente Komponente aus Formel 2.2 weg. Übrig bleibt die Differenz der Oszillatorfrequenzen. Dieser ist der interessante Anteil des Mischprozesses, da er im hörbaren Bereich ist.

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

Verstärker und Lautsprecher

Der Verstärker verstärkt das Ausgangssignal des Tiefpassfilter abhängig von der Spannung, welche vom Hüllkurvendetektor stammt.

2.2 Musiktheorie

Um besser an einem Musikinstrument arbeiten zu können ist es wichtig ein wenig Musiktheorie zu kennen. Der wichtigste Fakt ist, dass unser Gehör den Schallpegel logarithmisch wahrnimmt und in Dezibel (dB) angegeben 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 ansteigen, desto höhere Frequenzunterschiede entstehen zwischen ihnen. Für einfacheres Rechnen von diesen Unterschieden ist die Masseinheit Cent gebräuchlich. 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 diese “akustische” 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 das Gehör zwei aufeinanderfolgende Töne mit etwa 6 Cent unterschied registriert. Jedoch ist es schwierig zu sagen wann das Gehör einen Ton als ”nicht getroffen“ empfindet.[5]

Ein weiteres interessantes Thema ist die Pentatonik. Dabei handelt es sich um ein Tonsystem mit nur 5 Tönen. Ein gutes Beispiel dafür sind die schwarzen Tasten des Klavier. Benützt der Spieler nur diese Tasten, spielt er in einem pentatonischen Tonsystem. 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. Weit verbreitet ist dafür der Cordic Algorithmus. 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. Der Algorithmus besitzt zwei Modi. Zum einen der Vektor Modus, in welchem die Berechnung eines Winkels aus einem gegebenen Vektor möglich ist. Zum Anderen der Rotationsmodus, mit welchem die Berechnung der Elemente eines Vektors aus einem gegebenen Winkel möglich ist. Die folgenden Formeln sind für diese Berechnung notwendig [8]:

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

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

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

x_i und y_i sind dabei die Elemente des Vektors und z_i ist der Winkel des Vektors. x_{i+1},y_{i+1} und z_{i+1} sind die Resultate einer Iteration. Die Berechnung von d_i im Rotationsmodus lautet wie folgt:

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

Formeln 2.5 bis 2.7 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 Dezimations-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.

In Abbildung 2.2 ist zu sehen, dass das Filter in drei Stufen unterteilt ist. Links ist ein Integrator-Filter zu sehen, welches einen anliegenden Wert mit einem verzögerten Wert aufaddiert oder anders gesagt das Signal integriert. Anschliessend folgt ein Dezimierer, welcher das Signal um den Faktor R unter-abtastet. Zuletzt folgt ein Combfilter. Dieses nimmt den aktuellen Wert und subtrahiert den alten Wert. Ein Interpolation-CIC-Filter erhält man durch tauschen von Integrator-Filtern mit Comb-Filtern und umgekehrt. Weiter ist nun in der Mitte eine Überabtastung nötig.
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 [10]. In Abbildung 2.3 sieht man sehr schön, dass sehr auf die Frequenz des Signals im Vergleich zum verwendeten CIC-Filter geachtet werden muss.

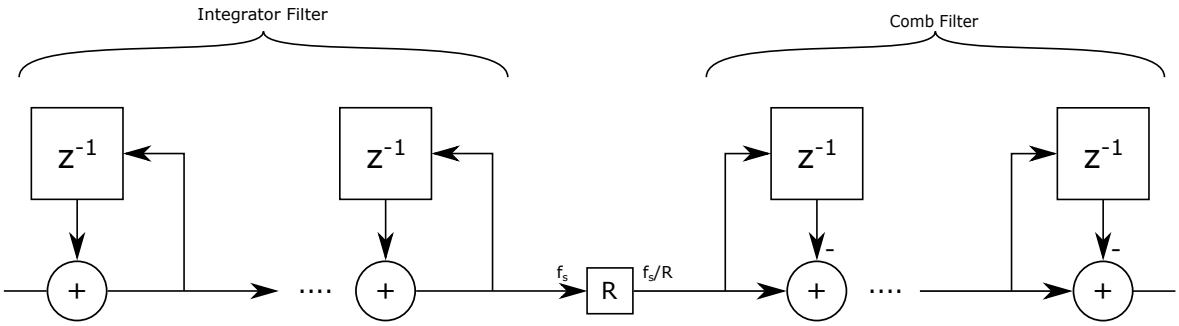


Abbildung 2.2: Aufbau eines CIC-Filters N-ter Ordnung mit Eingang mix_out und Ausgang filt_out

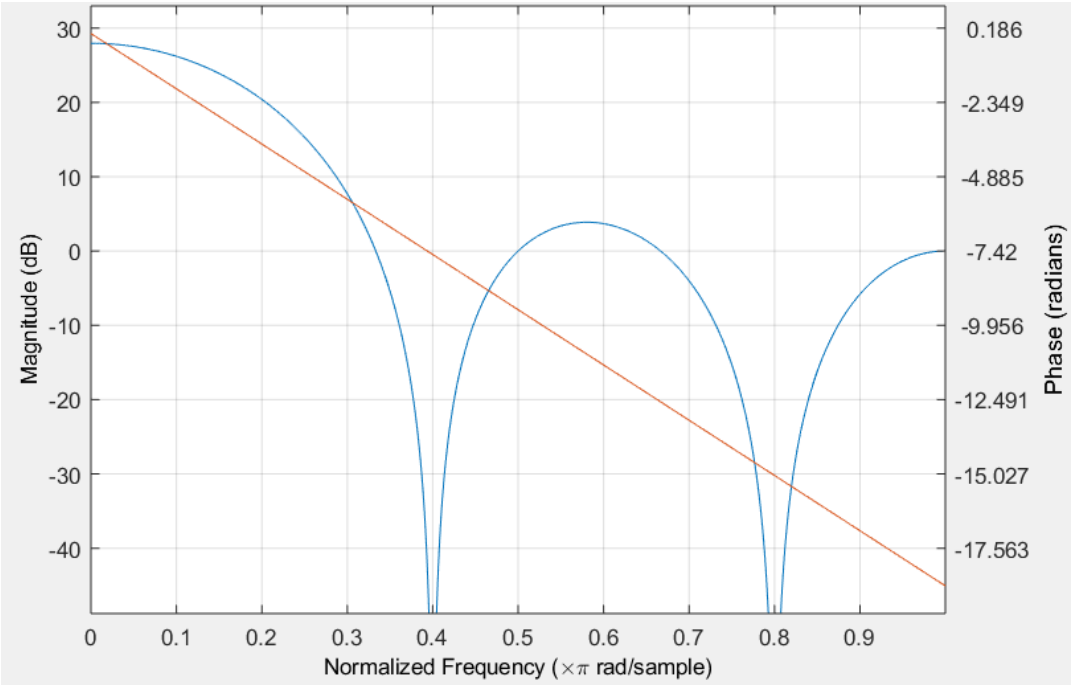


Abbildung 2.3: Amplitudengang und Phasengang eines CIC-Filters (M = 1, R = 5; N = 2)

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. Dieser 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_i.

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

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

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

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

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

N_i ist der Nenner, Z_i ist der Zähler und F_i ist der zuvor erwähnte Faktor der aktuellen Iteration. N_{i+1},Z_{i+1} und F_{i+1} sind die Resultate einer Iteration.

Damit der Algorithmus richtig funktioniert 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, da kein Analog-Digital-Wandler nötig ist. Weil der Referenzoszillator weiterhin einen Sinus generiert, ergibt die Mischung mit dem Rechteck auch Mischprodukte mit dessen Oberwellen. Da diese aber eine höhere Frequenz haben, ist eine spätere Filterung möglich.
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 den Sinus des Referenzoszillators mit dem Rechteck des analogen Oszillators. Auch diese Komponente stammt aus dem Projekt 5
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 Lautstärkenverarbeitung bei diesem Aufbau gleich wie der Tonhöhenverarbeitung. Dies haben wir so entschieden, um dieselben Komponenten nochmals nutzen zu können.
Um das Audiosignal zu verstärken,benötigt der Verstärker die Frequenzinformation der Lautstärkenverarbeitung. Diese erhält er vom Block Frequenzmessung. Da die Frequenz des Lautstärkeoszillators bei Veränderung der Distanz zu der Antenne exponentiell ändert, ist keinerlei Umrechnung nötig um eine exponentielle Lautstärkeänderung zu erzielen. Anschliessend konvertiert der Digital-Analog-Wandler das verstärkte Audiosignal und gibt es am Lautsprecher aus.
Wir entschieden zudem ein Nios II System einzusetzen um das Theremin zu bedienen und zu steuern. Dies hauptsächlich, um einen Einblick in den Nios II zu gewinnen und um eine Implementation der Steuerung zu vereinfachen. 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. Diese soll die verschiedenen Oszillatoren so aufeinander abstimmen, dass eine Annäherung an die Antennen eine Erhöhung der Tonhöhe und Lautstärke bewirkt.

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 II 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.

