# Dokumentation – Audio Synthesizer

| | |
|---|---|
| Fach / Institution: | DTP2 / ZHAW |
| Gruppenmitglieder: | Beat Sturzenegger<br>Markus Bodenmann<br>Tharmelan Theivanesan |
| Dozenten: | Marina de Queiroz Tavare<br>Johanna Decurtins |
| Source-Repository: | https://github.com/streifehoernli/midi-synthesizer-fpga |
| Erstellungsdatum: | 11.06.2020 |

# Inhaltsverzeichnis


**1 Einleitung ..... 1**

**2 Hauptteil ..... 1**

*2.1 Übersicht ..... 1*

*2.2 Übersicht Altera DE2-115 Belegung ..... 1*

*2.3 Meilenstein 1 – Analog Audio-Loop ..... 2*

2.3.1 Infrastructure ..... 2

2.3.2 Codec Control ..... 2

2.3.3 $I^2C$ Master ..... 3

*2.4 Meilenstein 2 – Digital Audio-Loop ..... 4*

2.4.1 Digital Audio ..... 4

*2.5 Meilenstein 3 – Direct Digital Synthesis ..... 5*

2.5.1 Tone Generator (DDS) ..... 6

*2.6 Meilenstein 4 – MIDI Interface ..... 7*

*2.7 Meilenstein 5 – Zusatzfunktion ..... 8*

2.7.1 LCD-Ausgabe ..... 8

*2.8 Visualisierung ..... 9*

*2.9 Verschiedene Instrumente in Tone Generator ..... 10*

*2.10 Phase Modulation ..... 11*

**3 Schluss ..... 14**

**4 Quellenverzeichnis ..... 15**

**5 Anhang ..... 15**

*5.1 Projektevaluation ..... 15*

*5.2 Verwendete Tools ..... 15*

*5.3 Pinbelegung FPGA ..... 15*

# 1 Einleitung

In dieser Dokumentation wird eine Hardwarebeschreibung erklärt. Der Projektauftrag ist es mit dem Altera Entwicklungsboard DE2-115 ein Synthesizer zu programmieren in der Hardwareprogrammiersprache VHDL. Der Synthesizer soll Audiodaten empfangen, gegebenen falls verändern und ausgeben.
Ein externes Keyboard wird über die MIDI Schnittstelle angeschlossen und empfängt Daten, anhand dieser wird ein Ton generiert. Die Erarbeitung des Projekts ist in fünf Meilensteine unterteilt. In den Meilensteinen 1-4 wird der Synthesizer aufgesetzt. Im Meilenstein 5 sind unsere Zusatzfeatures beschrieben.
Wir haben 4 Zusatzfunktionen im Projekt implementiert (siehe Abschnitt 2.5). Die erste Zusatzfunktion ist die LCD-Ausgabe, hier werden die eingestellten Einstellungen angezeigt. Die zweite Funktion ist die Simulation von verschiedenen Instrumenten Des Weiteren wurde eine LED-Ausgabe programmiert, welches die Lautstärke darstellt. Die letzte Funktion ist die Phasen Modulation, in welcher mittels Benutzereingabe der Ton verändert werden kann.

# 2 Hauptteil

## 2.1 Übersicht

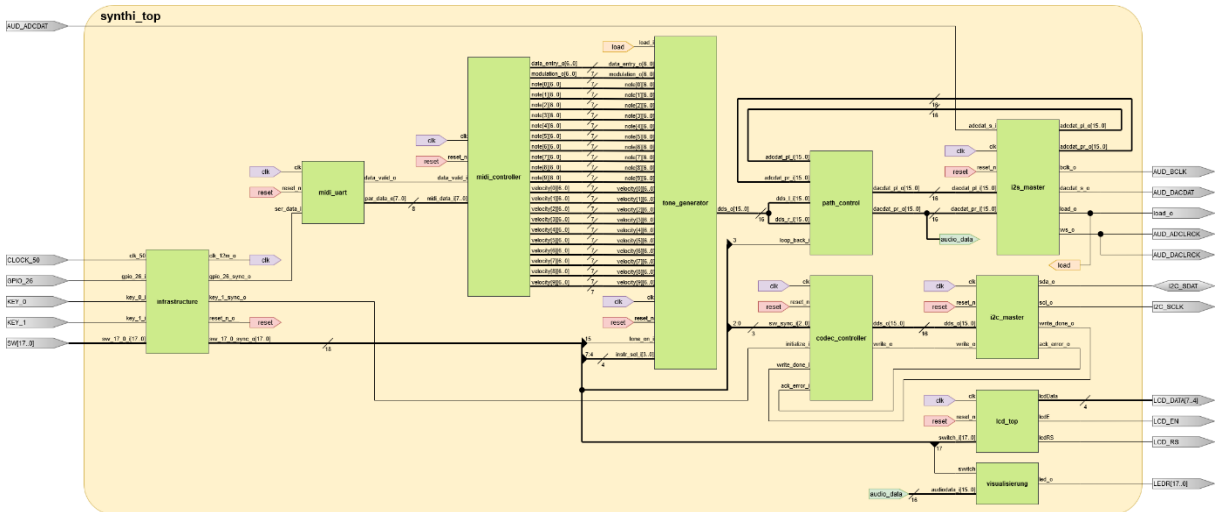



*Abbildung 1: Gesamtübersicht*

In Abbildung 1 sieht man den ganzen Blockaufbau des Projektes. Wie in der Einleitung erwähnt ist das Projekt in fünf Abschnitte unterteilt, welche in den folgenden Abschnitten genauer eingegangen wird.

## 2.2 Übersicht Altera DE2-115 Belegung

| SW_17 | SW_16 | SW_15 | SW_14 | SW_13 | SW_12 | SW_11 | SW_10 | SW_9 |
|---|---|---|---|---|---|---|---|---|
| LED sensitivity | unused | mute_n | unused | unused | unused | unused | unused | unused |

| SW_8 | SW_7 | SW_6 | SW_5 | SW_4 | SW_3 | SW_2 | SW_1 | SW_0 | GPIO26 |
|---|---|---|---|---|---|---|---|---|---|
| unused | Instruments | | | | Audio CODEC mode | | | | MIDI Channel |

| Key_3 | Key_2 | Key_1 | Key_0 |
|---|---|---|---|
| unused | unused | load Audio CODEC | reset |

