FH Bielefeld
University of
Applied Sciences

Fachhochschule Bielefeld
Fachbereich Ingenieurwissenschaften und Mathematik
Studiengang Ingenieurinformatik

Titel der Ausarbeitung

Art der Ausarbeitung

Name der/s Autors/in bzw. Autoren/innen inkl. Matrikelnummer

4. März 2022

Betreuer:
Prof. Dr. Axel Schneider
Dr. Hanno Gerd Meyer

In dieser Studienarbeit wird die Implementation eines Funktionsgenerators mithilfe der Hardwarebeschreibungssprache VHDL geschildert. Der Aufbau des Generators wird erläutert sowie seine korrekte Funktion überprüft. Darüber hinaus werden seine Leistungsdaten erfasst und dargelegt. Es wird gezeigt, dass

A short abstract in english.

Inhaltsverzeichnis

1 Einleitung[1]

Diese Studienarbeit behandelt die Konzipierung und Implementierung eines digitalen Funktionsgenerators in der Hardwarebeschreibungssprache VHDL. Ein Funktionsgenerator ist ein elektronisches Bauteil, das in der Lage ist, verschiedene Spannungsverläufe an seinem Ausgang auszugeben. Diese Spannungsverläufe entsprechen einer mathematischen Funtkion. Z. B. kann ein Funktionsgenerator genutzt werden, um ein Rechteck-Signal mit einer bestimmten Frequenz auszugeben, dass dann als Auslöser für eine Kamera fungiert, sodass die Kamera Bilder in einem bestimmten zeitlichen Abstand aufnimmt. Im Normalfall würde ein Funtkionsgenerator als Digitalschaltung in einen Chip integriert oder auf eine Platine gelötet werden. Einen anderen Ansatz zum Bau von digitalen Schaltungen bieten "Free Programmable Gate Arrays", kurz FPGA. Auf diesen ICs befinden sich verschiedene Bausteine die durch Anlegen einer Programmierspannung miteinander verknüpft werden können. Somit ist es möglich, verschiedenste Schaltungen auf demselben IC zu verwirklichen. Die Schaltungen können mithilfe einer Beschreibungssprache designed werden. Eine dieser Sprachen ist VHDL ("Very Highspeed Hardware Description Language"), welche in dieser Studienarbeit verwendet werden soll.

2 I

m Folgenden sollen die verschiedenen Funktionsmerkmal des Funktionsgenerators erläutert werden. Anschließend wird sein Konzept anhand des grundlegenden Aufbaus des Funktionsgenerators erklärt.

2.1 Funktionsmerkmale

2.1.1 Funktionen

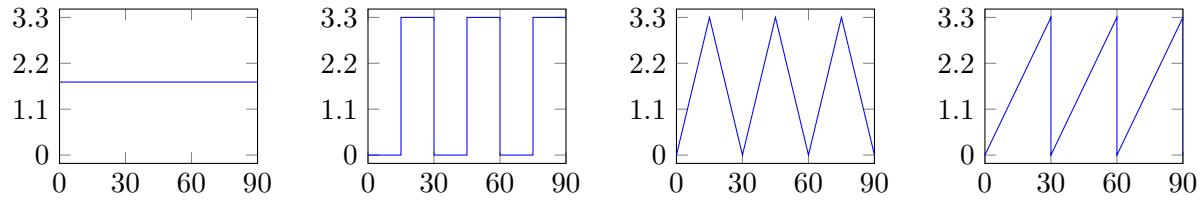
der Generator kann auf vier verschiedene Funktionsbausteine zurückgreifen, die jeweils eine Funktion ausgeben:

1. **Konstante**
   Die konstante analoge Spannung `high` liegt am Ausgang an.
2. **Rechteck-Funktion**
   Der Wert wechselt zwischen `high` und `low` in der Frequenz f. Der Anteil der Zykluszeit T, in dem der Ausgang auf `high` ist, wird über den dutycycle eingestellt. Mit dieser Funktion lassen sich so PWM-Signale erzeugen.
3. **Zick-Zack-Funktion**
   Der Analogwert steigt von `low` bis `high` linear an, erreicht er `high`, fällt der Analogwert wieder kontinuierlich auf `low` ab. Somit schwankt der Ausgangspegel mit der Frequenz f.
4. **Rampen-Funktion**
   Der Analogwert wächst, wie bei der Zick-Zack-Funktion, linear bis auf `high` an, dann fällt er aber auf `high` zurück. Alternativ kann die Rampe auch von `high` her abfallen und bei Erreichen von `low` wieder auf `high` zurück springen.

