Zürcher Hochschule
für Angewandte Wissenschaften

School of
Engineering
InES Institute of
Embedded Systems

PA15_gelk_1 Polyphonic DDS Synthesizer mit MIDI Steuerung

Zürcher Hochschule für Angewandte Wissenschaften

Institute of Embedded Systems

Autoren	Katrin Bächli
Hauptbetreuer	
Nebenbetreuer	
Datum	2. Dezember 2015

Kontakt Adresse
c/o Inst. of Embedded Systems (InES)
Zürcher Hochschule für Angewandte Wissenschaften
Technikumstrasse 22
CH-8401 Winterthur

Tel.: +41 (0)58 934 75 25
Fax.: +41 (0)58 935 75 25

E-Mail: katrin.baechli@zhaw.ch
Homepage: `http://www.ines.zhaw.ch`

Inhaltsverzeichnis

Liste der noch zu erledigenden Punkte

1. Einleitung

Die Projektarbeitet bietet die Möglichkeit, sich vertieft in VHDL einzuarbeiten. Mit vertieft ist das Erstellen eines eignen Projektes wie auch das Kennenlernen der Eigenheiten der Sprache VHDL gemeint.

Der erste Teil der Projektarbeit umfasst die Auseinandersetzung mit der Sprache VHDL und deren eigenen Herausforderungen. Die zwei Fehlerquellen *Glitches* und *Metastabilität* werden künstlich erzeugt, damit ihre Ursache verstanden wird.

Der zweiten Teil der Projektarbeit beinhaltet eine Weiterentwicklung des Synthesizer-Projektes aud der Vorlesung Digitale Technik II. Konkret soll die Frequenzmodulation vertieft und der Midianschluss implementiert werden.

Eine Projektarbeit verdient nicht ihren Namen, wenn am Schluss der Arbeit nicht ausführlich die Resultate diskutiert und die verwendete Literatur genannt wird. Es folgt deshalb ein ausführlicher Anhang, in dem unter anderem auch der verwendete VHDL-Code abgebildet ist.

1.1. Ausgangslage

Nennt bestehende Arbeiten zu diesem Thema (Literaturrecherche)
Stand der Technik: Bisherige Lösungen des Problems und deren Grenzen

1.2. Zielsetzung Aufgabenstellung Anforderungen

Formulierte Ziele der Arbeit
Verweist auf die **offizielle Aufgabenstellung des/der Dozenten im Anhang**
EV.Pflichtenheft, Spezifikation
EV. übersicht über die Arbeit: stellt die folgenden Teile kurz vor.

2. Glitches

2.1. Definition Glitches

Im technischem Bereich bedeutet gemäss Cambridge Dictionaire ein *glitch*, eine ungewollte, flüchtige Signalspitze, die ein Fehlverhalten im System verursacht. Im Anhang A befindet sich der Originaltext wie auch noch eine weitere Defintion aus dem englischen Sprachraum.

In der digitalen Signalverarbeitung ist das Glitch ein bekannter Begriff und wird dort unter anderem leicht sarkastisch beschrieben:

"Als "Glitch" wird eine ungewollte, flüchtige "Signalspitze" bezeichnet, die Zähler aufwärts zählt, Register löscht oder einen ungewollten Prozess startet." (Fletcher, Digital design, 472)

Am intuitivsten ist die bildliche Darstellung des soeben beschriebenen Fehlverhaltens (Abbildung 2.1). In dieser Signalabfolge treten zwei mal Glitches auf, die eigentlich nicht dort hingehören.

besseres Bild def Glitch

Abbildung 2.1.: Glitch-Signalspitzen

Auf den ersten Blick scheinen solch temporäre Spannungsspitzen nicht zu stören. Doch wenn man Pech hat, sind Glitches der Auslöser für Abstürze oder zumindest für ein Fehlverhalten eines Gerätes. Aus diesem Grund, wird nun der Ursache dieser Spitzen nachgegangen.