## 2.3 Meilenstein 1 – Analog Audio-Loop

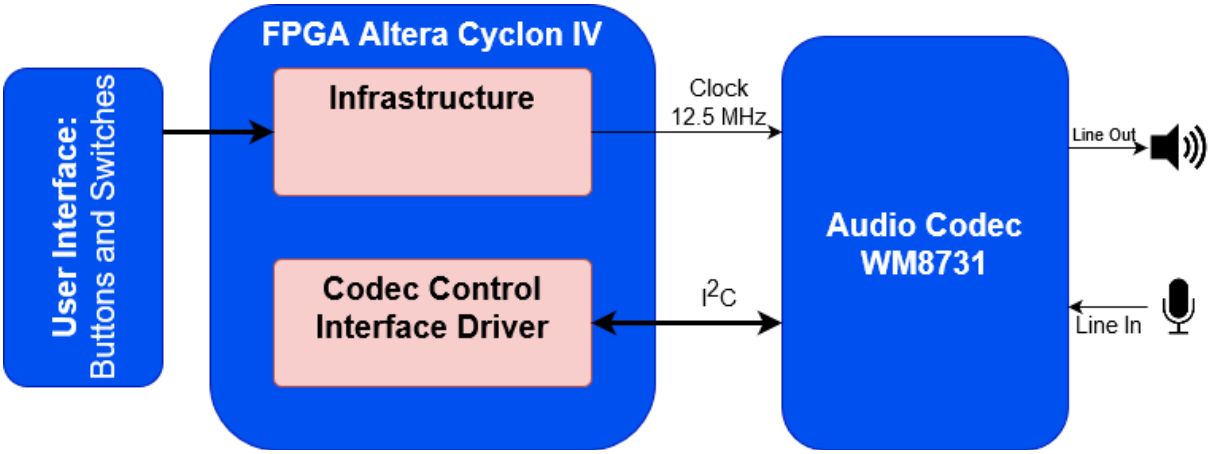



*Abbildung 2: Grober Aufbau des 1. Meilensteins*

### 2.3.1 Infrastructure

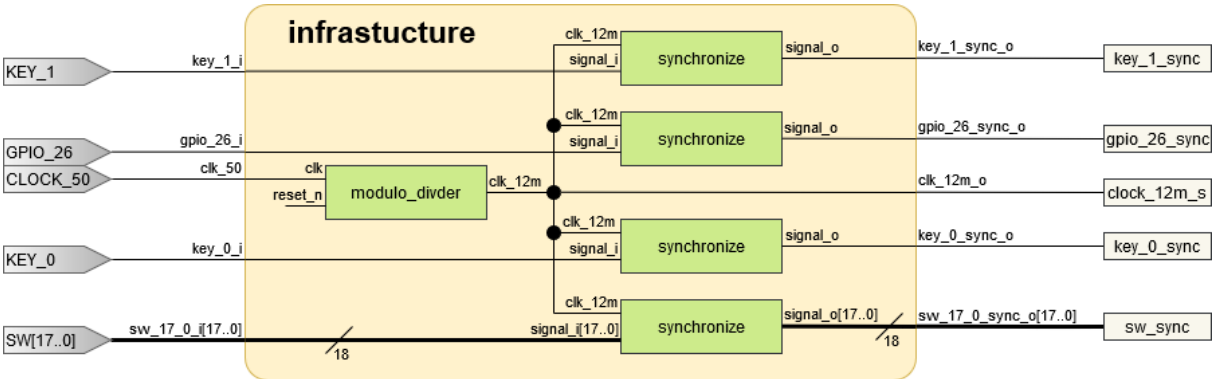



*Abbildung 3: Blockübersicht vom Infrastruktur-Block*

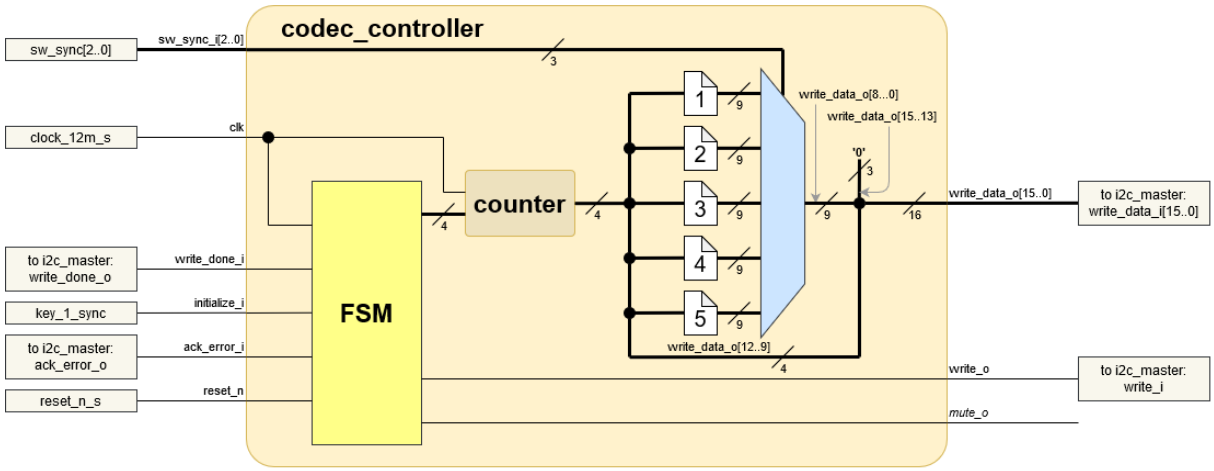
Der Infrastructure synchronisiert mittels zwei Filp Flop's die Eingangssignale KEY_0, KEY_1, SW(17..0) und GPIO_26 mit der 12.5 MHz Clock. Der Modulo Divider im Infrastructure wandelt den Takt 50 MHz auf 12.5 MHz herunter.

### 2.3.2 Codec Control

Der Codec Control verarbeitet das empfange Signal (AUD_XCK) je nachdem welche Schalter (SW2..0) eingeschalten sind, gemäss der Tabelle unten. Das verarbeitete Signal wird weitergeleitet.

| Modus | SW(2) | SW(1) | SW(0) | Einstellung |
|---|---|---|---|---|
| 1 | 0 | 0 | 1 | Bypass |
| 2 | 1 | 0 | 1 | Links stummschalten |
| 3 | 0 | 1 | 1 | Rechts stummschalten |
| 4 | 1 | 1 | 1 | Beide Kanäle stummschalten |
| 5 | x | x | 0 | DDS |

Die vom Audio Codec empfangene Signal werden im Modus 1 direkt wieder an den Ausgang des Audio Codec ausgegeben. Modus 2, 3 und 4 funktionieren gleich, jedoch wird der linke Kanal (Modus 2), der rechte Kanal (Modus 3) oder beide Kanäle (Modus 4) stummgeschalten.
Im Modus 5 werden die vom DDS (Direct Digital Synthesis) im noch nicht vorhandenen Tone Generator-Block, generierten Daten an den Audio Codec gesendet.




1: C_W8731_ANALOG_MUTE_LEFT
2: C_W8731_ANALOG_MUTE_RIGHT
3: C_W8731_ANALOG_MUTE_BOTH
4: C_W8731_ANALOG_BYPASS
5: C_W8731_ADC_DAC_0DB_48K

*Abbildung 4: Blockübersicht vom Codec Controller*

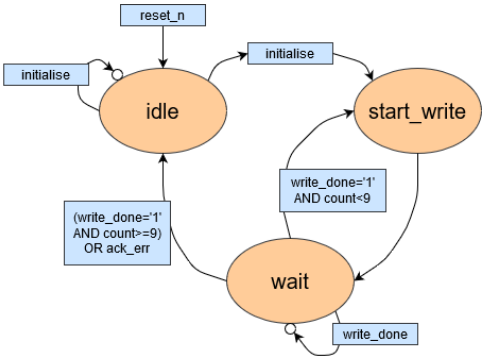



*Abbildung 5: Zustandsdiagramm (FSM) inerhalb des Codec Controller-Block*

### 2.3.3 I²C Master

Der Codec Controller benötigt noch einen Block, welcher die I²C-Kommunikation verarbeitet. Dieser I²C Master Block wird für das Projekt zu Verfügung gestellt.

