

PA15_gelk_1 Polyphonic DDS Synthesizer mit MIDI Steuerung

ZÜRCHER HOCHSCHULE FÜR ANGEWANDTE
WISSENSCHAFTEN

INSTITUTE OF EMBEDDED SYSTEMS

Autoren Katrin Bächli

Hauptbetreuer

Nebenbetreuer

Datum 16. Oktober 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

1	Einleitung	3
2	Glitches	4
2.1	Definition Glitches	4
2.2	Ursache für Glitches	4
2.2.1	Asynchroner Input	4
2.2.2	Nachteil getakteter Prozesse	5
2.3	Glitches erzeugen	5
2.3.1	Glitches Aufgrund von Bauteiltoleranzen	5
2.3.2	Glitches Aufgrund von Pfadverzögerung	6
2.4	Resultat Glitches provozieren	7
2.4.1	Erzeugen über Bauteiltoleranzen	7
2.4.2	Erzeugen über Routing	8
3	Metastabilität	9
3.1	Definition Metastabilität	9
3.2	Ursache von Metastabilität	9
3.3	Metastabilität erzeugen	10
3.3.1	Ansatz	10
3.3.2	Implementation	11
3.4	Resultat Metastabilität provozieren	11

Liste der noch zu erledigenden Punkte

besseres Bild def Glitch	4
Bild anpassen	6
Bild Asynchronem Zähler besser beschreiben	6
Bild FF1 verzögert, FF2 schneller	6
korrektes Wort ?(Concourent Assignment)	7
Timeanalyse für FF	8

1 Einleitung

Die Projektarbeit 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 aus 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.

2 Glitches

2.1 Definition Glitches

Im technischen Bereich bedeutet gemäss Cambridge Dictionaire ein *glitch*, eine ungewollte, flüchtige Signalspitze, die ein Fehlverhalten im System verursacht. Im Anhang ?? 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.



Abbildung 2.1: Glitch-Signalspitzen

besseres Bild
def Glitch

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 getätigt werden. In VHDL gibt es viele asynchrone Vorgänge (wie ungetaktete Prozesse oder Singalzuweisungen), deshalb ist es vorteilhaft, wenn das Risiko asynchroner Prozesse bekannt ist.

Abbildung 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.

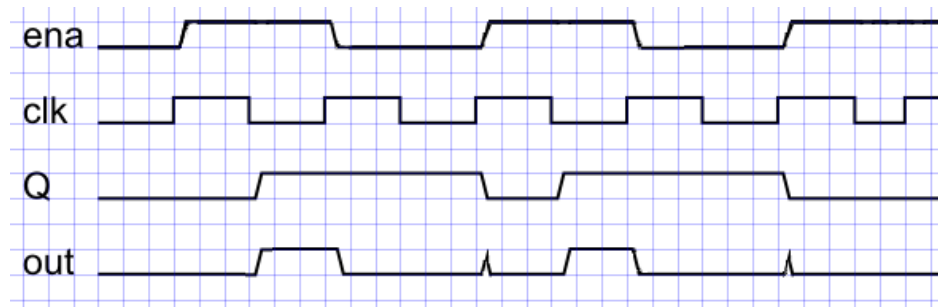


Abbildung 2.2: Mögliches Bsp. für Glitches

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 gleichzeitig 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-Bauteiltoleranzen ist ein kurzzeitiges Dekodieren einer 7 *ausserhalb der Periode T* zu erwarten.