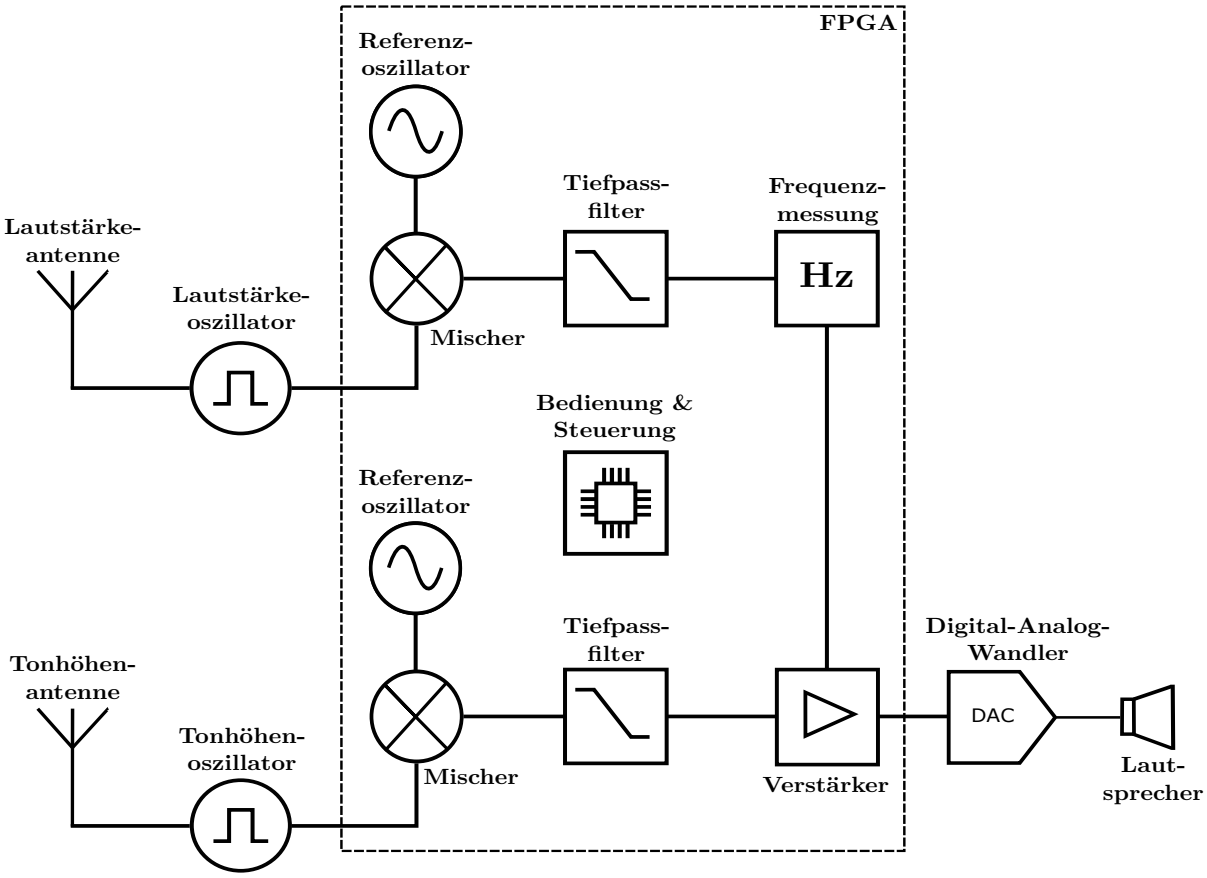


Abbildung 3.1: Blockschaltbild des digitalen Theremins

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 *Lautstärke-* und *Tonhöhenverarbeitung*, *DC-FIFO* und dem *Audioserialisierer*. Der zweite Bereich ist Das Nios II 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 und das Spielen des Theremin ermöglichen.
Die Kommunikation zwischen dem Nios II 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 *Tonhöhenverarbeitung* und *Audioserialisierer*.
Die Clocks, welche zu den verschiedenen Komponenten gehen sind in Abbildung 4.1 für eine bessere Übersichtlichkeit weggelassen worden. Für eine Liste aller Clock Frequenzen und deren Ziel siehe Kapitel 4.2.

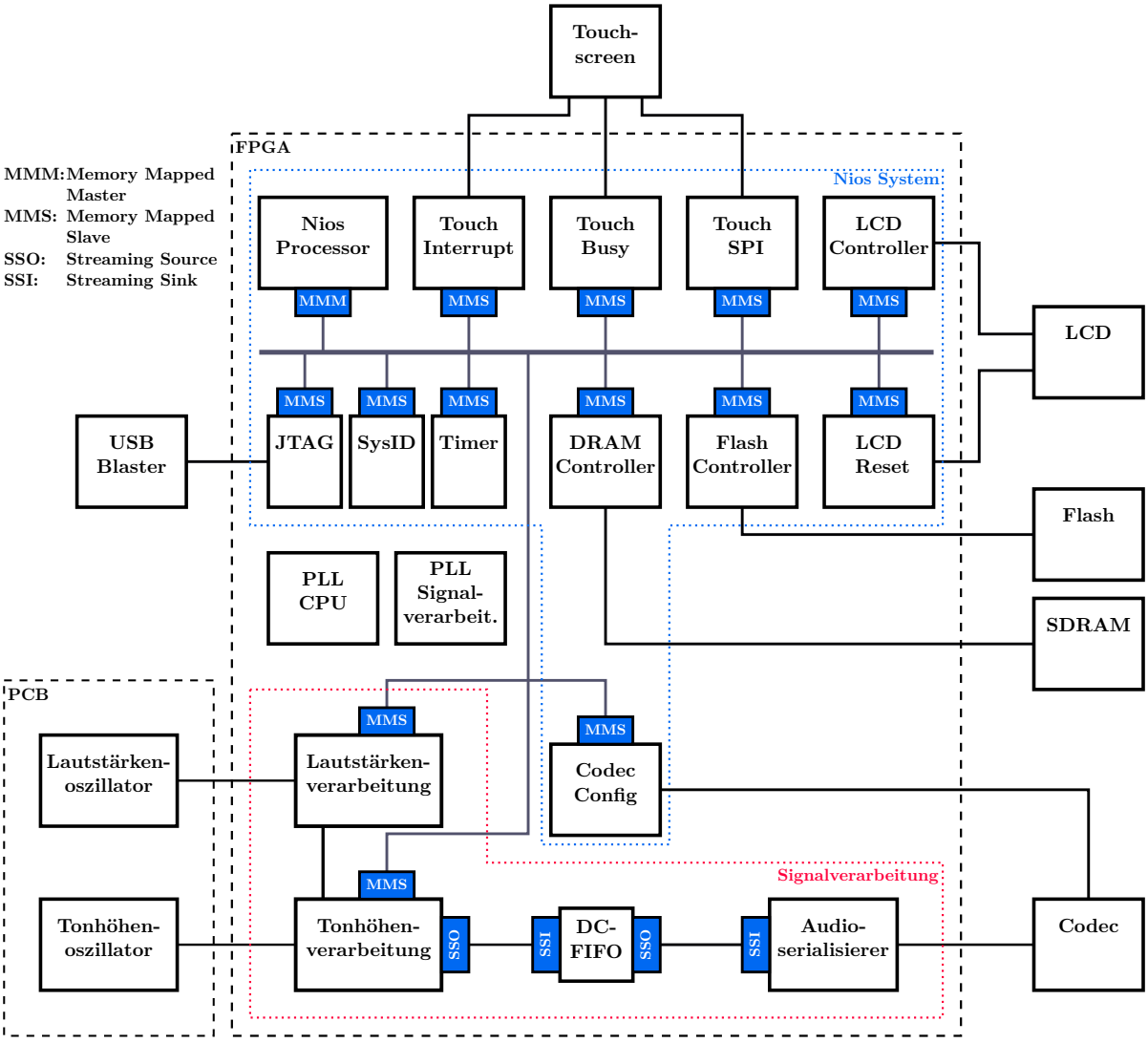


Abbildung 4.1: Blockschaltbild gesammtes Theremin

4.1 Tonhöhen- und Lautstärkenoszillator

Für die Antennenoszillator Schaltung haben wir uns im Projekt 5 für den Colpitts-Oszillator aus Abbildung 4.2 entschieden. Der Aufbau im Projekt 5 umfasste nur einen Oszillator zur Veränderung der Tonhöhe.

Es handelt sich dabei um einen Colpitts-Oszillator 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, war ein Wechsel auf den J113 N-Kanal JFET nötig. Die mit LTspice simulierten Werte des J113 glichen stark der original Schaltung, weshalb der Entscheid auf diesen fiel. 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. Dieser ist mit 3.3 V betrieben da die Logikeingänge des FPGA auf diese Spannung ausgelegt sind.

Im Projekt 5 wurde als Antenne ein Messing Rohr 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.

Auf dem PCB sind nun im Projekt 6 zwei solche Oszillatoren verbaut: Der Tonhöhenoszillator und der Lautstärkenoszillator. Das PCB 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 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.

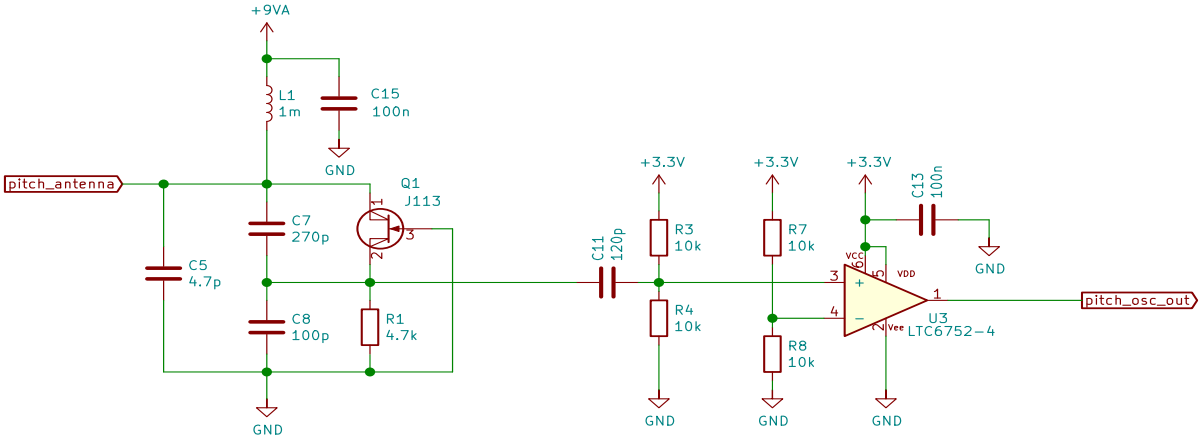


Abbildung 4.2: Schema Antennenoszillator. Links Collpitts-Oszillator,rechts Komparatorschaltung

4.2 Clock

Die verschiedenen Clocks für die Hardwarekomponenten und die CPU werden in zwei Phase-Locked-Loop (PLL) Blöcken generiert. Die Generierung der verschiedenen Clock Signale für die Hardwarekomponenten und die CPU geschieht in zwei Phase -Locked-Loop (PLL) Blöcken. Ein Block für die Signalverarbeitung und einer für das Nios II System. In Tabelle 4.1 sind alle Frequenzen aufgelistet.
Die Clockfrequenzen, welche an Komponenten für die LCD Steuerung, den Flash Controller und die Audio Konfiguration sowie Übertragung gehen sind Vorgaben der jeweiligen Peripherien. Weiter benötigen die Komponenten Tonhöhen- und Lautstärkenverarbeitung die Frequenz 54Mhz, da deren Frequenz ein vielfaches von 48kHz sein muss.

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
Codec Config	12 MHz	PLL CPU
Flash Controller	25 MHz	PLL CPU
Tonhöhenverarbeitung	54 MHz	PLL Signal-Processing
Lautstärkenverarbeitung	54 MHz	PLL Signal-Processing
DC-FIFO Input	54 MHz	PLL Signal-Processing
DC-FIFO Output	24 MHz	PLL Signal-Processing
Audioserialisierer	24 MHz	PLL Signal-Processing
Codec	12 MHz	PLL Signal-Processing

4.3 CPU

Der eingesetzte Nios II 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 ist nötig um beim Laden der Software sicherzustellen, dass das passende Hardware Image vorhanden ist. Alle drei Komponenten sind mit Standardeinstellungen in das Nios II 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 bereits auf dem Entwicklungsboard vorhanden ist, haben wir uns gegen On-Chip Speicher entschieden um Ressourcen zu sparen.