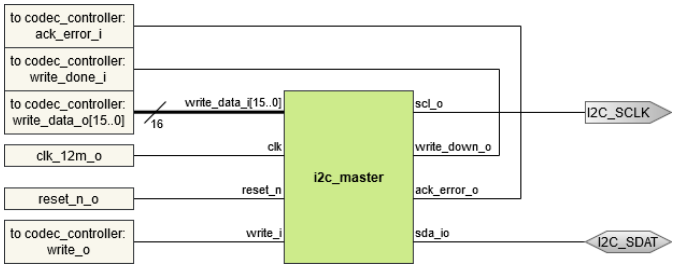



*Abbildung 6: Übersicht I²C Master Block*

## 2.4 Meilenstein 2 – Digital Audio-Loop

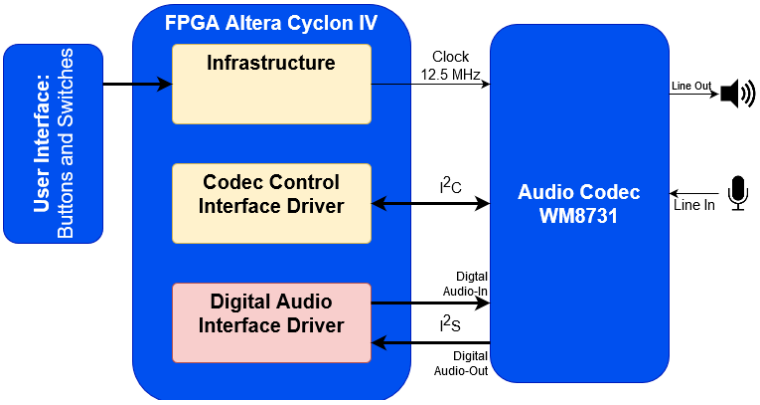



*Abbildung 7: Grober Aufbau des 2. Meilensteins*

### 2.4.1 Digital Audio

Der I2S Master wandelt parallelen Audio Daten in einen seriellen Datenstrom um. In der Abbildung 9 ist das Zeitdiagramm der Datenübertragen ersichtlich. In der Abbildung ist auch ersichtlich, dass zuerst die Audio Daten des linken Kanals und dann die des rechten Kanals übertragen werden. Der I2S Master wandelt die seriell erhaltenen Audio Daten von Audio Codec in Parallele Daten um. Die umgewandelten Daten werden dem Path Control-Block übergeben. Dazwischen könnten die Daten noch verarbeitet und verändert werden, doch dies wurde nicht in diesem Projekt realisiert.

Der I2S Master-Block enthält einen “Modulo Divider”, welches das 12,5 MHz Clock Signal in ein 48 kHz Signal umwandelt. Diese Umwandlung ist nötig, da das Signal “WS” (Word Select) alle 10,4 ms wechselt. Dieser Wechsel ist zuständig für das Senden der linken und rechten Kanaldaten. Aus diesem Grund besitzen die Blöcke im I$^2$S Master-Block, ausser der Counter-Block, dieses 48 kHz Signal als Clock Signal.