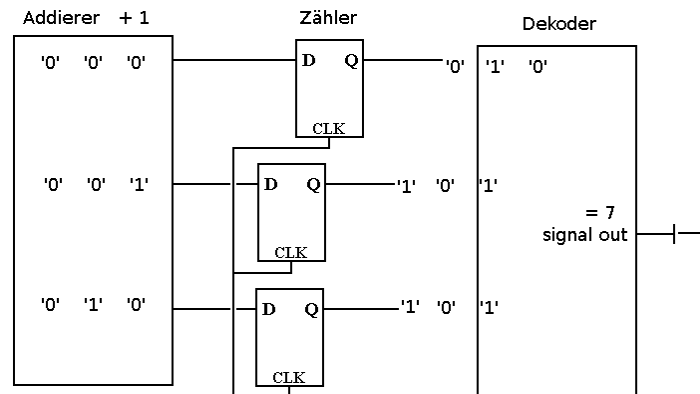


Abbildung 2.3: Ausnutzen der Bauteilverzögerung

Implementation

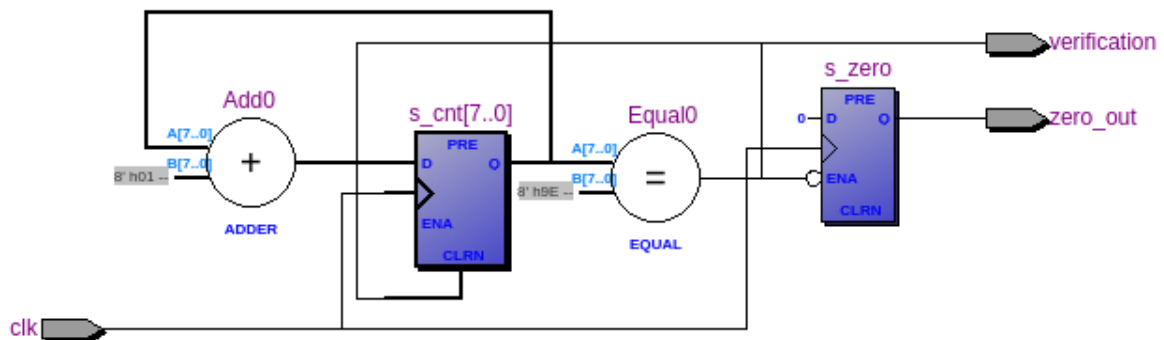


Abbildung 2.4: RTL Zähler mit asynchronem Dekoder

Um die gewünschten Zahlenwerte von den Glitches zu unterscheiden, wird das asynchrone Signal getaktet. Dadurch erscheint der korrekte Zählwert mit einer Taktverzögerung. Die Periode ist 20 ns (CLK = 50 MHz).

2.3.2 Glitches Aufgrund von Pfadverzögerung

Der zweite Ansatz ist die gesuchte Bauteilverzögerungen über längere Signalfade 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 forciert.

Bild anpassen

Bild Asynchronem Zähler besser beschreiben

Bild FF1 verzögert, FF2 schneller

Implementation

Cyclone II, Board De2. Quartus 13.0sp.

Die Pfadverlängerung 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 eingespiessen wird.

Signalverkürzung ist eine direkte Signalzuweisung .

korrektes W
?(Concouren
Assignment)

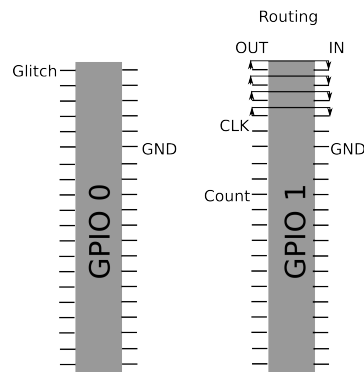


Abbildung 2.5: GPIO Anschlüsse

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.