Das Hardware Image und der Programmcode ist auf dem Board enthaltenen Flash Speicher gespeichert. Dabei lädt Quartus anders als bei dem nicht flüchtigen Programmieren nicht das SRAM Object File (.sof) sondern ein JTAG Indirect Configuration File (.jic). Das Erstellen dieses Files geschieht im Quartus aus dem SRAM Object File und dem in Eclipse generierten HEX File. Der USB Blaster lädt das .jic File über ein Serial Flash Loader Image auf den Flash Speicher. Das FPGA kopiert beim Einschalten des Gerätes zuerst das Hardwareimage und anschliessend den Programmcode. 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. [11]

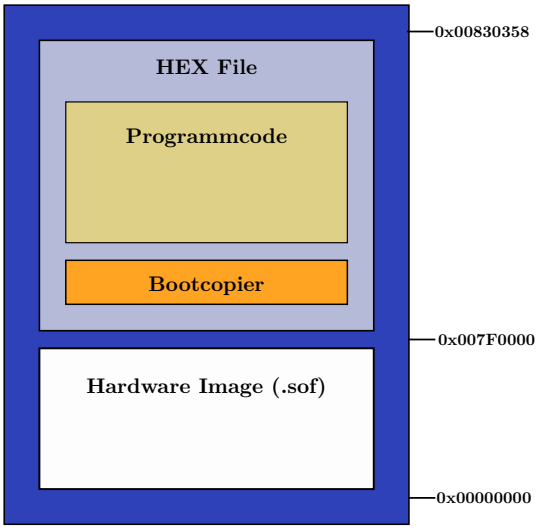


Abbildung 4.3: Layout des Flash Speichers

LCD Controller & Reset

Für das beschreiben des LCD ist die von Terasic bereitgestellte VHDL Komponente LT24_Controller zuständig. Der Nios II Prozessor steuert diese über das Memory Mapped Interface. 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. [12]

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 II über eine Betätigung des Touchscreens informiert. Bei einer Berührung des Touchscreens löst *Touch Interrupt* beim Nios II Prozessor einen Interrupt aus, welcher sofort die Koordinaten über SPI anfordert. [13]

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.

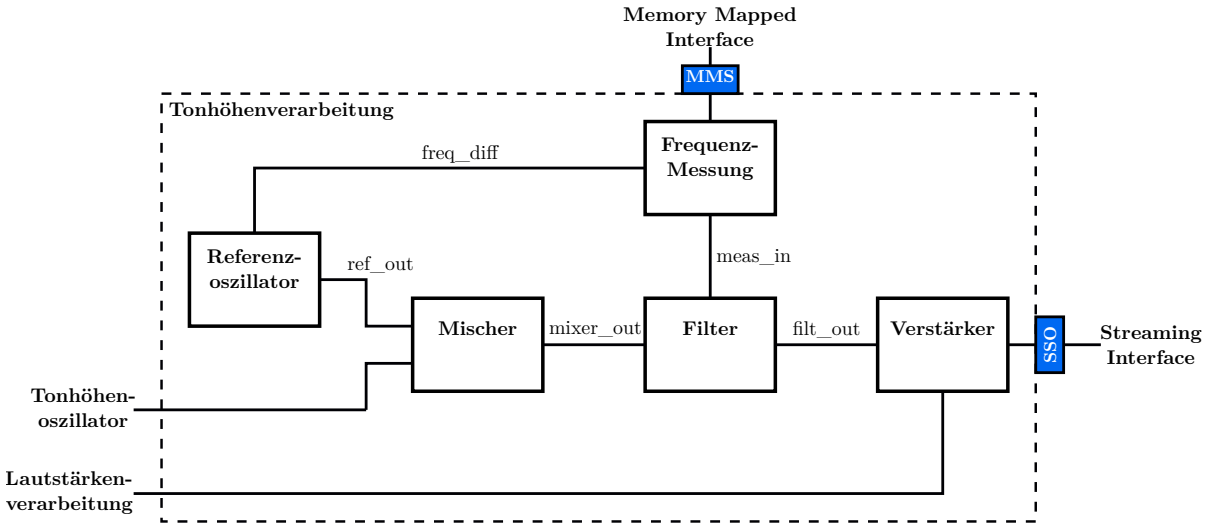


Abbildung 4.4: Blockschaltbild der Custom IP *Tonhöhenverarbeitung*

Referenzoszillator

Der *Referenzoszillator* ist wie in der analogen Version aus Kapitel 2.1 dafür zuständig ein Sinussignal mit einer Frequenz nahe der des Tonhöhenoszillators 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

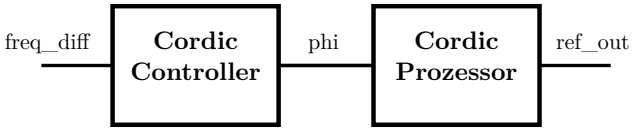


Abbildung 4.5: Aufbau des Referenzoszillators

stammen aus dem Projekt 5. Änderungen, welche im Projekt 6 stattfanden, sind entsprechend gekennzeichnet.

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 $\arctan 2^{-i}$ fand vorgängig statt 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 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 konvertiert 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]

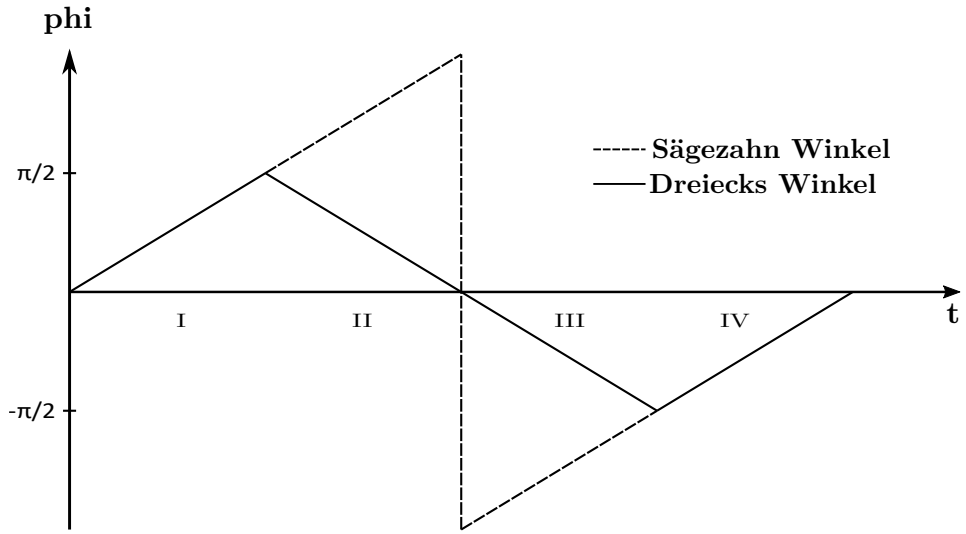


Abbildung 4.6: berechneter Winkel *phi* des Cordic Controllers in Funktion der Zeit

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.

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 stammt weitestgehend aus dem Projekt 5, ist jedoch auf mehr Modularität erweitert. 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 erhält man mit Formel 2.11 in Kapitel 2.4.
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

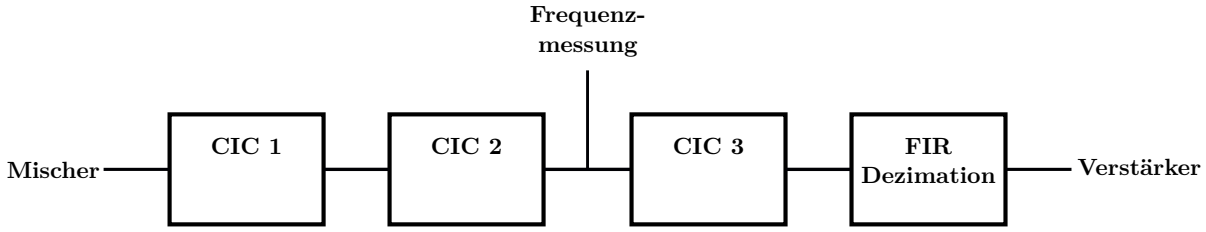


Abbildung 4.7: Aufbau des Filters in der Komponente Pitch Generation

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

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. [14] Weswegen der Ausgang Frequenzmessung nach dem zweiten CIC-Filter gewählt wurde ist in einem späteren Abschnitt beschrieben.

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 ist zu sehe, 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 ändern. 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.

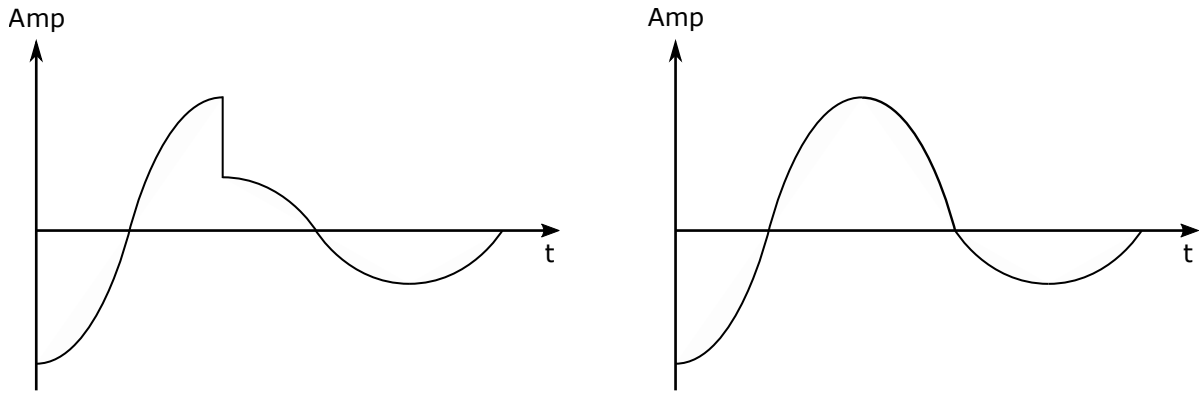


Abbildung 4.8: Unterschied Dämpfungswechsel (links ohne und rechts mit Nullstellenerkennung)

Frequenzmessung, Kalibration & Glissandoeffekt

Die Komponente *Frequenzmessung* hat mehrere Aufgaben und ist diejenige Komponente mit welcher über den Nios II Prozessor die gesamte *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 den CIC-Filtern, welches noch hochfrequente Anteile enthält zu filtern. Das FIR Filter hat eine Passfrequenz von 2 kHz eine Stopfrequenz von 40 kHz und eine Dämpfung von 30dB. Das Filter ist 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. [14]
Anschliessend wird das Signal *fir_out* im *Periodenzähler* ausgemessen. Dieser 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.