2.2. Ursache für Glitches

Der Auslöser der flüchtigen Spannungsspitzen sind asynchrone Inputs vor einem asynchronen Bauteil oder verzögerte Signale. Trifft z. B. vor einem Dekoder von vier Leitungen, das Signal einer Leitung zu spät an, entschlüsselt der Dekoder kurzfristig einen falschen Wert. Obwohl die Störung nur kurz ist, übermittelt ein asynchroner den falschen Wert direkt an seinen Ausgang.

2.2.1. Asynchroner Input

Das ungleichzeitige Eintreffen von Signalen kann z.B. durch lange Signalpfade (Leitungen), unterschiedliche Durchlaufverzögerungen der vorangehenden Flip-Flops oder unterschiedliche Logik-Zeiten entstehen. Grundsätzlich gelten alle nicht-getakteten Prozesse als potenzielles Risiko für Glitches, da man bei ungetakteten Prozessen nicht weiss, wie lange sie dauern.

2.2.2. Nachteil getakteter Prozesse

Jeder getaktete Prozess verzögert die Verarbeitung. Aus diesem Grund wird abgewogen, wo Prozesse getaktet und wo sie asynchron getätig werden. In VHDL gibt es viele asynchrone Vorgänge (wie ungetaktete Prozesse oder Singalzuweisungen), deshalb ist es vorteilhaft, wenn das Risiko asynchroner Prozesse bekannt ist.

Abbilung 2.2 zeigt ein leicht verzögertes (getaktetes) enable-Signal zu einem anders verzögerten (getakteten) Flip-Flop-Eingangssignal Q. Der Ausgang des Flip-Flops weist kurzzeitig Glitches auf.

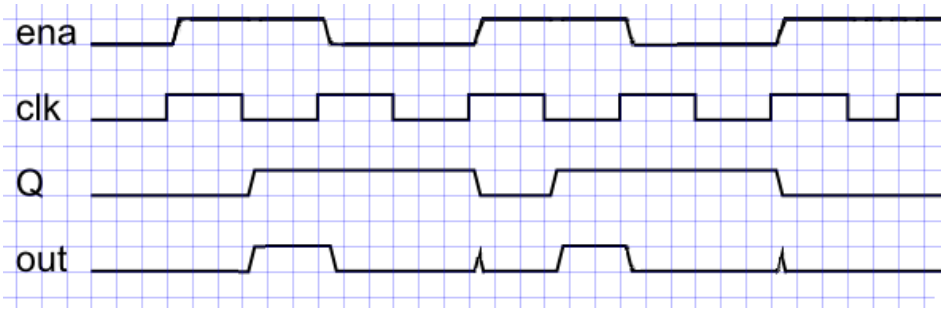

Abbildung 2.2.: Mögliches Bsp. für Glitches

2.3. Glitches erzeugen

Was in Glitch ist, ist relativ einfach zu beschreiben. Ein Glitch jedoch mit moderner Digitaltechnik zu erzeugen, erweist sich als etwas schwerer. Hier zwei Ansätze, die getestet werden.

2.3.1. Glitches Aufgrund von Bauteiltoleranzen

Der erste Ansatz ist, ein Zähler aus vier Flipflops mit asynchronem Dekoder zu implementieren.

Konzept

Die Erwartung ist, dass aufgrund der *Bauteiltoleranzen* der Flip-Flops die vier Ausgänge an den Flip-Flops nicht gleichzeitg ihren Wert übermitteln. Die einen sind leicht schneller, die anderen leicht verzögert. Dadurch ergibt sich kurzzeitig am asynchronen Dekoder einen falschen Wert.
Damit ein abnormaler Wert in einem Zähler erkannt wird, sendet der Dekoder bei der Zahl 7 einen Peak. Ohne Glitch entschlüsselt der Dekoder in regelmässigen Abständen von 160 ns diese Zahl. Aufgrund der Flip-Flop-Bauteiltolereanzen ist ein kurzzeitiges Dekodieren einer 7 *ausserhalb der Periode* T zu erwarten.