2.1.2 Konfiguration

2.2 Aufbau

Der Funktionsgenerator ist aus verschiedenen Einzelkomponenten zusammengesetzt. Ihre Verschaltung im Generator ist in Abb. 2.2 zu sehen. Hier kann man gut erkennen, dass die verschiedenen Funktionen parallel arbeiten und von der Konfigurationsschnittstelle mit den Signalen `cyc_ticks`, `high`, `low`, `thresh` und `direction` gesteuert werden. Das Signal `waveform` steuert den Multiplexer, der die Ausgabesignale `y_out` an den DAC-Wandler weitergibt.



(a) konstante Funktion (b) Rechteckfunktion (c) Zick-Zack-Funktion (d) Rampenfunktion

Abbildung 2.1: Funktionen

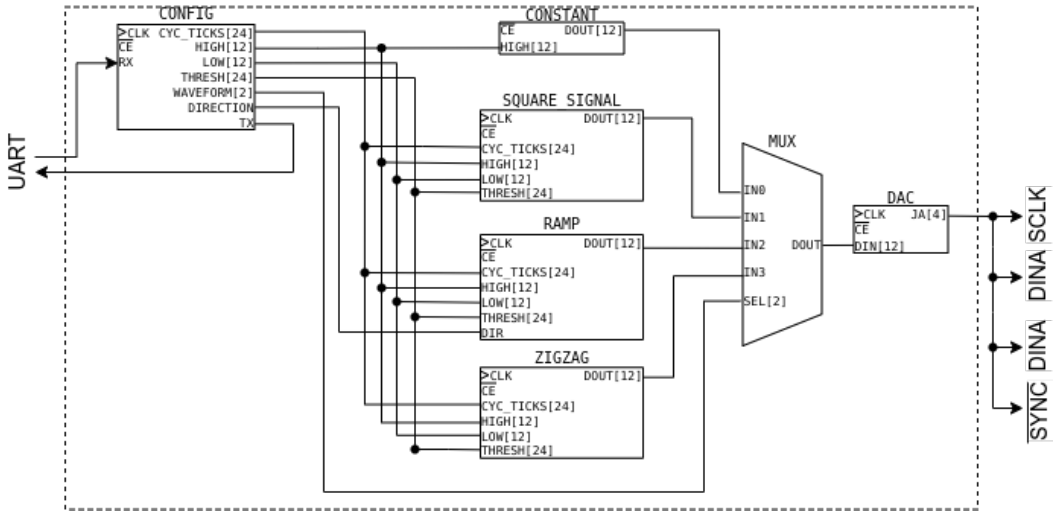


Abbildung 2.2: Diagramm des Funktionsgenerators, zusammengesetzt aus seinen Einzelkomponenten. Aus Gründen der Übersichtlichkeit sind die `CLK` Signale und $\overline{\texttt{CE}}$ Signale nicht angeschlossen dargestellt. Alle `CLK` Signale sind an den Systemtakt angeschlossen und alle $\overline{\texttt{CE}}$ Signale liegen auf Masse.

3 Komponenten

Nachdem im vorherigen Kapitel der grundlegende Aufbau des Funktionsgenerators erklärt wurde, sollen nun die einzelnen Komponenten aus denen der Funktionsgenerator aufgebaut ist genauer erläutert werden.

3.1 Arithmetik

Der Funktionsgenerator braucht für die richtige Berechnung des Ergebnisses Komponenten, mit deren Hilfe mathematische Operationen ausgeführt werden können. Zum Addieren, Subtrahieren und Multiplizieren vorzeichenloser Zahlen können die VHDL Standardfunktionen des Datentyps `unsigned` verwendet werden. Darüber hinaus benötigte Funktionen, die über eine eigene Implementierung verfügen, werden im Folgenden vorgestellt.