Die gemessene Frequenz wird anschliessend in der Komponente *Kalibration & 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 verstimmt die Komponente den Referenzoszillator entsprechenden. Kalibration & Glissando addiert die Differenz welche den Glissandoeffekt hervorruft und die Änderung welche während der Kalibration berechnet wurde und übergibt das Resultat dem Referenzoszillator. Diese wird von nun an als Frequenzdifferenz bezeichnet. Die Steuerung dieser Funktionen ist als State-

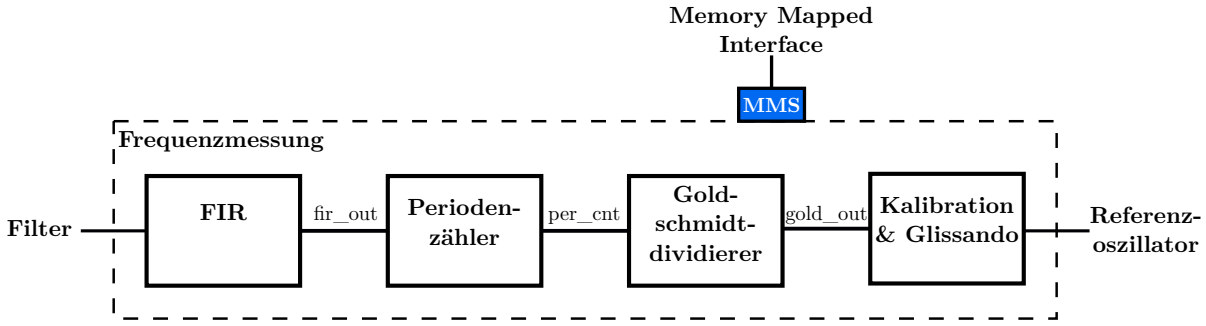


Abbildung 4.9: Aufbau der Frequenzmessung, Kalibration und Glissandoeffekt in der Komponente Pitch Generation

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 abgeschlossen ist.

check: Da die Messung Frequenzen unter 100 Hz immer als 100 Hz angibt, muss die Komponente diese erkennen. 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. Dies, da die Frequenz des Tonhöhenoszillators mit kleinerem Abstand immer mehr sinkt. Anschliessend würde sie wieder steigen, da der Tonhöhenoszillator kleiner ist als der Referenzoszillator. 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 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 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.

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 das Aufschliessen bevor eine neue Messung stattfand einen Unterschied von unter 6 Cent zu der anzunähernden Frequenz, stoppt das Zählen bis zu einer neuen Messung.

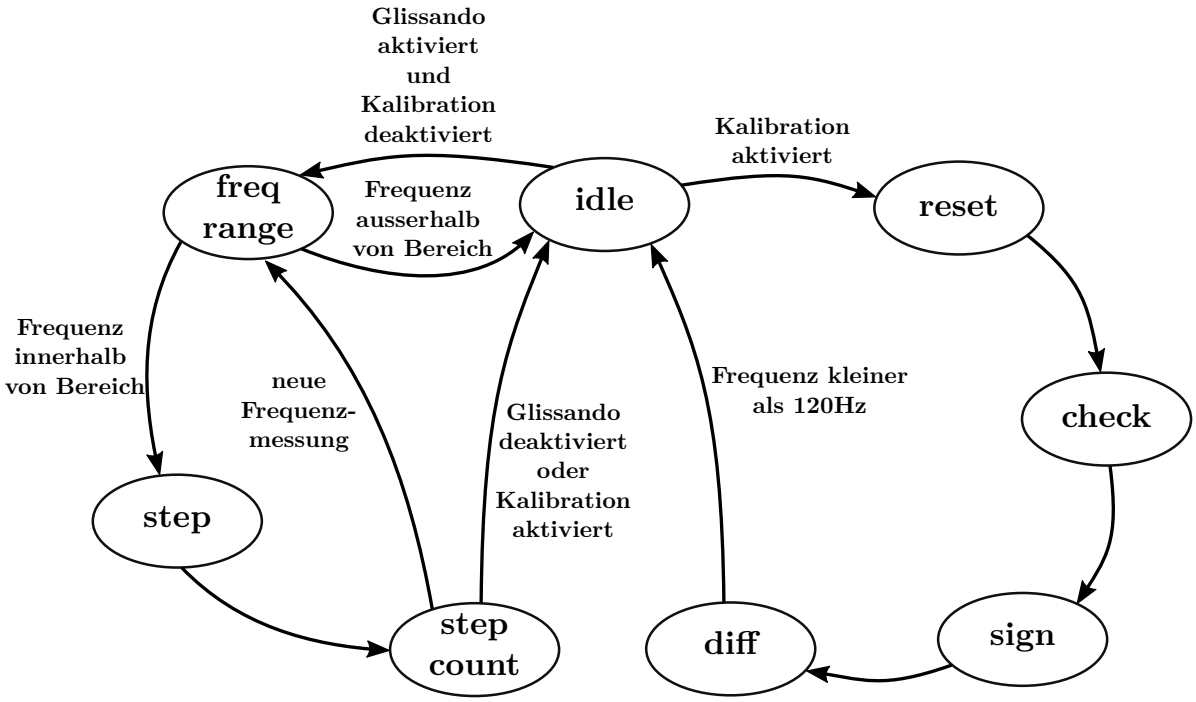


Abbildung 4.10: State Event Diagramm der Kalibration & Glissando Komponente

Register

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

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 verändern.

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.

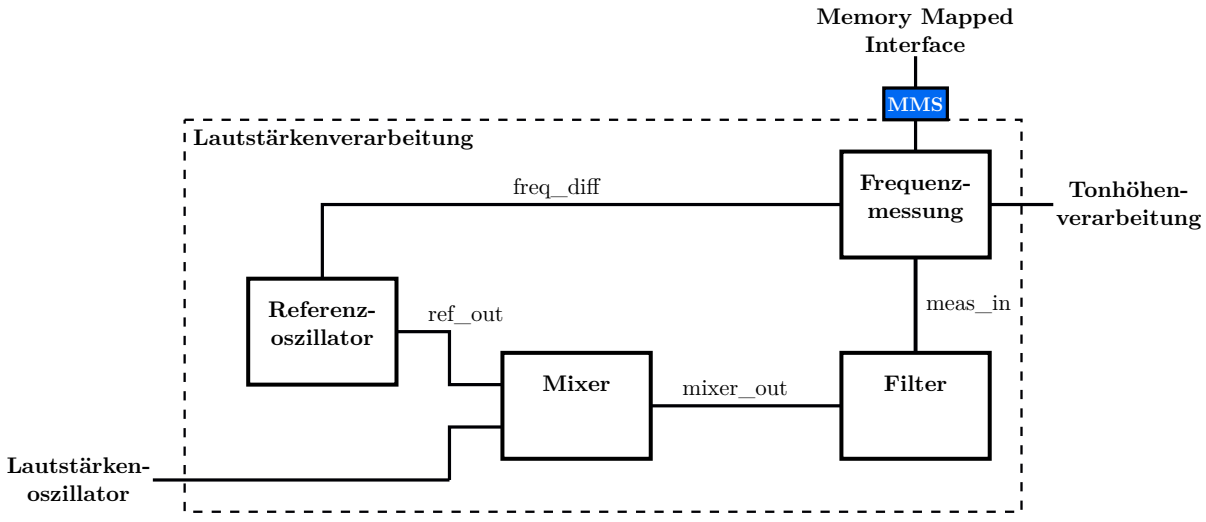


Abbildung 4.11: Blockschaltbild der Custom IP Volume Generation

Filter

Beim *Filter* der Laustsärkeverarbeitung ist die Frequenzmessung erst nach dem dritten CIC Filter angeschlossen. Wir haben dies so entschieden, 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 deswegen nicht mehr nötig.

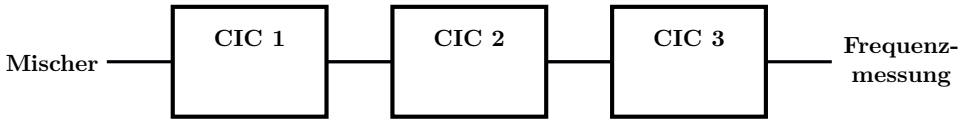


Abbildung 4.12: Aufbau des Filters in der Komponente Volume Generation

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 problematisch ist, verglichen zu der 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 den in Kapitel 4.4 besprochenen Dämpfungsfaktor zu berechnen und an die Tonhöhenverarbeitung weiterzugeben. 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 exponentiell ansteigt, muss keine Umrechnung stattfinden um die Eigenschaft des logarithmischen Gehörs zu kompensieren.

Anders als bei 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.

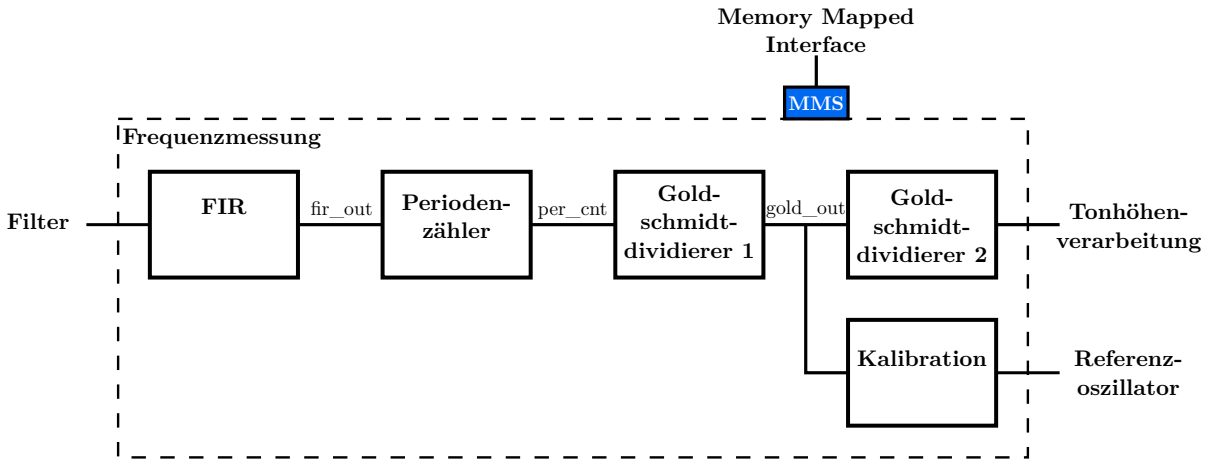


Abbildung 4.13: Aufbau der Frequenzmessung und Kalibration in der Komponente Volume Generation

Register

Um mit der Komponente *Lautstärkenverarbeitung* über das Memory-Mapped Interface kommunizieren zu können haben wir 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. Der zur Verfügung gestellte IP Core benötigt eine spezifische Clockfrequenz von 12.288 MHz. Jedoch müssen die Tonhöhenverarbeitung und der Serialisierer Clocks erhalten die denselben Referenzclock besitzen, um nicht Daten zu verlieren. Jedoch können die verlangten Clockfrequenzen von 54 MHz für die Tonhöhenverarbeitung und die zuvor genannten 12.288 MHz nicht in einem PLL generiert werden.
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 von DACLRC lädt der Serialisierer einen neuen Audiosignalwert ins Schieberegister und schiebt bei jeder negativen Flanke des BCLK ein neues Bit ins Signal DACDAT.