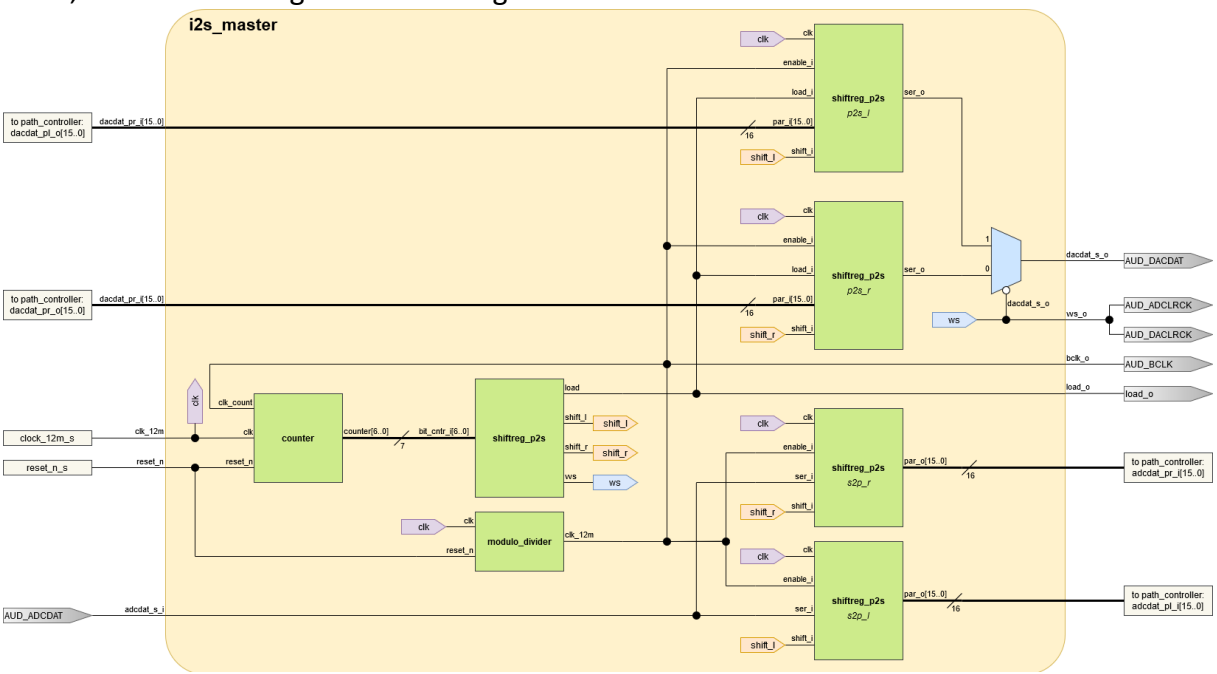



*Abbildung 8: Blockübersicht vom Codec Controller*

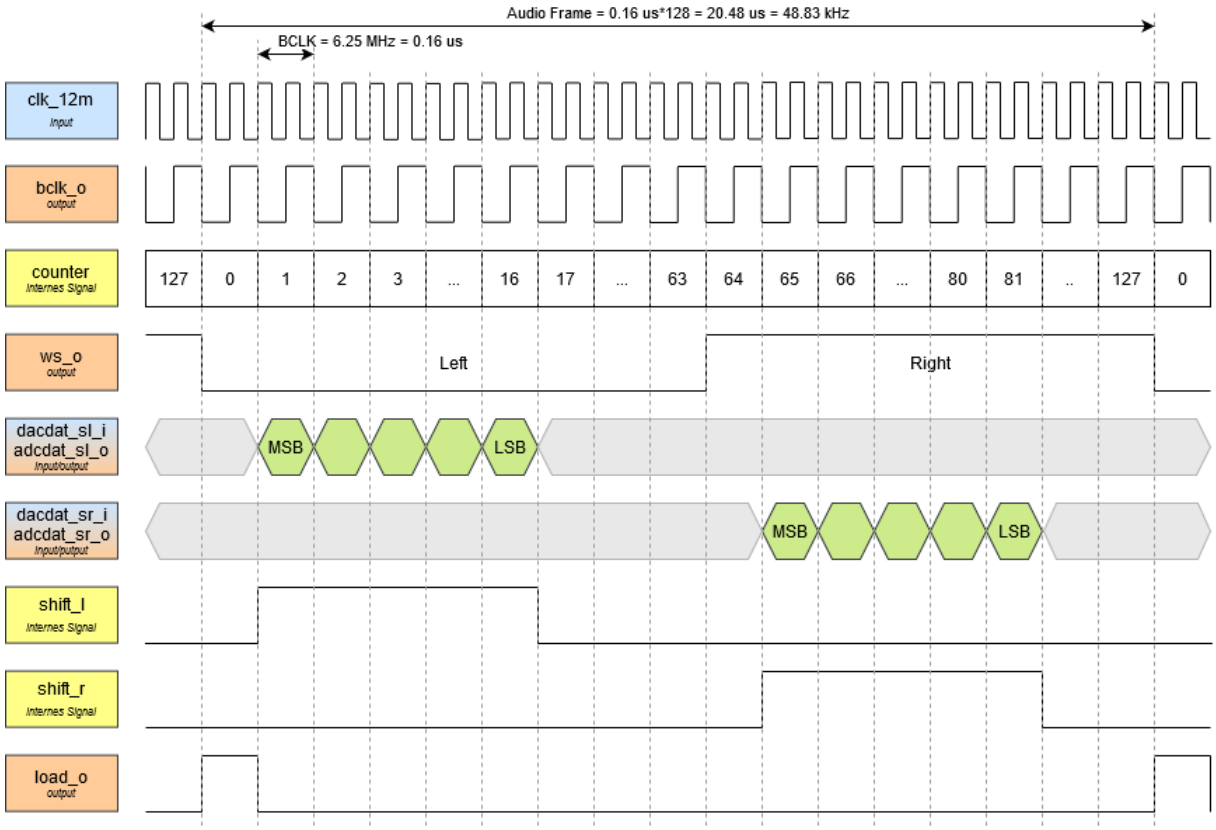



*Abbildung 9: Zeitdiagramm des I²S Master-Blockes*

## 2.5 Meilenstein 3 – Direct Digital Synthesis

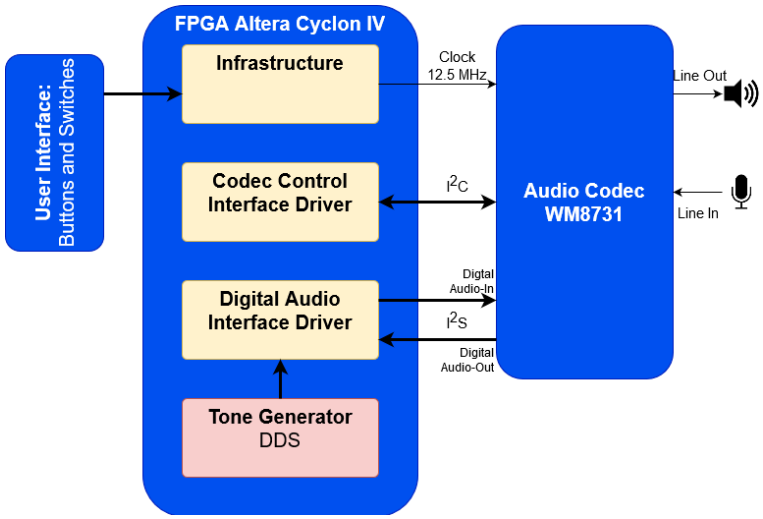



*Abbildung 10: Grober Aufbau des 3. Meilensteins*

### 2.5.1 Tone Generator (DDS)

Die Direct Digital Synthesis (DDS) ist ein Verfahren in der digitalen Signalverarbeitung zur Erzeugung periodischer Signale mit praktisch beliebig feiner Frequenzauflösung. In einer Lookup Table (LUT) werden die Amplitudenwerte zu einem bestimmten Zeitpunkt gespeichert.

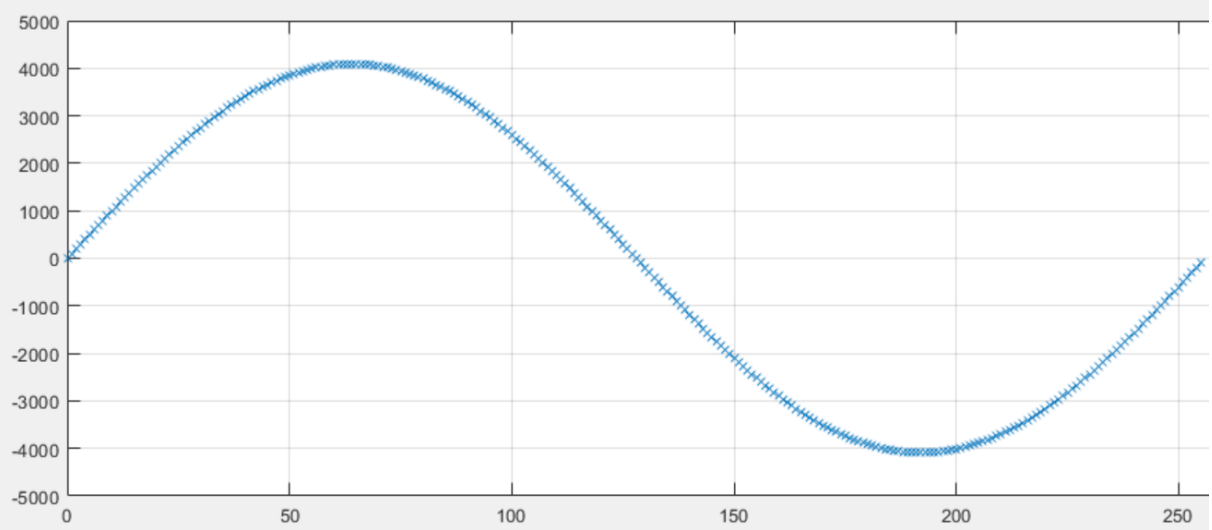



*Abbildung 11: Beispiel einer LUT für einen Sinus mit einer Auflösung von 13 Bit und einer Breite von 256*

Mithilfe eines Counters wird über die LUT iteriert. Der Counter wird zyklisch erhöht. Je grösser das Inkrement, desto höher die Frequenz. Auf dem FPGA wurde ein 19 Bit breites Counterregister angelegt. Der Counter wird mit einem Signal von 48khz erhöht. Bei einem minimalen Inkrement von eins ergibt sich somit die kleinste mögliche Frequenz von 0.0916Hz. Mit dieser Auflösung kann jeder Ton gespielt werden. Mit einem Multiplikator kann das Inkrement nun so erhöht werden, dass der gewünschte Ton gespielt wird.
Für das Testen dieses Features wird die gleiche Testbench verwendet, wie für das Zusatzfeature «Phasen Modulation».

## 2.6 Meilenstein 4 – MIDI Interface

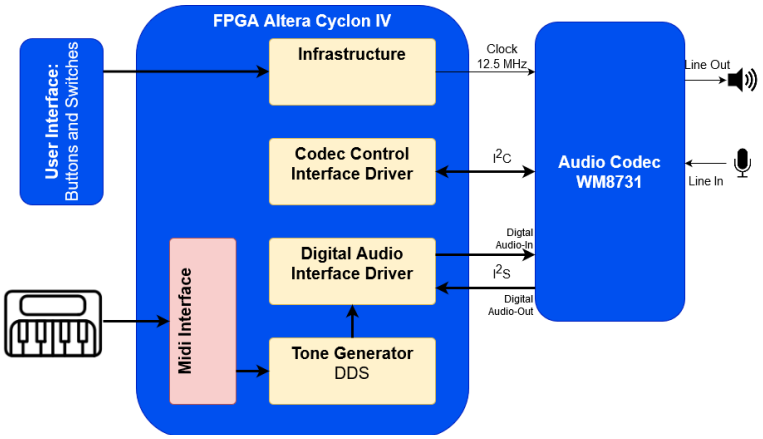



*Abbildung 12: Grober Aufbau des 4. Meilensteins*

MIDI ist die Schnittstelle für Musikinstrumente. MIDI steht für Musical Instrument Digital Interface. Wenn auf dem Keyboard eine Taste gedrückt wird, werden digitale Informationen zu Lautstärke und Ton am MIDI Ausgang des Keyboards ausgegeben. Die Daten werden mit einer Baudrate von 31250 übertragen. Eine Übertragung enthält immer ein Startbit gefolgt von 8 Datenbits und zum Schluss noch ein Stoppbit. Das MIDI Protokoll erlaubt die Interpretation von 128 Tönen.
Damit möglichst viele Informationen zu einem Ton mitgegeben werden kann, besteht eine Meldung aus drei Bytes.