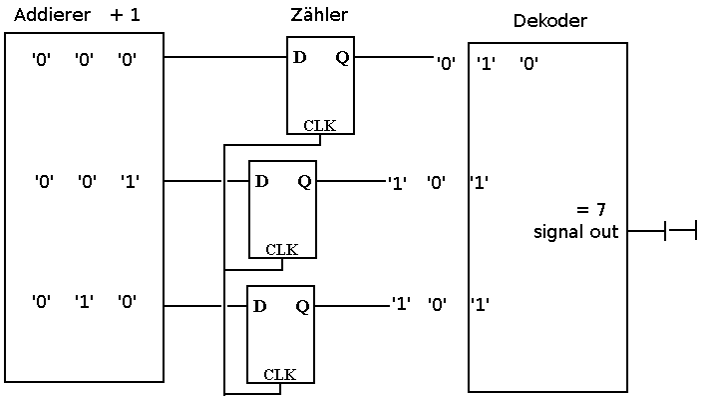

Abbildung 2.3.: Ausnutzen der Bauteilverzögerung

Implementation

Bild anpasse

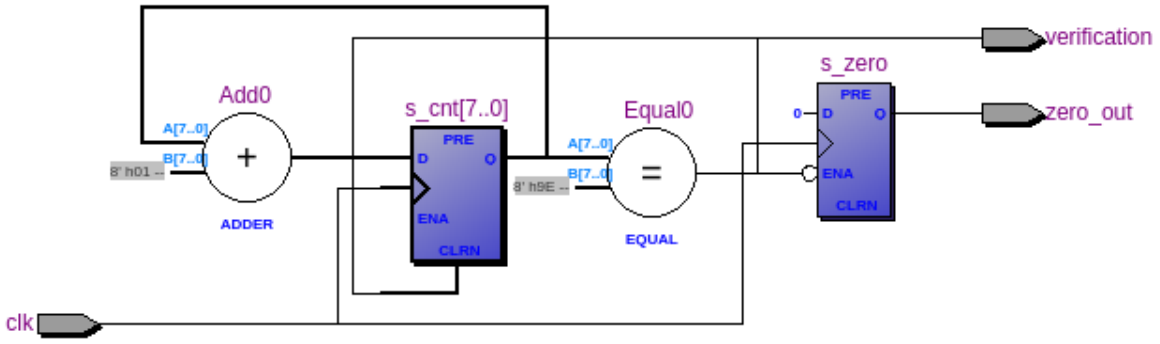

Abbildung 2.4.: RTL Zahler mit asynchronem Dekoder

Bild Asynchnem Zähler besser beschreiben

Um die gewollten Zahlenwerte von den Glitches zu unterschieden, wird das asynchrone Singnal getaktet. Dadurch erscheint der korrekte Zählwert mit einem Takt Verzögerung. Die Periode ist 20 ns (CLK = 50 MHz).

2.3.2. Glitches Aufgrund von Pfadverzögerung

Der zweite Ansatz ist die gesuchte Bauteilverzögerungen über längere Signalpfade zu simulieren.Der Dekoder des Zählers bleibt asynchron.

Konzept

Dekodiert wird die Zahl 15. Durch intelligentes Routing (FF 1 wird verzögert, FF 2 wird beschleunigt) wird der Zustand der Zahl 11 forcier

Bild FF1 verzögert, FF schneller

Implementation

Cyclone II, Board De2. Quartus 13.0sp.

Die Pfad*verlängerung* wird über das Routing über die GPIO-Pins des Headers 1 gemacht (siehe Abbildung 2.6. Die obersten vier Doppel-Pins erhalten eine "Brücke", sodass das Signal links ausgegeben und rechts wieder eingespiesen wird.
Signal*verkürzung* ist eine direkte Signalzuweisung .

korrektes Wo
?(Concouren
Assignment)

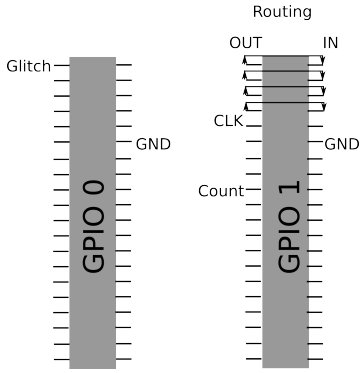

Abbildung 2.5.: GPIO Anschlüsse

Auf dem KO wird das asynchrone Glitch-Signal und das synchrone Zählersignal neben dem Takt ausgegeben. Weil der Zähler synchronisiert wurde, ist der Wert 1 Periode (= 20 ns) später als der Glitch.