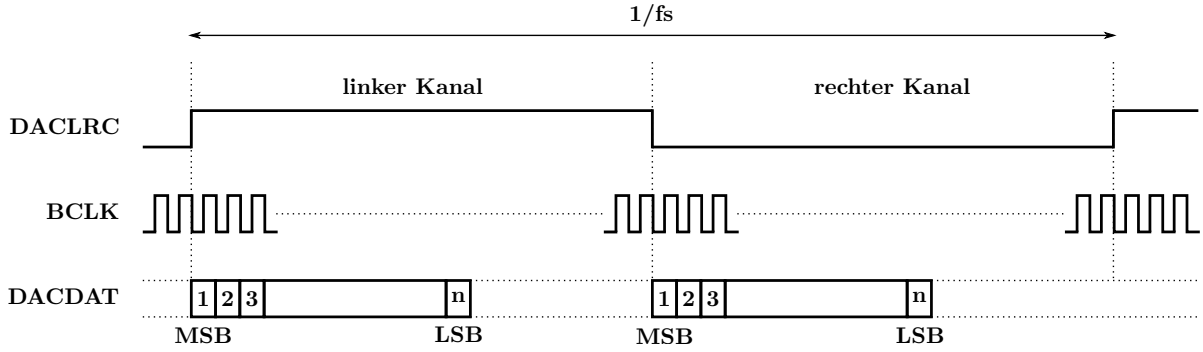


Abbildung 4.14: Signalverlauf des Codec Interface

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 programmierte State Machine realisiert. Die Bedienung geschieht über das LT24 LCD Touch Modul 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 links die Initialisierung und rechts den Hauptprogrammfluss der Software als State Machine.

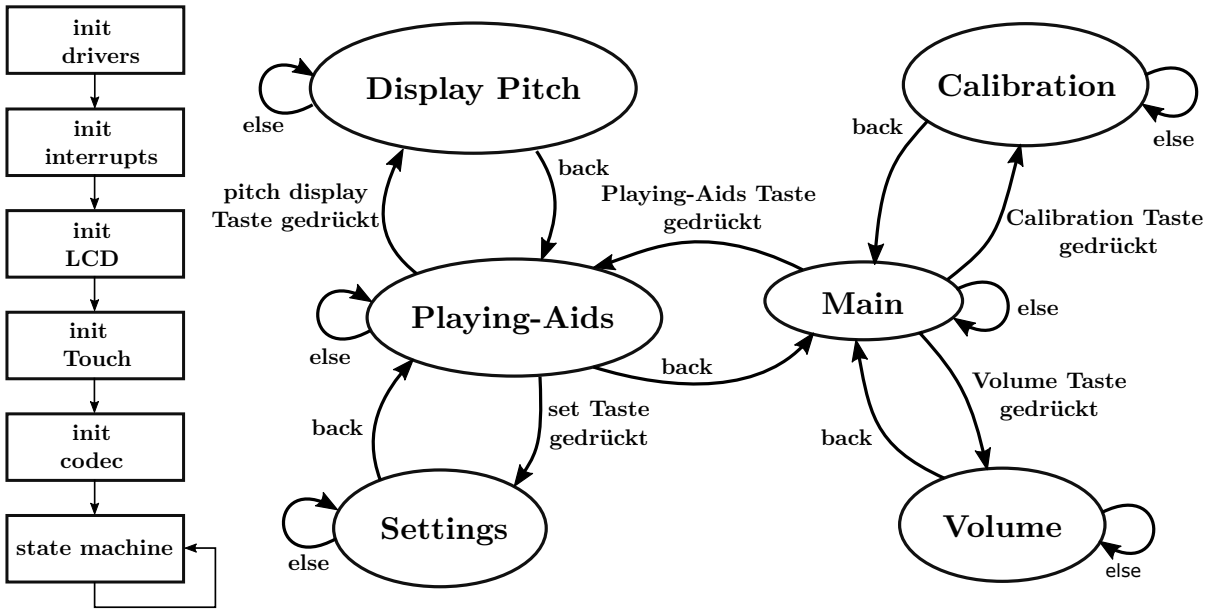


Abbildung 5.1: State Machine und Initailisierung.

Als erstes initialisiert das Programm alle Treiber, den Touchscreen, das LCD, und den Codec. Anschliessend führt das Programm in einer Endlosschlaufe die State Machine aus. Es folgen kurze Erklärungen zu jedem State.

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 den Glissando-Effekt aktivieren und deaktivieren. 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 am Glissando-Effekt gemacht werden. Der Delay des Glissando-Effekts ist in 10 Stufen einstellbar. Zudem kann der Anwender mit dem Taster *pentatonic on/off* zwischen der pentatonischen 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 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 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 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 hilfreich den vertikalen Strich anzuzeigen. Daher haben wir uns entschieden diese Anzeige nur bei der pentatonischen Tonleiter zu verwenden.

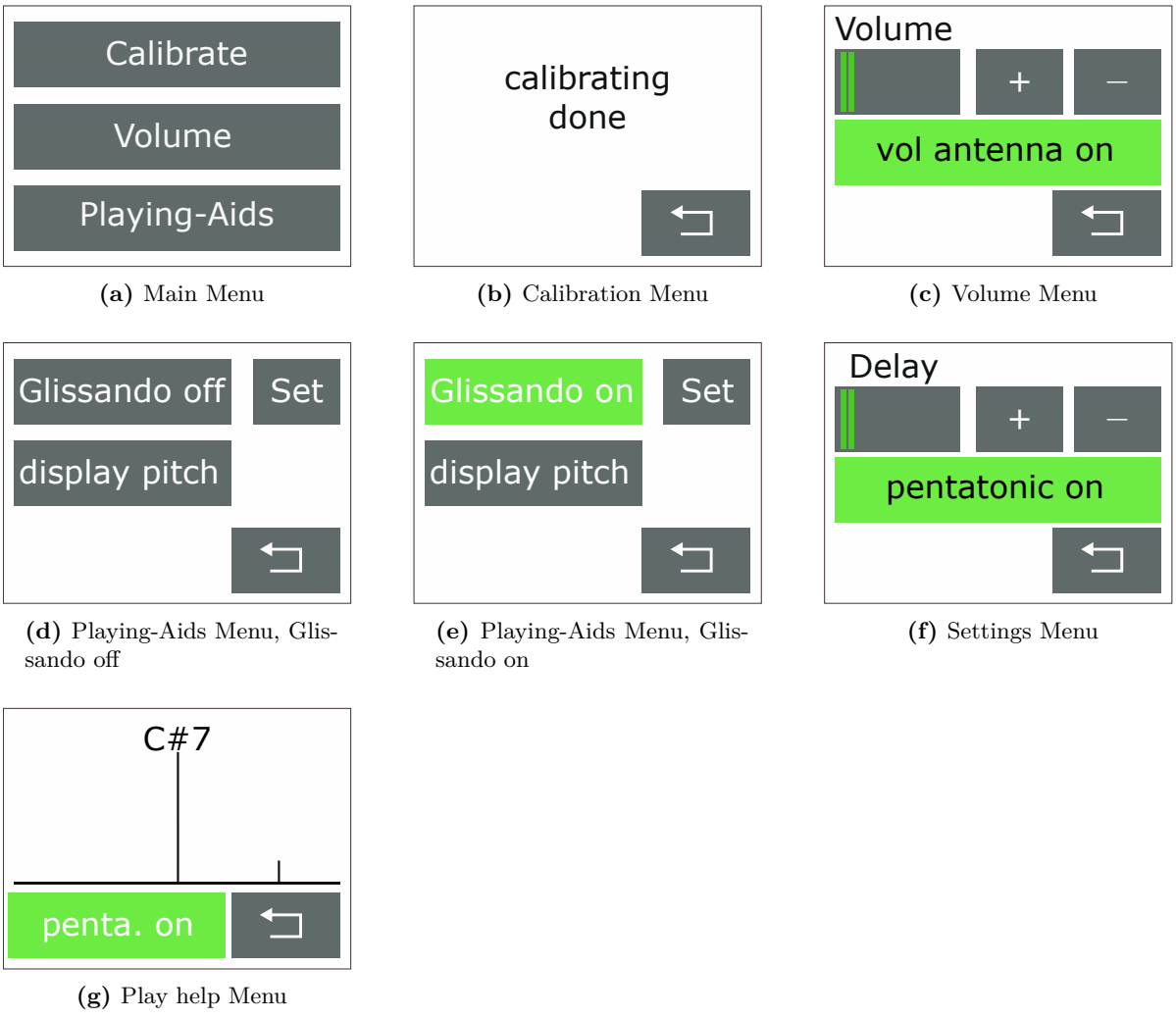
5.2 Treiber

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

Um ein selbst erstellten Treiber dem Board Support Package (BSP) hinzuzufügen, müssen die Benennungen und Ablageorte der Files einige Bedingungen erfüllen. Die Treiber Dateien müssen in dem Ordner IP abgelegt sein, welcher sich im Quartus Projekt 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 [15]. 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 es für die Steuerung des LCD gibt.

Das Modul *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, war eine effizientere Lösung nötig. 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 aufgespannt und danach alle Pixel eingefärbt. Da durch das Cursor Feld nicht für jedes einzelne Pixel ein Cursor gesetzt werden muss, wird Zeit eingespart.

Tonhöhenverarbeitung

Das Modul *Pitch_generation.c* ermöglicht auf die Register aus Kapitel 4.4 zuzugreifen. Die Funktion *get_pixel_pitch_accuracy(penta_on_off,pitch_freq)* liesst das freq_data Register aus. Dessen Wert wird für die Anzeige der Spielgenauigkeit in Abbildung 5.2g benötigt. Die Funktion berechnet aus dem gemessenen Frequenzwert und dem anzunähernden Ton einen Pixelwert um den senkrechten Strich auf dem Display einzuzeichnen.