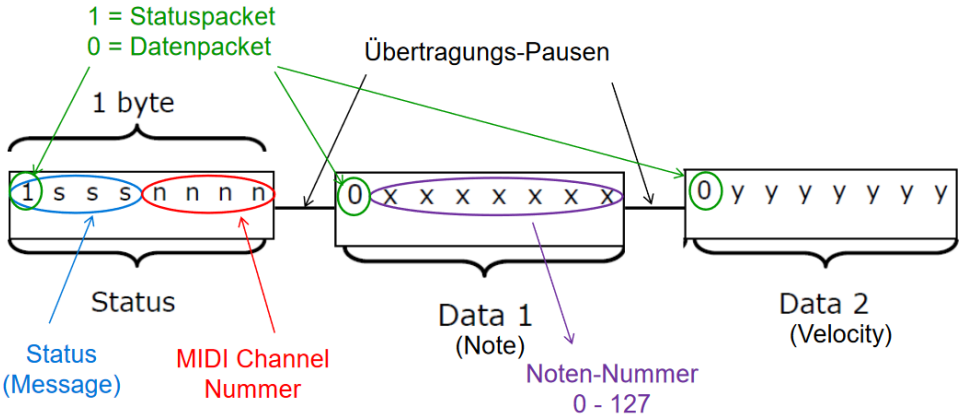



*Abbildung 13: Grafik aus den Vorlesungsfolien, Beispiel einer MIDI Meldung*

Folgende Stati werden unterstützt:

| Status | Status Value | Data1 | Data2 |
|---|---|---|---|
| Set Note | «001» | 7 Bit Note | Lautstärke |
| Delete Note | «000» | 7 Bit Note | - |
| Controller Value | «011» | "0000001" für Modulation Wheel<br>"0000111" für Data Entry | 7 Bit Zahlenwert |

## 2.7 Meilenstein 5 – Zusatzfunktion

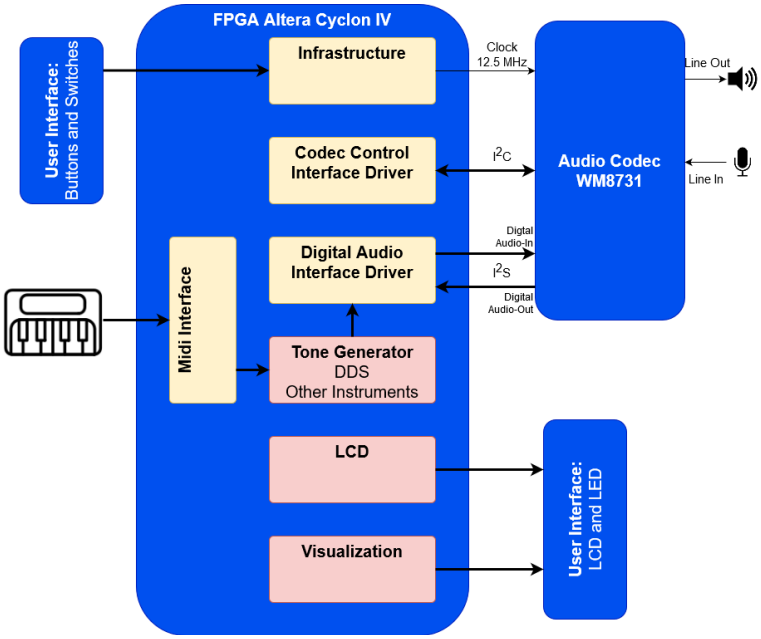



*Abbildung 14: Grober Aufbau des 5. Meilensteins*

### 2.7.1 LCD-Ausgabe

#### Beschreibung

Mit der LCD-Ausgabe wird den Benutzer angezeigt welches Instrument gespielt wird. Zusätzlich werden die verschiedenen Einstellungen aus den Kapitel 2.3.2 angezeigt.
Für die LCD-Aufgabe werden die einstellbaren Tasten abgefragt ob sie eingeschalten sind und dementsprechend ausgegeben.

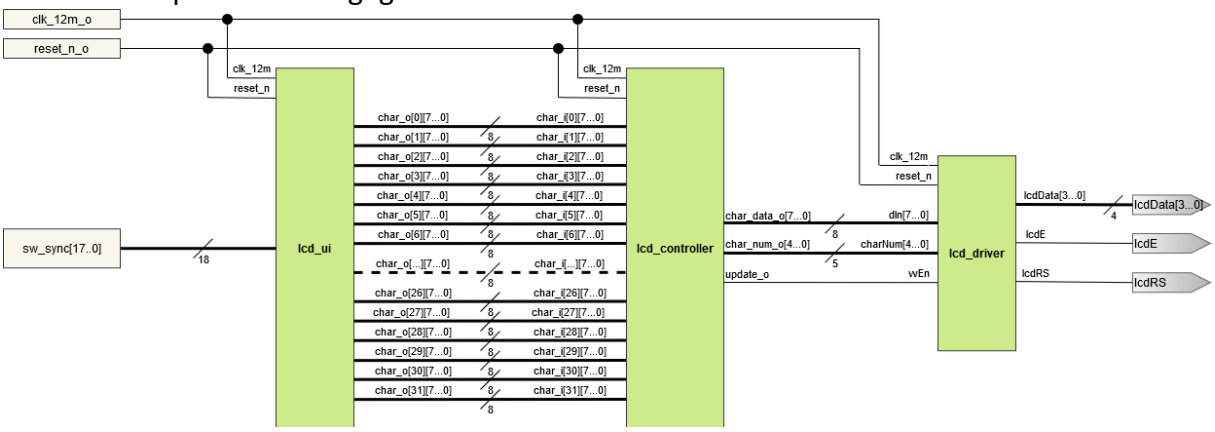



*Abbildung 15: Blockübersicht vom LCD Top*

Der LCD Top-Block besteht aus drei Blöcken. Im LCD UI-Block werden die Schalter ausgewertet. Je nach Schalterstellung werden andere Werte über die Ausgangsschnittstelle an den LCD Controller-Block weitergegeben. Der LCD Controller-Block teilt dem LCD Driver-Block nacheinander mit, was für ein Zeichen sich an welcher Stelle befindet. Der LCD Driver schreibt die Daten an den LCD. Für das Schreiben muss eine gewisse Abfolge der Steuer- und Datensignale erfolgen um den LCD Display mitzuteilen, an welcher Stelle welches Zeichen angezeigt werden soll. Da dieser Block uns zu Verfügung gestellt wurde, wird nicht weiter darauf eingegangen. Da die Überprüfung einfach optisch gemacht werden kann, wurde auch auf ein Testbench verzichtet.

### Optimierung des Features

Die LCD-Ausgabe, welche die Einstellungen anzeigt, wird nur aktualisiert, wenn eine Taste gedrückt wird, das bedeutet wird Einstellung geändert und man drückt die Taste nicht um die Anzeige zu aktualisieren, werden falsche Einstellungen angezeigt. Um das zu verhindert könnte man ein Zeichen (z.B. ein Stern) implementieren welche bei der LCD-Ausgabe auftaucht sobald die Anzeige nicht mehr aktuell ist.