3.1.1 Zähler

Der Zähler bildet die Basiskomponente der Funktionskomponenten. Er eignet sich sehr gut, um die Zeit zu messen, da sich die Zeit seit dem letzten Rücksetzen des Zählers T_R immer aus dem Zählstand N und der Taktzeit des Zählers T_{count} berechnen lässt: $T_R = N \cdot T_{count}$. So kann durch wiederholtes Rücksetzen des Zählers bei einem bestimmten Zählstand ein zyklischer Funktionsverlauf erstellt werden. Der Zählstand selbst dient in diesem Fall als diskrete X-Komponente, der dann von der Funktionskomponente ein Y-Wert zugeordnet wird. Der hier implementierte Zähler kann allerdings schon selbst als Funktionskomponente mit dem Ausgang `o_count` für die Funktion $o\_count = inc \cdot N \cdot (-1)^{1-D}, N \in \{0, 1, 2, \ldots, max\_ticks\}$ gesehen werden. Die Komponente besitzt nämlich die Eingänge `inc`, `D` und `max_ticks`. Diese erfüllen folgende Aufgabe:

- `max_ticks`: Bit-Vektor, der den maximalen Wert repräsentiert, ab dem der Zähler automatisch zurückgesetzt wird. Diese Funktion erleichtert es, die Zykluszeit einer Funktion T_{func} als Zeitraum zwischen zwei Rücksetzungen festzulegen: $T_{func} = max_{ticks} \cdot T_{func}$
- `D`: Richtung, in die der Zähler Zählen soll: 0 heißt aufwärts, 1 heißt abwärts
- `inc`: Bit-Vektor, er repräsentiert das Inkrement, um das der Zähler hoch- bzw. runterzählen soll

Der maximale Zählstand ist durch die Anzahl der Stellen von `o_count` beschränkt. Sie kann generisch durch den Wert `data_width` bestimmt werden.

```
Beginn des Algorithmus
// verhindern, dass durch 0 geteilt wird:
wenn N ungleich 0, dann
    setze Q = 0
    setze R = 0
    zähle i von n - 1 bis 0 runter  // n ist die Anzahl der Bits in N
        schiebe R um 1 Bit nach links
        // letzte Stelle von R wird ite Stelle des Zählers:
        R(0) = Z(i)
        wenn R >= N, dann
            setze R = R - N
            setze Q(i) = 1  // ite Stelle des Quotienten wird 1
        Ende der bedingten Anweisung
    springe zum Anfang der Schleife
Ende der bedingten Anweisung
Ende des Algorithmus
```

Abbildung 3.1: Pseudocode zur Beschreibung des Teilungsalgorithmus.

Würde es im nächsten Takt dazu kommen, dass der Zählstand größer als `max_ticks` wird (Überlauf), setzt er sich automatisch wieder auf Null zurück. Würde es im nächsten Takt dazu kommen, dass der Zählstand kleiner als Null wird (Unterlauf), wird der Zähler auf `max_ticks` zurückgesetzt. Zusätzlich kann der Zähler asynchron auf Null gesetzt werden, wenn der Eingang $\overline{\mathtt{R}}$ auf 0 gesetzt wird.

3.1.2 Teiler

Während beim Multiplizieren, Addieren und Subtrahieren auf Standardfunktionen zurückgegriffen wird, wird für die Division zweier Binärzahlen eine eigene Komponente entworfen. Die Gründe hierfür werden in Abschnitt 3.3.3 erläutert.
Diese Komponente kann die Division des Zählers Z durch den Nenner N lösen, dabei sind Z und N zwei Binärzahlen mit gleicher Stellenanzahl und ohne Vorzeichen. Das Ergebnis ist der Quotient Q und der Rest R. Der Algorithmus, den die Komponente ausführt, ist in Pseudocode in Abb. 3.1 beschrieben. Er ermittelt pro Rechenschritt (d.h. pro Takt) i eine Stelle des Quotienten , indem er den Zähler von der größten Stelle $Z(n)$ bis zur kleinsten Stelle $Z(0)$ durchgeht. Dabei schiebt er immer die aktuelle Stelle $Z(i)$ in den Rest, sodass der neue Rest im binären $R' = R \cdot 2 + Z(i)$ ist. Immer wenn der Rest größer wird als der Nenner, ist das Ergebnis der Division größer Null, also wird der neue Rest durch Subtraktion mit dem Nenner gebildet: $R'' = R' - N$. Ist eine Subtraktion möglich, wird der Quotient an der Stelle i zu 1 ($Q(i) = 1$), ansonsten gilt $Q(i) = 0$. Es gilt dabei aber zu beachten, dass alle Schritte innerhalb der Schleife in einem Takt ausgeführt werden können, da es sich bei der Komponente um eine digitale Schaltung handelt.