Lautstärkenverarbeitung
Die einzige Kommunikation welche die Lautstärkeverarbeitung Komponente mit dem NIOS II hat, ist das schreiben und lesen des Kontroll Registers um die Kalibrierung zu starten und um die Bedienung über die Antenne zu aktivieren oder deaktivieren. Das Modul *Volume_generation.c* ermöglicht dies.

5.3 Audio

Die Kommunikation mit WM8731 Codec geschieht über die IP Komponente Audio/Video Configuration Core von Intel. Diese besitzt die Funktion *alt_up_av_config_write_audio_cfg_register* um die Kontrollregister des Cores über die Hardware Abstraction Layer (HAL) anzusprechen[16]. Das Modul **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 Kommunikation mit der Audioserialisierer Komponente möglich. Um Störungen zu vermeiden sind die Line In Eingänge des Codec stumm geschaltet, da nur der Line Out nötig ist. Der Linke und Rechte Kanal sind so eingestellt das beide Kanäle die selben Lautstärken haben[17].

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 Pegel um 76 db. Jeweils eine Stufe grösser reduziert die Dämpfung um 7 db. Die höchste Stufe dämpft den 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 und Rauschen verursacht.

Ursprünglich war geplant die Lautstärke des Theremin über den Codec zu steuern. Nicht wie in Kapitel 4.5 beschrieben über die VHDL Komponente. Diese Methode konnte jedoch nicht realisiert werden, da die Zero-Cross Detection des Codec nicht funktionierte, obwohl das entsprechende Flag gesetzt war. 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.

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 das Projekt anzuwenden. Daher entschieden wir uns selbst ein Touch Interrupt Routine zu erstellen.

Der resistive Touch Display des LT24 LCD Touch Modul ist mit dem AD783 Analoge Devices Chip verbunden. Sobald der Touch Display berührt wird löst der Chip am Pen Interrupt Pin ein Interrupt aus welches vom Nios II detektiert wird. Darauf fordert der Nios II eine Messung der X und Y Koordinaten an. Diese können wiederum über den SPI Bus vom Prozessor ausgelesen werden[18].

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 Service Routine *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 an den Rändern der Buchstaben durchführt. Dabei wird die Schriftfarbe mit der Hintergrundfarbe gemischt. Dieser Prozess nimmt viel Zeit in Anspruch und hat wenig Nutzen. Wodurch bei jedem neuen Zeichnen eines Texts zugeschaut werden konnte wie die Pixel gezeichnet wurden. Dies ist für unser Projekt unbrauchbar. Zudem war eine ansprechendere Schriftart nötig. Daher entschieden wir uns eine eigene Funktion zu schreiben welche Texte zeichnet. Diese ist im Modul **simple_text.c** realisiert.

simple_text.c

Mit dem Open Source Tool "The Dot Factory"liess sich eine Bitmap für die Schrifftart Arial generieren, welche 22 Punkte gross gewählt ist. Das Tool generiert zwei Arrays. Im Bitmap Array sind die Buchstaben in einer Bitmap gespeichert. Das Descriptor Array enthält Informationen über die Breite jedes Characters und den Offset in der Character Bitmap. Um den Offset eines Buchstabens 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 a, b und c. 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

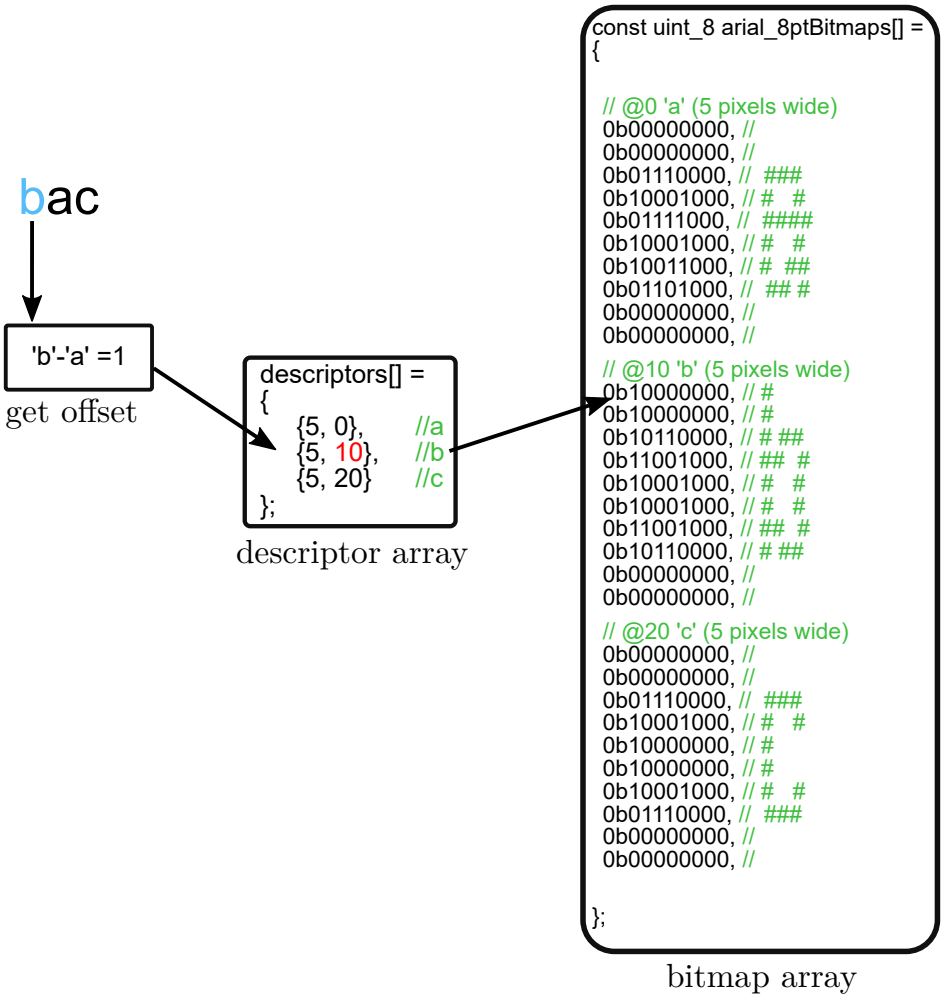


Abbildung 5.3: Bitmap und Discriptor Array.

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.

6 Realisierung Gehäuse

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

Die Anforderungen an das Gehäuse des digitalen Theremin sind:

- Die Lautstärke- und die Tonhöhenantenne müssen genügend Abstand zueinander haben, damit der Spieler das Theremin richtig spielen kann.
- Das Antennenoszillator PCB und das DE1-SoC Board müssen für den Spieler ersichtlich sein.
- Für die Bedienung muss das LT24 Touch Modul 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, soll das Gehäuse 3D-gedruckt sein.

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 mit dem die ausgefallene Form des Gehäuses leicht zu realisieren war. 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 $330\,\text{mm} \times 240\,\text{mm} \times 300\,\text{mm}$. Das Gehäuse musste jedoch in vier Teile unterteilt werden, damit es im Drucker platz hatte.

Wir entschieden uns die Form des Theremin oval zu gestalten, da andere kommerziell erhältliche Theremin eine solche Form haben. Die vier Einzelteile des Gehäuses sind alle gleich aufgebaut. Abbildung 6.1 zeigt die Grundstruktur eines Einzelteil. Die Funktion Wandung von Inventor 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 wasserlösslich ist.

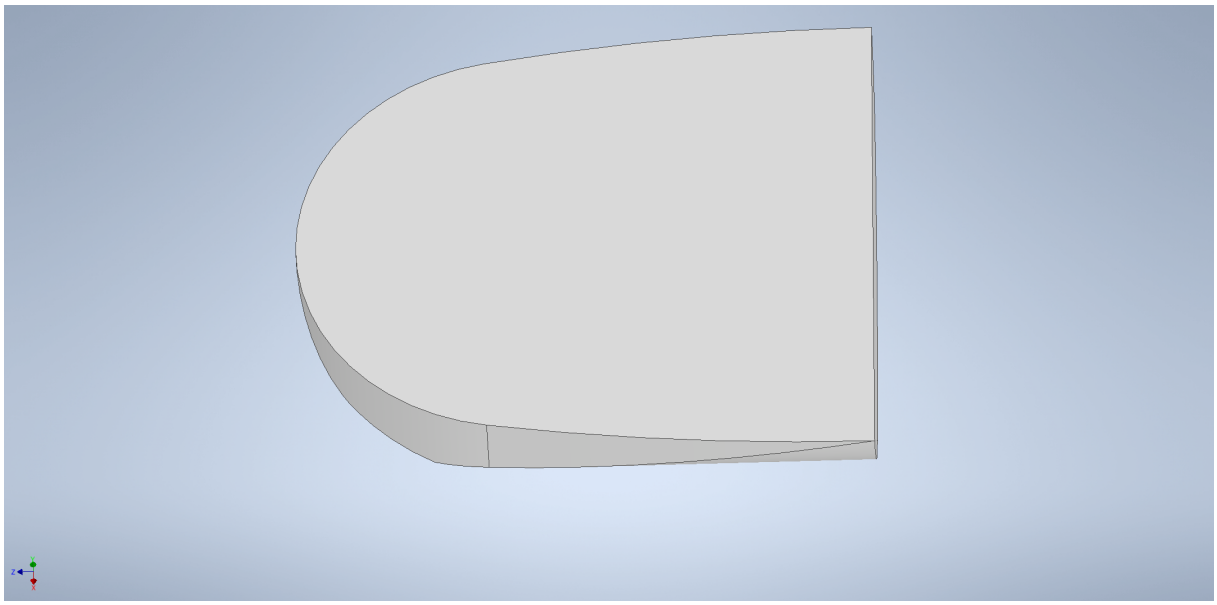


Abbildung 6.1: Grundstruktur eines Einzelteils.