## 2.8 Visualisierung

### Beschreibung

Mit der Visualisierung wird das Audiosignal, dass an den Audio Codec gesendet wird, an die LEDs ausgegeben. Sie sollen also die Lautstärke anzeigen. Der Visualisierungs-Block ist ziemliche simple. Das Audio Signal wird mit verschiedenen Referenzwerten verglichen und je nachdem zu welchem Referenzwert das aktuelle Audio Signal passt, leuchten die LEDs unterschiedlich.
Da die Anzahl der LEDs begrenzt ist, befindet sich nur ein Visualisierungs-Block innerhalb des Projektes, welcher den linken Kanal anzeigt.

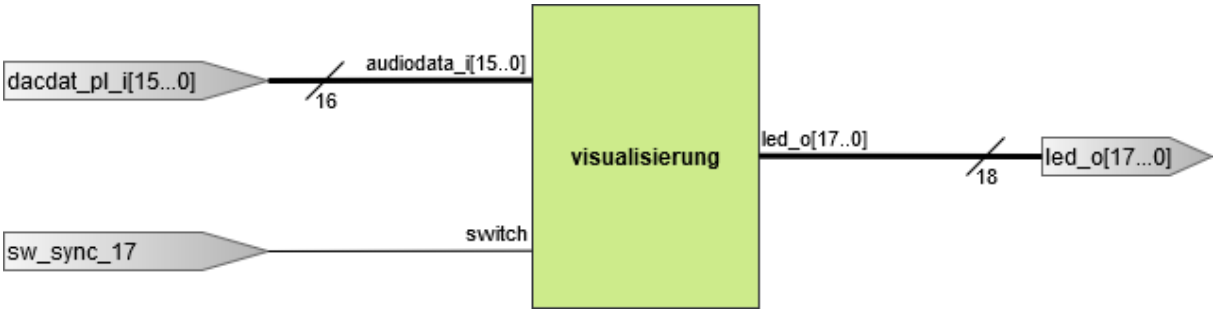



*Abbildung 16: Blockübersicht von der Visualisierung*

### Testing

Für diesen Block wurde ein Testbench geschrieben. Aus dem Grund, dass man die Visualisierung der LEDs schlecht optisch nachvollziehen kann, entschied man sich, mit einer kleinen Testbench die Funktion des Blocks zu testen. Die Testbench gibt nur nacheinander die Grenzwert-Werte in den Eingang ein. Die Überprüfung der Ausgangswerte wird manuell im Modelsim gemacht.

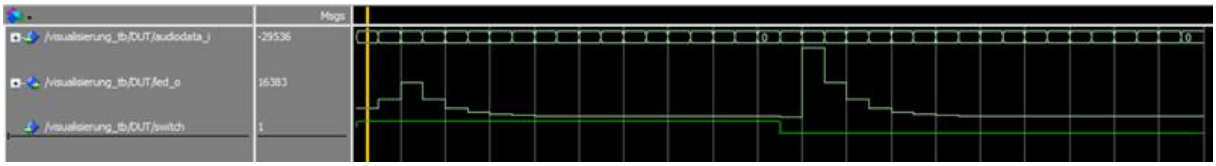


*Abbildung 17: Ausschnoitt aus der Testbench für die Visualisierung*

## 2.9 Verschiedene Instrumente in Tone Generator

### Beschreibung

Ein weiteres Zusatzfeature, welches der Synthesizer unterstützt, ist die Einstellung von verschiedenen Instrumenten. Wie im Milestone 3 beschrieben benutzt der Tonegenerator eine LUT, in welcher er die Amplitude des Audiosignals zu einem bestimmten Zeitpunkt ausliest. Aus diesem Grund benötigt jedes Instrument eine eigene LUT. Diese wurden mithilfe des Matlab-Skripts “lut_gen.m” erstellt. Die Funktionen, welche den Ton eines Instrumentes beschreiben, wurden mit dem Java Programm "FourierCoefficientsToFormula.java" erstellt. Dieses Java Programm benötigt die Fourier-Koeffizienten eines Instruments als Input. Die Koeffizienten wurden aus dem Sandell Harmonic Archive kopiert [1]

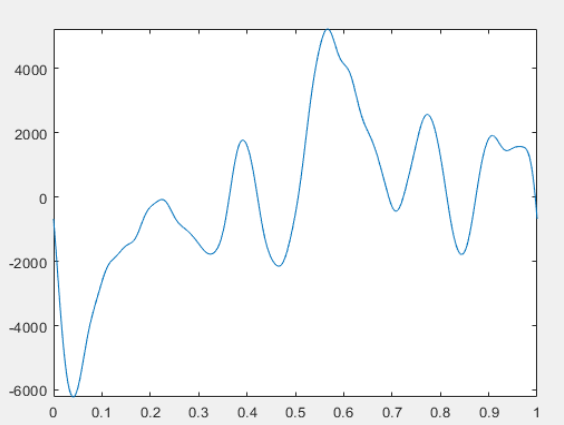



*Abbildung 18: Tonsignal einer Klarinette*

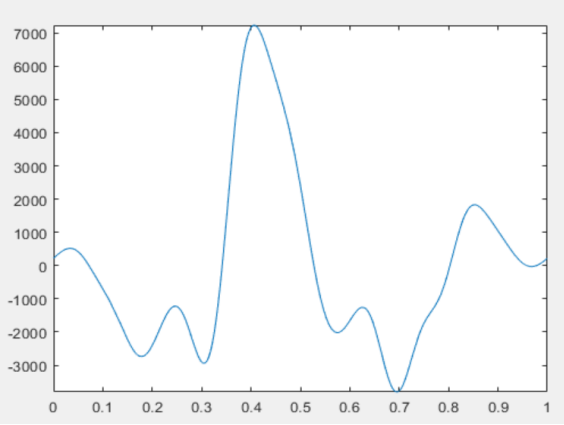



*Abbildung 19: Tonsignal einer Flöte*

Bei der Erstellung der LUT muss beachtet werden, dass die Auflösung nur 13 Bit ist. Da die Amplitude des Ton Signals sowohl positiv wie auch negativ sein kann, können maximal die Werte von [-4096;4095] angenommen werden. Aus diesem Grund wird im Matlab-Skript die Amplitude auf diesen Wert normiert.

### Implementation

Die Implementation in VHDL ist danach ganz einfach. Im DDS wird anhand der Input Switches SW(7..4) entschieden aus welcher LUT der Amplituden Wert gelesen werden soll. Die folgenden Instrumente werden unterstützt.

| Input Switch Value | Instrument Name | Input Switch Value | Instrument Name |
|---|---|---|---|
| 0000 | Sinus | 1000 | Oboe |
| 0001 | Bratsche | 1001 | Flöte |
| 0010 | Fagott | 1010 | Geige |
| 0011 | Klarinette | 1011 | Tuba |
| 0100 | Englisches Horn | 1100 | Piccoloflöte |
| 0101 | Trompete | 1101 | Nicht belegt – Sinus |
| 0110 | Klavier | 1110 | Nicht belegt – Sinus |
| 0111 | Orgel | 1111 | Nicht belegt – Sinus |

### Testen

Für dieses Feature wurde keine Testbench geschrieben. Es muss manuell getestet werden. Damit das Feature getestet werden kann müssen die Switches wie folgt eingestellt werden: SW(3..0): 0, SW_15: 1, SW(7..4): je nach Instrument (siehe Tabelle)
Nach dem Hochladen auf den FPGA muss der KEY 0 (reset) und danach der KEY 1 (laden Audio-CODEC). Zudem muss der Midi Controller über den GPIO Pin 26 verbunden werden. Das Audiosignal wird dann auf dem Kopfhörerausgang des Audio-CODEC ausgegeben.