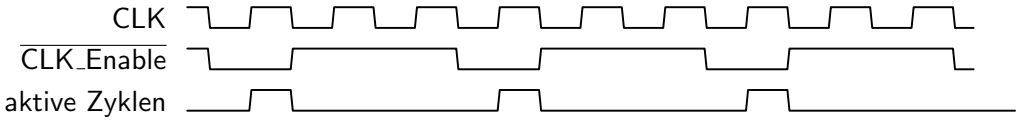


Abbildung 3.2: Beispielhaftes Timing-Diagramm. Jeder vierte Taktzyklus wird durch `CLK_Enable` aktiviert.

3.2 Takterzeugung

Da die Einzelkomponenten des Funktionsgenerators in verschiedenen Geschwindigkeiten arbeiten, braucht es Komponenten für das Clock-Management.

3.2.1 Clock-Enable

Um aus dem Systemtakt Takte mit niedrigerer Frequenz zu erzeugen, wird die Komponente `SCLK_ENABLE` verwendet. Der Takt wird geteilt, indem ein sogenanntes Enable-Signal `SCLK_EN` erzeugt wird, das die jeweils nte steigende Flanke des Systemtakts mit `low` maskiert. Der Eingang `CE` der Komponente, die im erzeugten Takt arbeiten soll, wird an `SCLK_EN` angeschlossen, so dass die Komponente nur dann aktiv ist, wenn eine steigende Flanke detektiert wurde und `CE` `low` ist. Das Timingdiagram in Abbildung Abb. 3.2 verdeutlicht, auf welche Taktzyklen die angeschlossene Komponente reagiert. Der große Vorteil dieser Methode liegt darin, dass jede Komponente weiterhin direkt an den Systemtakt angeschlossen ist und dieser somit nicht durch zwischengeschaltete Komponenten verzögert wird.

3.3 Funktionen

Die Herzstücke des Funktionsgenerators bilden seine Funktionskomponenten. An ihren Ausgängen liegen die von ihnen berechneten Werte an, von denen einer zum in Abschnitt 3.5 beschriebenen DAC-Konverter per Multiplexer weitergeleitet wird. Bis auf die Konstante verfügen alle Komponenten, neben den üblichen `CLK` und `CE` Eingängen, über folgende Anschlüsse:

- `cyc_ticks`: Eingang, dieser Bit-Vektor legt die Periodendauer einer Funktionskomponente fest.
- `high`: Eingang, dieser Bit-Vektor legt den maximalen Wert des Komponentenausgangs fest.
- `low`: Eingang, dieser Bit-Vektor legt den minimalen Wert des Komponentenausgangs fest.
- `y_out`: Ausgang, Bit-Vektor, der den digitalen Funktionswert, der zum DAC geschickt wird repräsentiert

Alle Komponenten außer der Konstante verfügen über einen Zähler und eine Clock-Enable-Komponente, die die Geschwindigkeit des Zählers steuert. Darüber hinaus können sie noch über drei generische Größen konfiguriert werden:

- `data_width`: legt die Bitbreite des `high`, `low` und `y_out` Signals fest. Bei allen Funktionen ist diese auf zwölf eingestellt, da es sich beim DAC-Wandler um einen 12-Bit-Konverter handelt.
- `clk_width`: legt die Bitbreite von `cyc_ticks` und `thresh` fest. Diese muss größer als `data_width` sein. Die `clk_width` legt fest, welche Periodendauer maximal erreicht werden kann, da sie den maximal möglichen Zählerstand begrenzt.
- `clk_ticks_per_count`: Dies ist der Teilungsfaktor zwischen dem angelegten Takt und dem Takt, mit dem der interne Zähler hochzählt.

3.3.1 Konstante

Die konstante Funktion ist die einfachste der vier Funktionskomponenten. Sie gibt lediglich den an ihr anliegenden high-Wert auf ihrem Ausgang aus. Sie verfügt darüber hinaus noch über den Eingang $\overline{\texttt{CE}}$, der bewirkt, dass der Ausgang asynchron auf 0 gesetzt wird.

3.3.2 Rechteck

Die Rechteckfunktion verfügt über einen Eingang `thresh`, der mit dem Stand des internen Zählers verglichen wird. Ist der Zähler größer als `thresh`, so wird der Ausgang auf den Wert `low` gesetzt, ansonsten ist er auf den Wert `high` eingestellt.