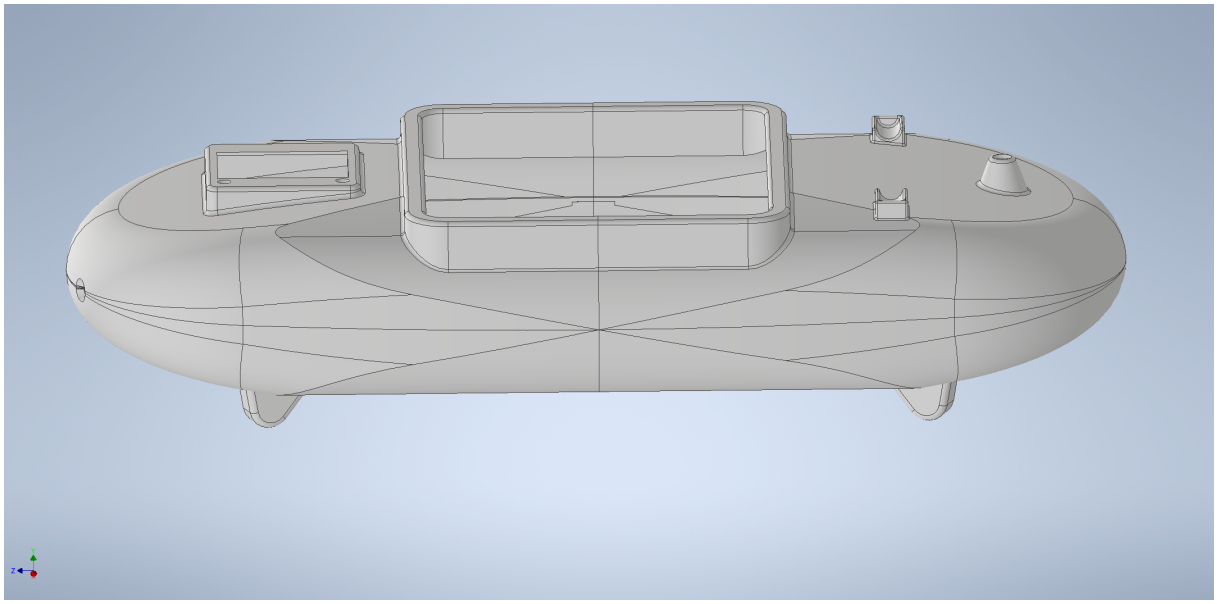


Abbildung 6.2: Theremin Gehäuse.

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 Komparatoren waren auf den Rechtecken bei der steigenden und fallenden Flanke Überschwinger ersichtlich. In einem Paper von Analog Device haben wir erfahren, dass 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 Ausgangsleitung mit einem 300 Ohm Widerstand ab.

Beim ersten Austesten der Oszillatoren mit angeschlossenen Antennen wiesen die Oszillatoren ähnliche Frequenzen auf. Bei einer Veränderung der Lautstärkenoszillator 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 das Problem verschwand.

Durch den häufigen Gebrauch des PCB ist uns bewusst geworden, dass der verwendete JFet sehr empfindlich gegenüber elektrostatischer Entladung ist.

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

Tabelle 7.2: Verwendete Messmittel

Messgerät	Bezeichnung
Lecroy wavesurfer 3054	MSZ-M-0091

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, nutzten wir das Tool *SignalTap Logic Analyzer* in Quartus um auf das Register der Frequenzmessung mit den Messresultaten zuzugreifen. Zu Beginn der Messung war eine Bestimmung des Offset des Referenzoszillator nötig. Dazu wurde der Frequenzgenerator um 120 Hz tiefer eingestellt als der Referenzoszillator. Da 120 Hz die Frequenz ist auf die Kalibriert 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 "getroffen" empfunden wird. 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 Matlab aus Abbildung 7.2 überein. Dabei wurde in Matlab dieselbe Messmethode Simuliert in 1 Hz schritten von 100 Hz bis 2 kHz 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	

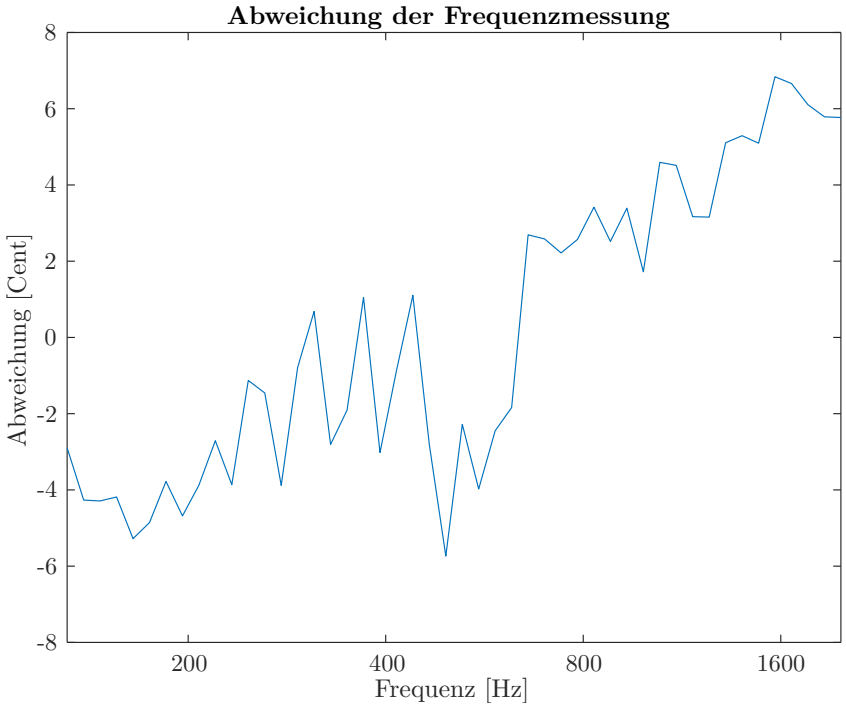


Abbildung 7.1: Maximale Abweichung der Frequenzmessung.

7.3 Glissando Effekt

Auch für den Glissando-Effekt war es nötig einen Test durchzuführen um die Genauigkeit der Töne festzustellen. Wir verwendeten auch hier anstatt unseres PCB einen Funktionsgenerator. Um die Frequenz einzustellen, wurde das Tool *SignalTap Logic Analyzer* in Quartus eingesetzt um auf das Register der Frequenzmessung mit den Messresultaten zuzugreifen. 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 Funktion genauer als 8 Cent. Interessanterweise nimmt bei beiden Einstellungen offenbar die angenäherte Frequenz mit höheren Tönen immer mehr ab. Da die Frequenzmessung wie zuvor besprochen mit grösseren Frequenzen immer mehr Abweichung hat ist dies auch erklärbar.
Weshalb die Kurve nicht wie in der Frequenzmessung immer mehr positiv wird erklärt die Abbildung 7.4. Da bei tiefen Frequenzen die gemessene Frequenz eher tiefer ist als die wirkliche bedeutet dies, dass der ist-Wert grösser ist als der Messungs-Wert. Der Glissando-Effekt nähert nun die Differenz zwischen dem gemessenen Wert und dem soll-Wert an. Dies bedeutet dass der angenäherte Wert jedoch immer noch die Abweichung zwischen ist und Messung hat. Bei den hohen Frequenzen ist es genau umgekehrt. Die Messung ist nun höher im Vergleich zum ist-Wert. Somit bewirkt der Glissando-Effekt in diesem Szenario, dass die angenäherte Frequenz kleiner ist als der soll-Wert.