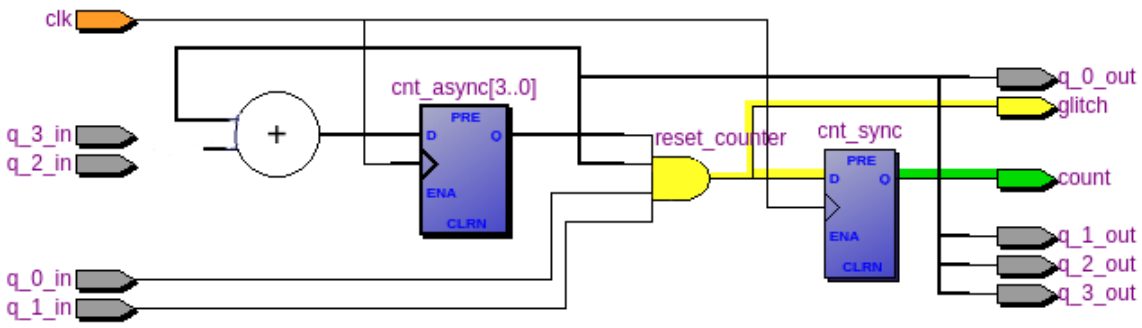

Abbildung 2.6.: Zähler mit Signal-Routing über GPIO

Im RTL-Diagramm sieht man deutlich den Unterschied zwischen dem asynchronen Zähler, der über das Gate *reset_counter* beim Wert 15 einen Impuls an den Ausgang glitch gibt und dem synchronisierten Zähler *cnt_sync* der dem asynchronen Ausgang nachgeschaltet ist und dieses Signal taktet. Das getaktete Zähl-Signal geht an den Ausgang *count*.

2.4. Resultat Glitches provozieren

2.4.1. Erzeugen über Bauteiltoleranzen

Der Ansatz, dass die Bauteiltoleranzen der Flip-flops eine Ursache für asynchrone Inputs in den Dekoder sind ist korrekt. Die Umsetzung zeigte sich jedoch als schwierig, da die heutigen Flip-Flops zu

schnell sind bzw. ihre Toleranzen zu klein um sichtbar zu werden. Aus diesem Grund entschlüsselte der asynchrone Dekoder trotz kleinen Verzögerungen die Werte stets korrekt.