3.3.3 Zick-Zack

Die Zick-Zack-Funktion besteht aus einer von `low` zu `high` ansteigenden Treppenfunktion, die anschließend wieder zu `low` abfällt. Sie erreicht dies, indem sie die Zählrichtung (`D`) ihres internen Zählers invertiert, sobald ihr Zählwert `count` die Hälfte von `cyc_ticks` erreicht. Der Zählstand könnte im Prinzip so an den Ausgang ausgegeben werden, allerdings ist die Bitbreite des Funktionswerts geringer als die Bitbreite des Zählstands. Dazu kommt, dass der Zähler von 0 bis `cyc_ticks`/2 zählt, während der Ausgang einen Wertebereich von `low` bis `high` hat. Darum muss der Wert des internen Zählers auf die Breite 12 Bit konvertiert werden. Allgemein lautet die Berechnung dafür:

$$y\_out = \frac{(high - low) \cdot count}{cyc\_ticks/2}$$

Für die Division sorgt der in Abschnitt Abschnitt 3.1.2 beschriebenen Teiler. Dieser muss das Ergebnis Takt für Takt berechnen, da bei der Implementierung fest-

gestellt wurde, dass die standardmäßige Division aus der `unsigned`-Bibliothek zu langsam gewesen wäre.

3.3.4 Rampe

Die Rampenfunktion funktioniert genau so wie die Zick-Zack-Funktion, bis auf dass das Umkehren von `D` wegfällt und der Zähler dafür bis `cyc_ticks` hochzählt und schließlich wieder auf `low` zurückgesetzt wird. Dadurch ergibt sich das typische Sägezahnmuster. Der zusätzliche Eingang `dir` kann noch dazu genutzt werden, die Richtung des Flankenanstiegs umzukehren. `Dir` ist einfach direkt mit dem Eingang `D` des Zählers verbunden und steuert durch die Zählrichtung die Flankensteigung

3.4 Konfigurationsschnittstelle

Die Konfigurationsschnittstelle `CONFIG_INTERFACE` besteht aus einer UART-Schnittstelle, über die der Datenaustausch zwischen Benutzer und Funktionsgenerator erfolgt, sowie der Instruktionsauswertung, die die empfangenen UART-Signale in Konfigurationsbefehle übersetzt.

3.4.1 UART-Schnittstelle

Die UART-Schnittstelle beruht auf dem **U**niversal-**A**synchronous-**R**eceiver-**T**ransmission-Protokoll. Das Protokoll ermöglicht es, byteweise serielle Daten zu verschicken und zu empfangen. Hierfür reichen zwei Drähte aus, die jeweils eins der beiden Signale RX (Receive) und TX (Transmit) transportieren. Zum Start der Kommunikation wird die RX Leitung vom Sender von high auf low gezogen. Der Empfänger detektiert dieses Startsignal und fängt an, die nachfolgenden acht Bits zu einem Byte zusammenzusetzen. Wurde ein Byte übertragen, muss mindestens ein Stop-Bit folgen, bei dem die Receive Leitung des Empfängers auf High liegt. Darauf kann, je nach Implementierung, noch ein Stop-Bit sowie ein Paritätsbit folgen. Da es zwischen dem Sender und Empfänger kein synchrones Taktsignal gibt, ist es wichtig, dass ihre Sende- und Empfangsfrequenz gleich ist. Diese Frequenz ist die sogenannte Baudrate. Im Funktionsgenerator ist sie auf 115200 Bits / s festgelegt.
Die im Generator verwendete Schnittstelle wurde, um den Arbeitsaufwand zu verringern, aus einer Vorlage übernommen (Quelle). Sie beinhaltet sowohl eine Empfänger- als auch eine Sender-Komponente. Es gibt folgende Eingangssignale:

- $\overline{\texttt{reset}}$: Eingang, die Schnittstelle wird auf den Initialisierungszustand zurückgesetzt und die aktuelle Übertragung bzw. aktuelle Empfangsprozesse werden abgebrochen.
- `tx`: Ausgang, Das von der Schnittstelle versendete TX-Signal
- `tx_start`: Eingang, wenn `tx_start` auf high gesetzt wird, wird mit der Übertragung von `data_in` begonnen

Befehlsname	Hex-Code	Argumentbits	Funktion
SETCYCTICKS	0x01	23 - 0	ändern der Zykluszeit der aktuellen Funktion
SETHIGH	0x02	11 - 0	ändern des `high` Werts
SETLOW	0x03	11 - 0	ändern des `low` Werts
SETDUTYCYCLE	0x04	7 - 0	ändern des dutycycles der Rechteckfunktion
SETWVFRM	0x05	1 - 0	ändern der Funktion
SETDIR	0x06	0	ändern der Richtung der Rampenfunktion

Tabelle 3.1: Befehlssatz inklusive Funktion der Befehle.