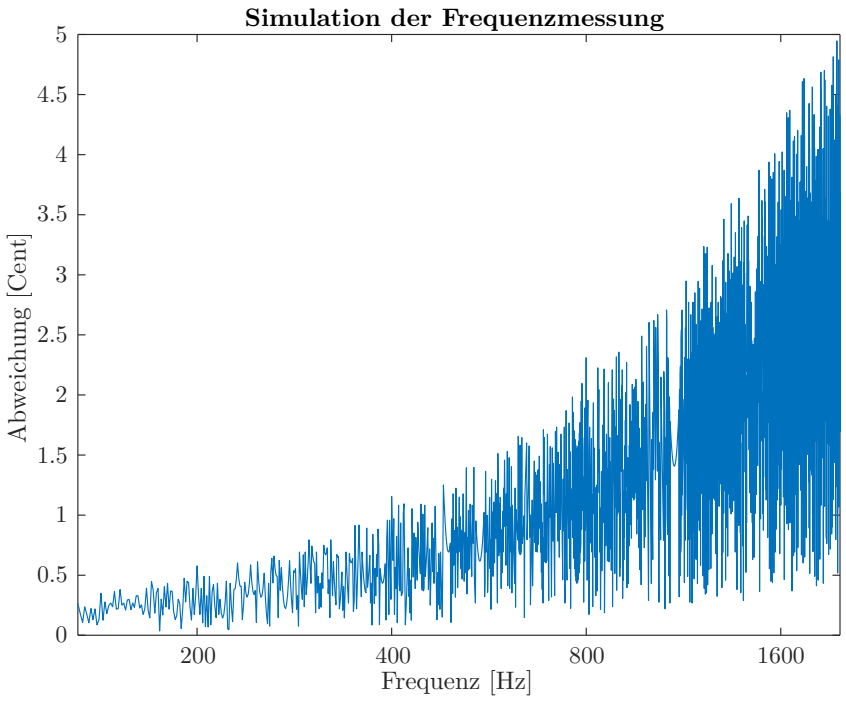


Abbildung 7.2: Abweichung der Frequenzmessung in der Simulation

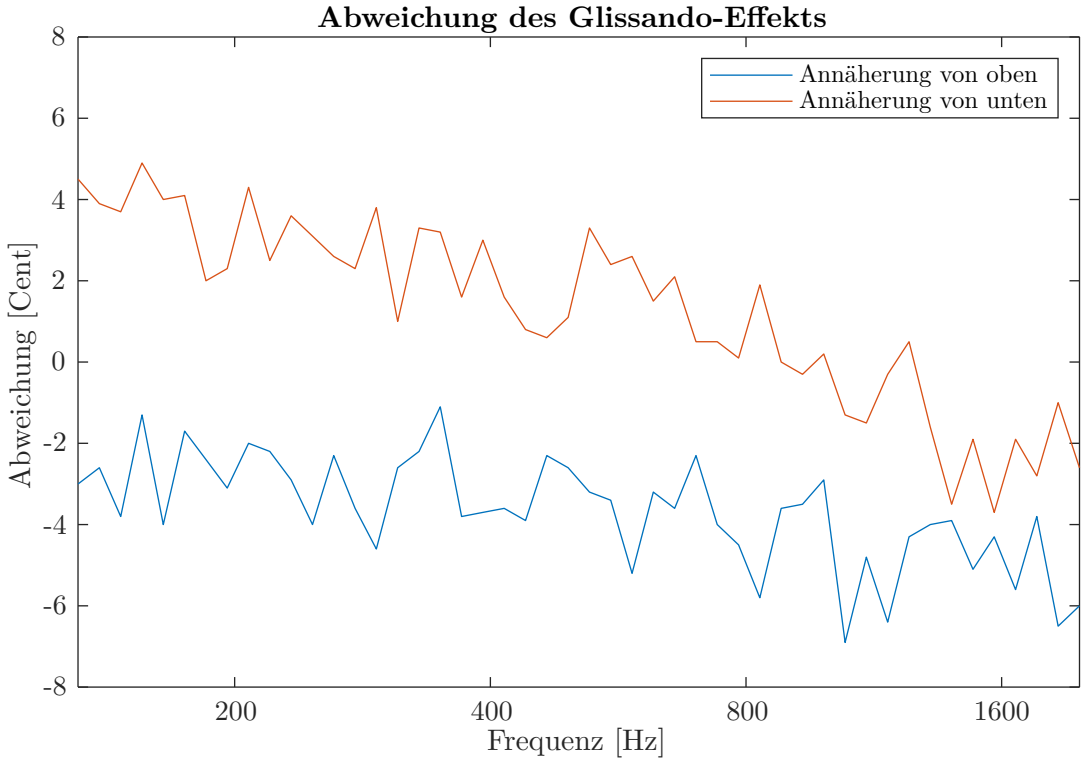


Abbildung 7.3: Resultate der Validierung des Glissando-Effekts

Tabelle 7.4: verwendete Messmittel

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

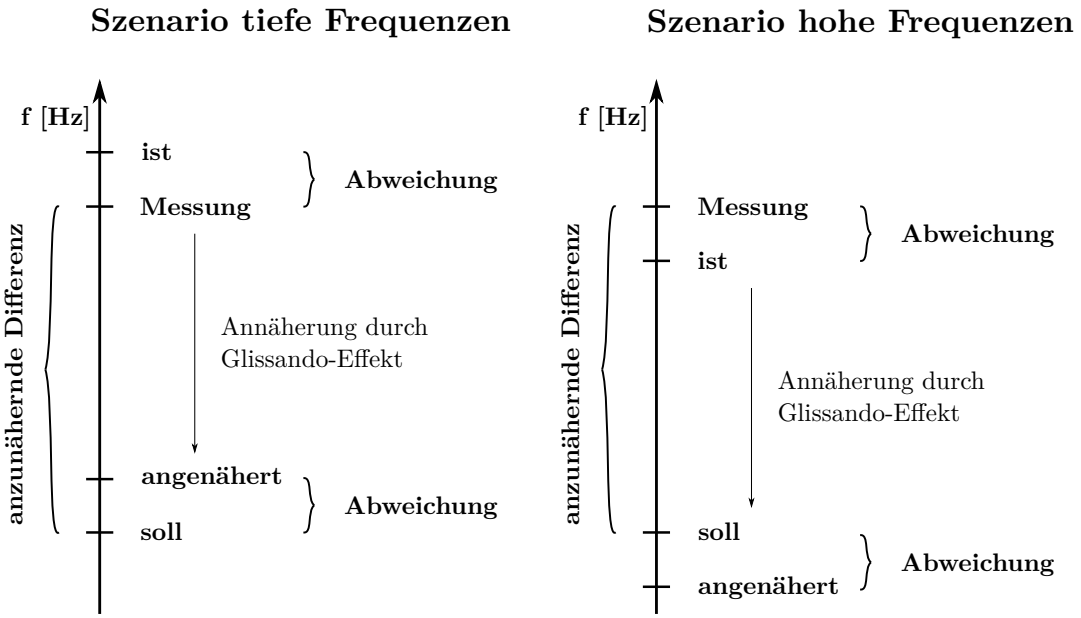


Abbildung 7.4: Erklärung der beobachteten Änderungen

7.4 Gesamttest

Zuletzt haben wir den gesamten Aufbau auf dessen Funktionsfähigkeit getestet. Dabei sind drei Punkte aufgefallen.
Erstens tritt das leichte Aliasing, welches zu erwarten war, 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 auf dem 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 in diesem Fall ein nicht voll ausgesteuertes digitales Signal erhält, fällt dessen Rauschen mehr ins Gewicht.
Weiter ist erst nach der nicht flüchtigen Programmierung des Theremin aufgefallen, dass wenn das USB Kabel ausgesteckt wird das Theremin nicht mehr funktioniert. Dies ist darauf zurückzuführen, dass das verwendete Netzteil, keinen Erdungsanschluss hat. Während den Tests war das Entwicklungsboard ständig über das USB-Kabel und den angeschlossenen PC geerdet. Nachdem das Theremin separat über einen Stecker eine Erdung erhielt, war dieses Problem verschwunden.

8 Schlusswort

Das Projekt 6 umfasste die abschliessende Entwicklung eines digitalen Theremin auf einem FPGA. Die Grundlagen für die Weiterentwicklung stammen aus der Arbeit des Projekt 5 ein Semester zuvor. Diese Grundlagen beinhalteten die Verarbeitung des Antennensignals.

Ergebnisse

Die Implementierung der Lautstärkesteuerung über eine zweite Antenne ist abgeschlossen und funktioniert einwandfrei. Damit diese auch ein Signal erhält ist der zweite analoge Oszillator in einem Redesign des PCB hinzugefügt worden. Dieses Redesign umfasst nebst des zweiten Oszillators eine Speisung der beiden Schaltungen über das gleiche Netzteil wie das FPGA.

Die Bedienung des FPGA läuft über den implementierten Nios II Prozessor und den von ihm gesteuerten LCD Display. Der Benutzer kann über den Touchscreen diverse Einstellungen vornehmen und Funktionen des Theremin einfach aktivieren und deaktivieren.

Der Spieler kann die automatische Kalibration des Theremin über das Menu starten um es auf seinen Spielstil abzustimmen. Dazu gleicht das Theremin die beiden digitalen Referenzoszillatoren auf die analogen Antennenoszillatoren ab.

Das Spielen von genauen Tönen ist nun auch möglich, wenn der Spieler keine ruhige Hand hat. Der eingebaute Glissando-Effekt korrigiert während dem Spielen die Töne auf den nächstgelegenen Ton. Gespielt werden kann in zwei Tonsystemen: normale Tonleiter mit Halbtönen oder eine pentatonische Tonleiter. Die pentatonische Tonleiter ist praktisch um ohne grossen Aufwand ansprechend klingende Melodien zu spielen.

Während dem Spielen ermöglicht es die Anzeige der Spielgenauigkeit zu sehen wie falsch der Spieler seine Hand hält, wenn die pentatonische Tonleiter aktiv ist. Wenn die normale Tonleiter aktiviert ist, wird der nächstgelegene Ton angezeigt. Dies hilft denjenigen Spielern mit weniger musikalischen Talenten.

Schlussendlich wurde das Gerät in ein ansprechendes Gehäuse verpackt. Damit wir ein etwas ausgefalleneres Gehäuse fertigen konnten, entschieden wir uns dieses mit einem 3D-Drucker herzustellen.

Schwierigkeiten

Weil wir viel Zeit für die Inbetriebnahme der ganzen Signalverarbeitung benötigten und wegen Umwelteinflüssen ausserhalb unserer Kontrolle, konnten wir die zusätzlichen Klangeffekte nicht implementieren.
Weiter ist beim Spielen bei höheren Frequenzen ein leichtes Aliasing zu hören, was auf den Aufbau der Filter zurückzuführen ist. Die Filter sind so eingestellt, dass dieses Aliasing möglichst minimiert ist.
Zuletzt kann bei der höchsten Lautstärkeeinstellung auf dem Display und der tiefsten Einstellung über die entsprechende Antenne ein Rauschen gehört werden. Dies ist auf die Implementierung der Lautstärkeeinstellung zurückzuführen. Bei dieser mussten wir Kompromisse eingehen, wegen Schwierigkeiten mit dem verbauten Codec.

Weiterentwicklung

Das Aliasingproblem könnte durch die Verwendung eines Analog-Digital-Wandlers verkleinert werden. Dies da ein Analog-Digital-Wandler anstelle des Rechtecksignals direkt den Sinus des analogen Oszillators einlesen könnte. Die Multiplikation der Sinusse des Referenzoszillators und des analogen Oszillators würde nur eine hochfrequente Komponente bedeuten ohne Oberwellen. Nötig wäre dazu jedoch ein Wandler mit sehr hoher Abtastfrequenz.

Die Verwendung eines neuen Codec, mit welchem Nullstellenerkennung möglich wäre ist ein Entfernen des Rauschens erreichbar, welches bei bestimmten Gegebenheiten hörbar ist.

Für die zwei zuvor genannten Verbesserungen ist ein Redesign des PCB nötig, bei dem eine Überarbeitung der Oszillatoren nötig ist. Eine Schutzbeschaltung der JFet ist nötig um diese vor statischer Entladung zu schützen.

Lerngewinn

Bei der Projektwahl haben wir das digitale Theremin gewählt um das erlernte Wissen über VHDL auszubauen und zu festigen. Über die letzten zwei Semester haben wir einige Erfahrungen im Umgang mit VHDL Design und der Implementierung des NIOS II Softcores gesammelt.

Im Umgang mit Audio und Signalverarbeitung haben wir gelernt darauf zu achten, das die Clocks dieser Komponenten vom gleichen Referenz Clock abgeleitet sein müssen. Dies hat zur Folge das die Komponenten synchron laufen. Somit kann es nicht vorkommen das eine Komponente das Signal schneller verarbeitet als die andere oder dass Daten verloren gehen.

Im Umgang mit CIC Filtern haben wir gesehen, dass diese sehr praktisch sind. Jedoch müssen die Nullstellen in der Übertragungsfunktion berücksichtigt werden, da Signale in diesen Bereichen Aliasing verursachen. Der Einfluss der Nullstellen kann durch den Einsatz von mehreren CIC Filtern mit kleinem Dezimationsfaktor verringert werden.

9 Danksagung

In erster Linie möchten wir uns bei unserem Projektbetreuer Herrn Schenk und unserem Auftraggeber Herrn Hanspeter Schmid für Ihre Unterstützung während der vergangenen zwei Projekten bedanken. Mit ihrem Fachwissen und Ihren Ideen haben sie uns geholfen verschiedenste Problemstellungen zu meistern.

Ebenfalls möchten wir uns bei den Mitarbeitern des ISE bedanken. Insbesondere Herrn Pichler und Herrn Zardet die uns in FPGA Design Fragen unterstützten.

Besonders möchten wir uns auch bei Herrn Stefan Muhr bedanken. Material und PCB Bestellungen wurden von ihm jeweils inert rekordverdächtig kurzer Zeit bearbeitet. Auch bei anderen Anliegen bezüglich Messgeräten und Bauteilen hat er uns stets aufgeholfen und mit seiner positiven Einstellung zu einem guten Arbeitsklima beigetragen.

Zudem möchten wir uns bei Herrn Patrick Specht bedanken, für die nützlichen Tipps die er uns gegeben hat bezüglich 3D Druck.

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] Rick Lyons. (2020). A Beginner's Guide To CIC-Filters. (Abrufdatum 19.08.2020), Adresse: https://www.dsprelated.com/showarticle/1337.php.

[11] Intel. (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.

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

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

[14] 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.

[15] ——, (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.

[16] ——, (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.

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

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

[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`.