Timeanalyse für FF

2.4.2. Erzeugen über Routing

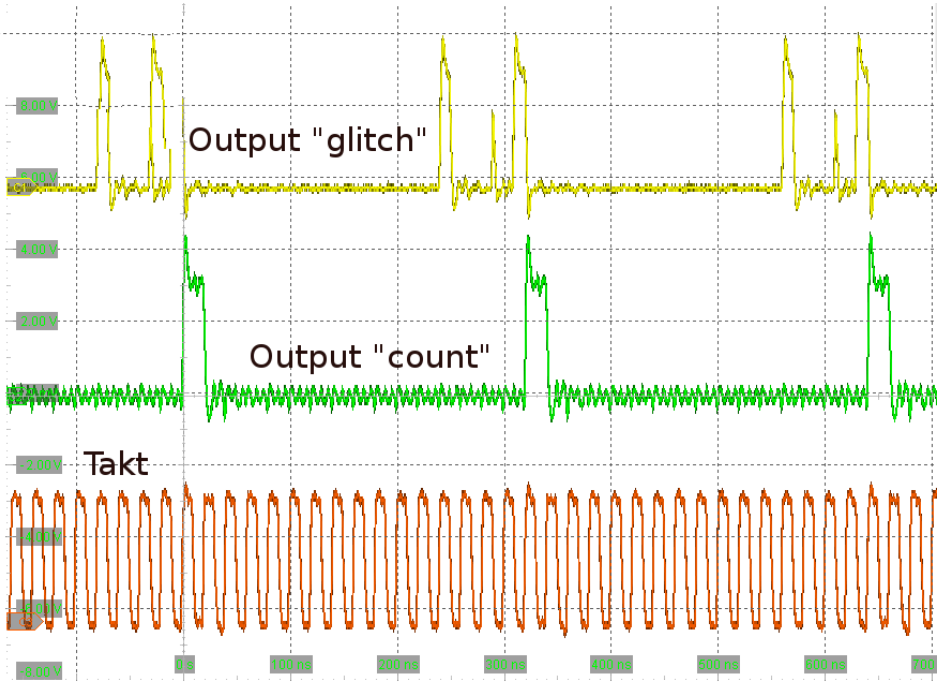

Abbildung 2.7.: Glitch (gelb), Zähler (grün) und Takt (orange)

Typisch ist, dass der synchrone Zähler eine Signalbreite von genau einer Periode hat, da dieses Signal getaktet ist. Dagegen hat der asynchrone Glitch keine konstante Breite.

1. Bei welchen Zählständen treten Glitches auf?
2. Wie hängen die Zählständen mit dem gewählten Routing zusammen?

3. Metastabilität

3.1. Definition Metastabilität

Metastabilität bedeutet, dass der Ausgang eines Flip-Flops nicht dem Eingang entsprechen muss. Wechselt das Inputsignal eines Flip-Flops zur falschen Zeit, ist der Wert des Ausgangssignal unsicher. Hier zwei kurze englische Beschreibungen, dieses Phänomens:

" If data inputs to a flip-flop are changing at the instant of the clock pulse, a problem known as *metastability* may occur. In the metastable case, the flip-flop does not settle in to a stable state" (Camara, S. 32-2)

"If the amplitude of the runt pulse is *exactly the treshold level of the SET input of the output cell*, the cell will be driven to its metastable state. The metastable state is the condition that is roughly defined as "half SET and half RESET" (Fletcher, 482.)

Im besten Fall wählt der Ausgang bei unklarem Eingangssingal selbst einen Wert an ('0' oder '1'). Im schlechten Fall "hängt" sich das Flip-Flop "auf" und toggelt permanent zwischen '0' und '1' oder setzt sogar beide Werte parallel.

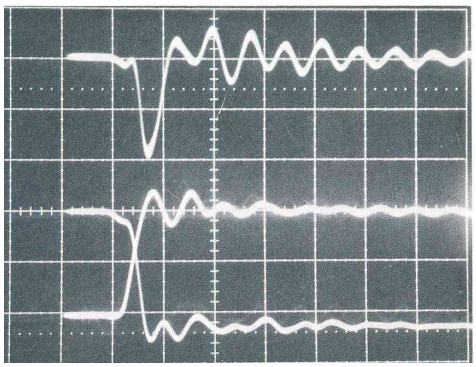

Abbildung 3.1.: Metastabilität schlimmster Fall (Fletcher, 482.)

3.2. Ursache von Metastabilität

Der Grund für Metastabilität ist, dass der angelgte Wert entweder zu spät eintrifft (verletzen der setup-Zeit) oder zu früh wieder verschwindet (verletzen der hold-Zeit). Metastabilität kann vermieden werden, wenn diese zwei Zeiten strikt eingehalten werden:

"Metastabilit is avoided by holding the information stable before and after the clock pulse for a set period of time, called the setup time for the data line an the hold time for the control line."(Camara, S. 32-2)

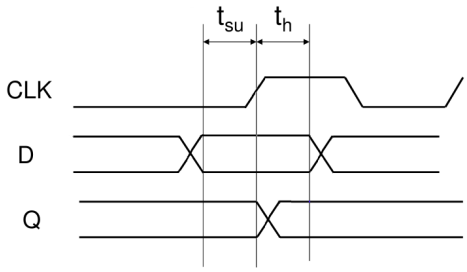

Abbildung 3.2.: Einhalten der Datenzeiten

Es gibt mehrere Gründe für das Nichteinhalten der geforderten setup-Zeit:
- Ein Logikpfad kann zu lange sein, bzw. die Taktfrequenz ist zu schnell
- Zwischen den Bauteilen liegen zu lange Pfade, die das Eintreffen der Daten verzögern
- Ein vorangehendes Bauteil hat eine zu lange Durchlaufverzögerung.