### Optimierung des Features

Der Synthesizer unterstützt zehn gleichzeitig gespielte Töne, deshalb werden auch zehn DDS benötigt. Jeder dieser DDS muss sich alle LUT für die Instrumente speichern. Einen Ansatz, um Ressourcen zu sparen wäre, dass man die LUT ins Memory schreibt und dieses in einem separaten Block ausliest. Es muss somit nur ein einzelner Wert in Flip-Flops gehalten werden. Von dem Timing ist es auch kein Problem, da die interne Clock mit 12.5Mhz läuft kann man problemlos auf den Speicher zugreifen ohne, dass man am Ton etwas merkt. Denn der Audio-CODEC wird mit einer Frequenz von 48kHz beliefert.

## 2.10 Phase Modulation

### Beschreibung

Durch Nutzung von Frequenzmodulation (FM) oder Phasenmodulation (PM) wird eine andere Klangfarbe erzeugt. FM und PM haben in der Musik die gleiche Auswirkung. Meistens wird von FM gesprochen, aber in Wirklichkeit ist PM gemeint. Dieser Synthesizer nutzt Phasenmodulation. Die Idee hinter der Phasenmodulation ist sehr simpel, aber das Resultat nur schwer vorstellbar. Grundsätzlich geht es darum, dass man mit einem Carrier und einem Modulator arbeitet. Der Carrier ist das Grundsignal. Der Modulator ist ein anderes Signal, dessen Amplitude auf die Phase addiert wird. Der Carrier ist das Signal des Grundtons. $s_{PM}(t) = A \cdot \cos(\omega_c t + \beta \cdot \cos(\omega_m t))$ [2]. Ein wichtiges Merkmal der Phasenmodulation ist das C:M Ratio, es ist das Verhältnis zwischen der Frequenz des Carriers und der Frequenz des Modulators. Dieses Ratio wird meist in ganzen Zahlen angegeben. Eine andere wichtige Rolle spielt das $\beta$. Es ist der Vorfaktor, welcher entscheidet wie viel Einfluss der Modulator auf die Phase des Carrier hat.
Um zu verstehen was genau passiert wurde das Matlab-Skript «pm_modulation.m» erstellt.

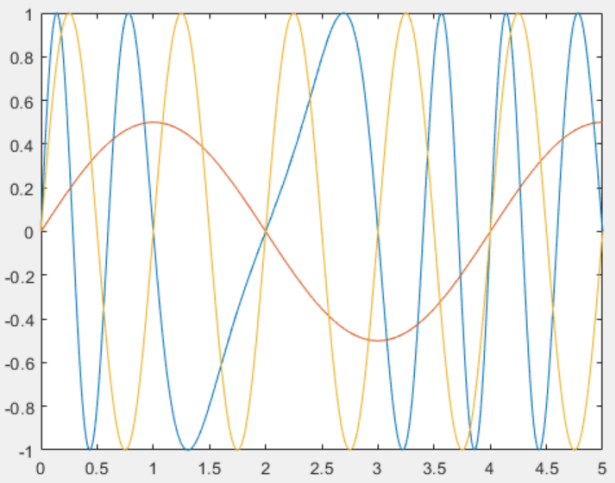



*Abbildung 20: Beispiel für ein mit Phasenmodulation modulierter Sinus*

Aus dieser Abbildung kann man erkennen was für einen Einfluss der Modulator auf das Carrier Signal hat. Das modulierte Signal ist in blau Abgebildet. Das in dieser Abbildung verwendete C:M Ratio beträgt 4:1. $\beta = 0.5$

Dort wo die Ableitung des Modulators am kleinsten ist, ist die Frequenz des modulierten Signals am Kleinsten.

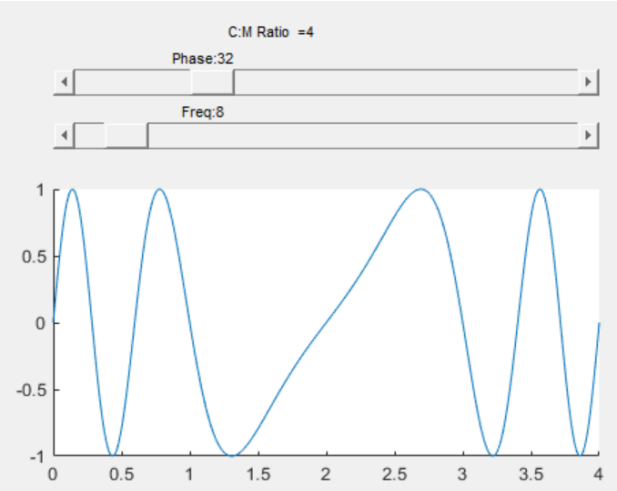



*Abbildung 21: Graphical User Interface (GUI) für parametrisierte Phasenmodulation*

Damit nicht jede mögliche Funktion neu geplottet werden muss, wurde zudem ein GUI erstellt, welches es ermöglicht die Frequenz des Modulators und den Vorfaktor β über einen Slider zu verändern.

Dieses GUI hat zwei Parameter.

Phase: $\beta = \frac{phase}{64}$

Freq: $\omega_{Mod} = \frac{freq}{32}$

## Implementation

Die Implementation auf dem FPGA ist simpel, in zwei Zeilen Code kann die Phasenmodulation implementiert werden. Das schwierige war die Datentypen richtig einzusetzen und die richtige Teilung der Faktoren. Im Tonegenerator wurden, für den Modulator, nochmals zehn DDS instanziiert. Die Frequenz eines Tones wird durch die Erhöhung des Zählerinkrements vergrössert (siehe Milestone 3). Um also die Modulator Frequenz anzupassen muss das Inkrement angepasst werden. Dazu wird das Inkrement des Carriers durch 32 geteilt (shifting um 5) und danach mit einem Faktor, welcher auf dem Keyboard eingestellt werden kann, multipliziert. Der Faktor ist sieben Bit gross und kann somit maximal 127 sein. Somit können Ratios von 1:$\frac{1}{32}$ – 1:4 eingestellt werden.
Der kompliziertere Faktor ist das β. Der Counterinkrement ist ein 19 Bit breiter Speicher, welcher je nach gespieltem Ton erhöht wird. Die Amplitude des Modulators ist 12 Bit gross, das heisst mit einem Vorfaktor von 7 Bit (0-127), kann in der LUT eine ganze Periode übersprungen werden. Das macht für die PM keinen Sinn, aus diesem Grund wird die Amplitude des Modulators geteilt. Somit hat Amplitude des Modulators maximal einen Einfluss von $\frac{1}{2}$ und minimal einen Einfluss von $\frac{1}{256}$ der Periode.
Eine weitere Herausforderung war der korrekte Umgang mit den Datentypen signed, unsigned, std_logic_vector und integer. Um einen unsigned std_logic_vector in einen signed std_logic_vector zu konvertieren, muss er zuerst in einen integer konvertiert werden. Danach kann er dann mit der Funktion to_signed() in einen signed Wert umgewandelt werden.Die beiden Faktoren für die Frequenz des Modulators und das β, können über das Keyboard eingestellt werden.

| | |
|---|---|
| *Data-Entry:* | *Frequenz des Modulators* |
| *Modulation Wheel:* | *Vorfaktor für $\beta$* |

### Testen

Um die Testbench für dieses Feature zu nutzen, muss im Modelsim der testcase_ms3.dat mit der Wave wave_ms3.do verwendet werden. Die Testbench dient rein zur visuellen Prüfung. Damit das Feature manuell getestet werden kann, müssen die Switches wie folgt eingestellt werden: SW(3..0): 0, SW_15: 1, SW(7..4): 0 (Für den Sinus)
Nach dem Hochladen auf den FPGA muss der KEY 0 (reset) und danach der KEY 1 (laden Audio-CODEC). Zudem muss der Midi Controller über den GPIO Pin 26 verbunden werden. Auf dem Keyboard kann die Frequenz des Modulators über das Data Entry verändert werden. Wie einen grossen Einfluss der Modulator hat kann über das Modulation Wheel eingestellt werden.