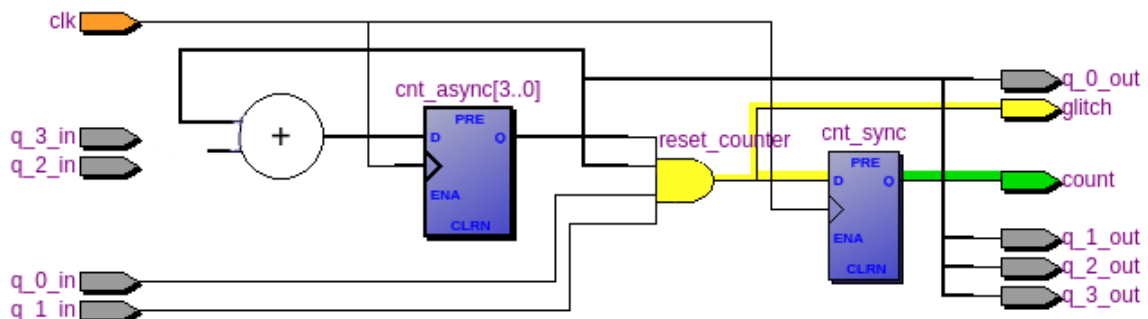


Abbildung 2.6: Zähler mit Signal-Routing über GPIO

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

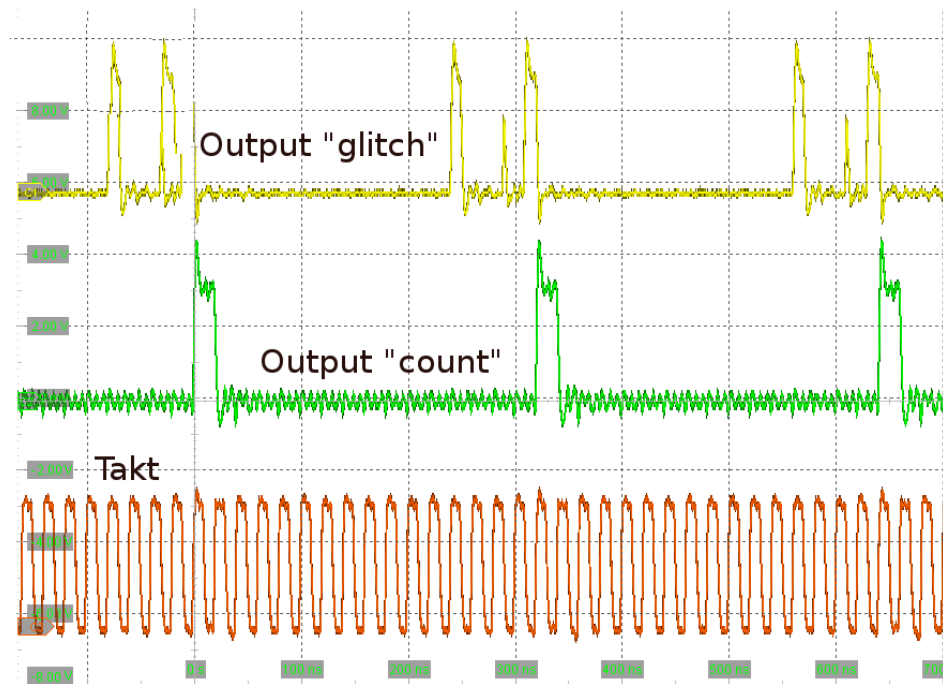


Abbildung 2.7: Glitch (gelb), Zähler (grün) und Takt (orange)

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 Zählstände 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 threshold 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 Eingangssignal 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.

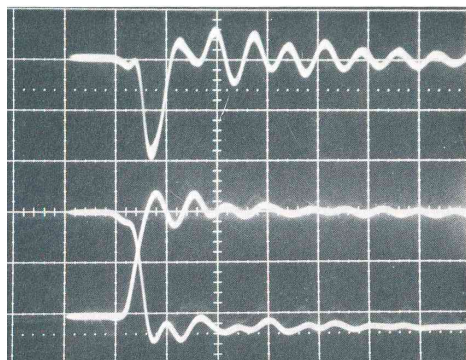


Abbildung 3.1: Metastabilität schlimmster Fall (Fletcher, 482.)

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:

"Metastability 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 and the hold time for the control line." (Camara, S. 32-2)

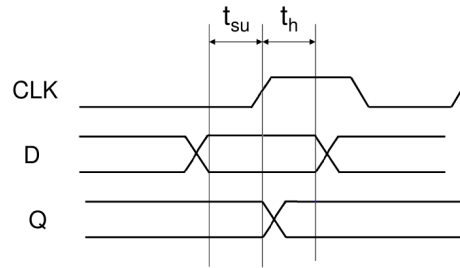


Abbildung 3.2: Einhalten der Datenzeiten

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 erlaubte 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 Synchronisation 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: 3.3 Um die zwei Zustände zu erkennen, werden beiden Zuständen ein logischer Pegel zugefügt:

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

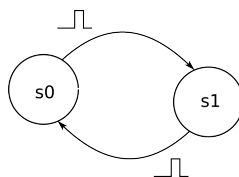


Abbildung 3.3: Statemachine im zentralen Block

Der Impuls, 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

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

einsetzen set
zeit gemäss
glossar für ..

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 Cyclone 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 *bevor* ein Taktflanke die Daten triggert.

Die setup-Zeit beträgt beim Cyclone IV E 10 ns (Device Handbook, S. 8-19)