Um Metastabilität zu vermeiden, sollte die Logik möglichst klein gehalten werden, die Bauteile bewusst nahe beieinander platziert und vor allem der Systemtakt an die längste Pfadzeit angepasst werden. Der maximal elaubte Systemtakt kann in quartus mit dem Timequest Time Analyser abgefragt werden.
Als Alternative bietet sich eine Synchronisierungsschaltung an. Zwischen den zwei Takt-Flanken kann sich der metastabile Ausgang erholen und gelangt so stabil in den Verarbeitungspfad. Der Nachteil der Synchronistation ist jedoch, eine um einen Takt längere Verarbeitungszeit.

3.3. Metastabilität erzeugen

3.3.1. Ansatz

Aufgebaut wird ein System mit zwei Takten. Der zentrale Block hat eine Taktfrequenz von 50 MHz und beinhaltet eine State Machine. Diese wechselt bei jedem Impuls von einem Zustand in den anderen (Abbild: 3.4 Um die zwei Zustände zu erkennen, werden beiden Zuständen ein logischer Pegel zugefügt:

- Zustand 1: s0 = Logisch '0'
- Zustand 2: s1 = Logisch '1'

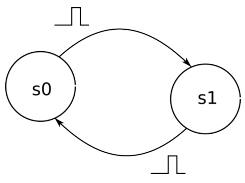

Abbildung 3.3.: Statemachine im zentralen Block

Der Inpuls, der die Statemachine steuert ist asynchron. Er wird von einem Zähler generiert, der mit der Taktfrequenz von 27 MHz läuft. Alle 37 ns sendet der Zähler einen Puls an die State Machine. Die State Machine selbst arbeitet mit einer Taktfrequenz von 20 ns. Der Impuls ist ihr gegenüber asynchron.

Erwartet wird, dass die setup-Zeit der State Machine-Flip-Flops regelmässig verletzt werden.

Abbildung 3.4.: Die zwei Taktzeiten

einsetzen set
zeit gemäss
glossar für ..

3.3.2. Implementation

3.4. Resultat Metastabilität provozieren

Was ist das Ergebnis beim Verletzen der setup Zeit? Beide Ausgänge immer an? Keiner von beiden? aufhängen des Systems? (Keine LED geht mehr).

Synchronisation Schaltung erhärtet die These b

4. MIDI Steuerung

4.1. Einleitung

Im Modul DTP2 entwickeln Studentinnen und Studenten ihren eignenen Synthesizer. Eine Option in diesem Projekt ist es, den Synthesier über ein Keyboard per Midi steuern zu können.
Am Institut bestand bereits die MIDI UART und die Aufgabe im zweiten Teil dieser Projektarbeit ist es, die MIDI Steuerung zu entwickeln, die sowohl einezelne Töne weiterleitet wie auch polyphoniefähig ist (siehe nächstes Kapitel).