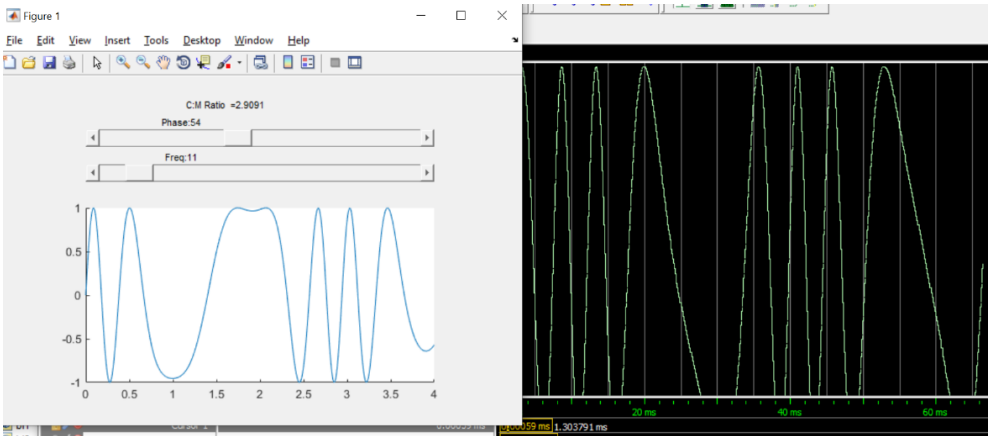



*Abbildung 22: Auszug aus der Testbench für eine Phasenmodulation*

### Optimierung des Features

Ressourcen mässig kann für dieses Feature nichts mehr optimiert werden. Jeder DDS speichert sich alle LUT ab, diese benötigen sehr viel Speicher. Da für die Modulator DDS aber nur die LUT für den Sinus gebraucht wird, könnte dort viel Speicher gespart werden, wenn ein eigener DDS Block für den Modulator erstellt würde. Quartus merkt aber selbst, dass die anderen LUT gar nie gebraucht werden und speichert sie deshalb auch nicht ab.

| Compilation Hierarchy Node | Combinational ALUTs | Dedicated Logic Registers |
|---|---|---|
| \|dds:\dds_carrier_gen:5:carrier_dds\| | 3827 (3827) | 19 (19) |
| \|dds:\dds_carrier_gen:6:carrier_dds\| | 3827 (3827) | 19 (19) |
| \|dds:\dds_carrier_gen:7:carrier_dds\| | 3825 (3825) | 19 (19) |
| \|dds:\dds_carrier_gen:8:carrier_dds\| | 3826 (3826) | 19 (19) |
| \|dds:\dds_carrier_gen:9:carrier_dds\| | 3826 (3826) | 19 (19) |
| \|dds:\dds_modulator_gen:0:modulator_dds\| | 53 (53) | 19 (19) |
| \|dds:\dds_modulator_gen:1:modulator_dds\| | 53 (53) | 19 (19) |

*Abbildung 23: Auszug aus der "Rousource Utilisation by Entity" für die DDS*

Um die Benutzerfreundlichkeit zu steigern können noch zwei weitere Extras eingebaut werden. Zum einen könnte man durch einen Switch die Modulation komplett ausschalten und zum anderen könnte man den Teilungsfaktor für die Modulator Frequenz und das β ebenfalls über Switches einstellen.

# 3 Schluss

Das Projekt wurde erfolgreich erstellt und funktioniert einwandfrei. Die Grundeinstellungen Meilenstein 1-4 sind fehlerfrei und erfüllen ihre Aufgabe. Das Gleiche gilt auch für die Zusatzfunktionen. Die Zusatzfunktionen könnten noch optimiert werden. Die Optimierungen wurden in den jeweiligen Abschnitten erklärt.

# 4 Quellenverzeichnis

[1] Kholomiov, Anton (2016): sharc-timbre: Sandell Harmonic Archive. A collection of stable phases for all instruments in the orchestra. URL: https://hackage.haskell.org/package/sharc-timbre [Stand: 11.06.2020]

[2] Wikipedia (2018): FM Synthese. URL: https://de.wikipedia.org/wiki/FM-Synthese [11.06.2020].

# 5 Anhang

## 5.1 Projektevaluation

Wir sind sehr zufrieden mit unserem Projekt. Die Arbeit in der Gruppe machte Spass, auch wenn unsere Arbeitsweise durch ungeplante Ereignisse durcheinandergebracht wurde. Wir sind sehr gut ins Projekt gestartet und sind dann auch schnell vorwärtsgekommen. Als dann der Lockdown wegen Corona kam, hatten wir am Anfang Schwierigkeiten mit der Zusammenarbeit. Vor allem weil dann Tharmelan für mehrere Wochen ins Militär musste. Durch die Einrichtung einer WhatsApp Gruppe, konnten wir uns dann besser absprechen und haben Termine vereinbart, an welchen wir über Teams zusammenarbeiten. An diesen Terminen haben wir sehr von der Teams-Funktion, den Bildschirm zu teilen, profitiert. Mit *Pair programming* kamen wir sehr schnell voran. Zudem wusste jeder was genau implementiert wurde. Wir waren so auf die Fertigstellung des Produktes fokussiert, dass wir leider das Kommentieren und Dokumentieren verschleift haben. Das hat uns am Ende sehr viel Zeit gekostet, da wir alle Überlegungen, welche wir uns während des Projektes gemacht haben, nochmals aufarbeiten mussten.
Wir haben viel über die Zusammenarbeit gelernt, wenn man sich nicht treffen kann. Zudem hat sich in diesem Projekt auch wieder gezeigt, dass man mit der Planung flexibel sein muss und bei einem unvorhergesehenen Ereignis frühzeitig umplanen muss.

Uns hat diese Projektarbeit sehr gefallen und wir haben nicht nur neue Tools und Sprachen kennen gelernt, sondern auch viel über unsere Gruppendynamik. Wenn wir etwas gelernt haben, dann ist es, dass wir die Dokumentation parallel zum Programmieren schreiben sollten.

## 5.2 Verwendete Tools

| Programm | Version |
|---|---|
| Quartus Prime | 18.0.0 Build 614 |
| ModelSim | 2016.10 |
| Matlab | R2018a (9.4.0.813654) |
| Java | Java™ SE Runtime Environment (build 1.8.0_151-b12) |
| Draw.io (Online-Tool) | 13.2.3 |

## 5.3 Pinbelegung FPGA

DE2_115_pin.csv