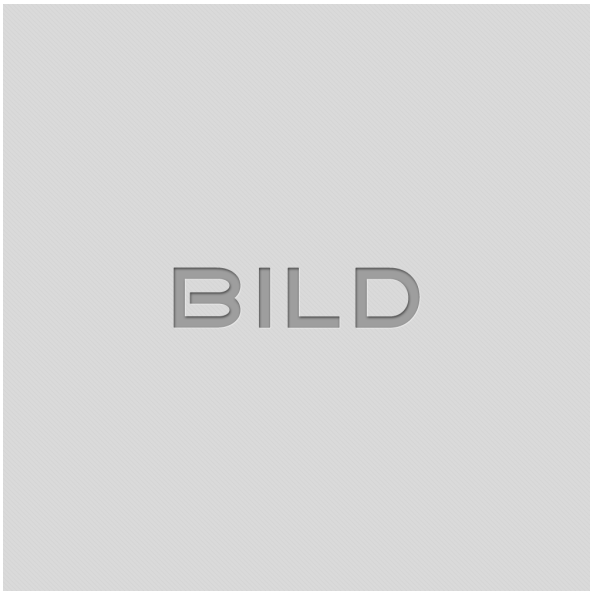

Abbildung 4.1.: blabla

4.2. Spezifikation Midi Protokoll

4.3. Umsetzung Midi Interface

4.3.1. Aufbau der Blöcke

4.3.2. Schnittstellen

4.3.3. Inhalt des zu entwickelnden Midi Controllers

4.3.4. Inhalt des zu entwickelnden Polyphonie Blocks

5. Polyphonie

Abbildung 5.1.: Bildbeschreibung ....

5.1. Midi Spezifikation

5.2. Umsetzung

5.2.1. software nahe

hardware nahe

6. Resultate der Projektarbeit

Zusammenfassung der Resultate

6.1. Generieren von Glitches

Beides erreicht. Viel Aufwand, da wenig Wissen wie ungewollter Zustand erzeugt werden kann. Es dauerte 4 Wochen (15. oktober + Doku), der insgesammt 16 Wochen PA.

6.2. Zustand von Metastabilität provozieren

Beides erreicht. Viel Aufwand, da wenig Wissen wie ungewollter Zustand erzeugt werden kann. Es dauerte 4 Wochen (15. oktober + Doku), der insgesammt 16 Wochen PA.

6.3. MIDI Controller entwickeln

6.4. Polyphonie Block

Midiansteuerung nach vielen Redesignes gelungen. Mehr in der Software geübt, ist das Timing in VHDL übungsbedürftig.

6.5. DDS Generatoren basierend auf Frequenzmodulation entwickeln

Sitzung: Nur 10 DDS einbauen. Schnittstellen da. Nicht da,Frequenzmodulation anstelle von LUT. Aus zeitgründen. Nur erster Entwurf, wie es umzusetzen ist.

6.6. Textbasierte Testbench für alle entwickelten Blocks

7. Diskussion und Ausblick

Bespricht die erzielten Ergebnisse bezüglich ihrer ERwartbarkeit, Aussagekraft und Relevanz
Interpretation und Validierung der Resultate
Rückblick auf Aufgabenstellung: erreicht nicht erreicht

Legt dar, wie die Resultate weiterhin genutzt werden können
an sie angeschlossen werden kann

Verzeichnis

7.1. Literaturverzeichnis

7.2. Glossar

Durchlaufverzögerung
Wird englisch *propagation delay* genannt und bezeichnet die Zeit, die Daten vom Eingang bis zum Ausgang des Bauteils brauchen.
Die Durchlaufverzögerung beträgt beim Cylone IV 4 ns (Device Handbook, S. 8-19).

hold time
Ist die minimale Zeit, in der die Inputdaten *nach* der Taktflanke stabil sein müssen.
Die hold-Zeit beträgt beim Cyclone IV E 0 ns (Device Handbook, S. 8-19).

Pfadzeit
... (Unter 3.2. Metastabilität Ratschläge erwähnt)

quartus
IDE von altera zum Kompilieren, Synthesizieren und einbauen von IPs für die altera FPGAs.

setup time
minimale Zeit, in der Inputdaten stabil sein müssen be*vor* ein Taktflanke die Daten triggert.
Die setup-Zeit beträgt beim Cyclone IV E 10 ns (Device Handbook, S. 8-19)

A. Anhang: Englische Definitionen Glitches

A.1. Offizielle Aufgabenstellung

Projektauftrag

A.2. CD mit vollständigem Bericht

B. Anhang: VHDL-Code Glitch detect

C. Anhang: VHDL-Code Metastability detect

C.1. Bliblibli

- sfasdfasdfasfasf