- `data_in`: Eingang, ein 8-Bit breites Signal, dass das zu versendende Byte enthält.
- `rx`: Eingang, das von der Schnittstelle empfangene RX-Signal
- `data_out`: Ausgang, ein 8-Bit breites Signal, dass das zuletzt von der Schnittstelle empfangene Byte beinhaltet.
- `rx_uart_rdy`: Ausgang, dieses Signal zeigt an, wenn ein komplettes Byte empfangen wurde und bereit ist, gelesen zu werden.

3.4.2 Instruktionsauswertung

Die vom Nutzer gesendeten Bytes werden in einem vier Byte großem Schieberegister gespeichert. Das erste dieser vier Byte repräsentiert das Befehlsbyte, das steuert, welchen Befehl das Interface ausführt. Die folgenden drei Byte sind die Argumente des Befehls. In Tabelle 3.1 sind alle Befehle, ihre Funktion und die Position der Argumente im Speicher beschrieben. Beispielsweise kann der `high`-Wert der Funktionen mit dem Befehl `0x01000FFF` auf das Maximum der Ausgangsspannung gesetzt werden. Dabei ist darauf zu achten, dass für das Argument nur der Wert der Bits Null bis Elf verwendet wird (Bit Null ist das LSB) und die Bits 12 bis 23 ignoriert werden.

3.5 DAC-Konverter

Um den digital berechneten Ausgangswert in ein analoges Signal umzuwandeln, braucht es eine Komponente, die auf dem Basys3-Board noch nicht vorhanden ist. Hier kommt der digital zu analog Wandler PmodDA2 der Firma Digilent zum Einsatz. Dieser DAC verfügt über zwei DAC121S101-ICs von Texas Instruments die als jeweils ein Kanal die seriell empfangenen zwölf Bit Werte in eine analoge Spannung umwandeln (vgl. Datenblatt). Von diesen Kanälen sind zwei im Funktionsgenerator implementiert, es soll aber zunächst nur einer zum Einsatz kommen. Der Chip

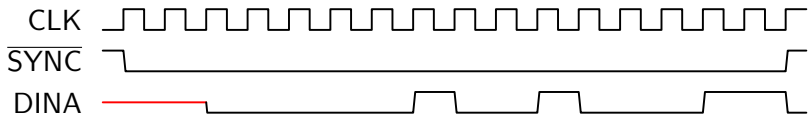


Abbildung 3.3: Signalverlauf bei der Übertragung des Werts 0x123 an den Kanal A des DACs

verfügt über ein Kommunikationsprotokoll ähnlich dem SPI-Protokoll. Es gibt ein $\overline{\texttt{SYNC}}$ Signal, vergleichbar mit dem $\overline{\texttt{SS}}$ Signal von SPI, ein `CLK` Signal und zwei Datenleitungen `DINA` und `DINB` für Kanal Eins und Zwei.

3.5.1 DAC-Kanal

Ein einzelner Kanal kümmert sich um die Serialisierung des anliegenden zwölf Bit Werts `DATA_in`. Ergänzend zu den zwölf Datenbits können dem gesendeten Wort noch zwei Bit für Zusatzfunktionen des Wandlers mitgeschickt werden. Für diese Implementation werden sie allerdings auf `low` gehalten, so dass der DAC im Normalbetrieb läuft. Zusätzlich werden die ersten beiden gesendeten Bits ignoriert, innerhalb einer Übertragung werden also 16 Bit geschickt. Das Senden ist dann erfolgreich, wenn das $\overline{\texttt{SYNC}}$ Signal 16 DAC-Takte lang auf `low` gehalten wurde. Sodann liegt am analogen Ausgang die entsprechende Spannung an.

4 Clocking

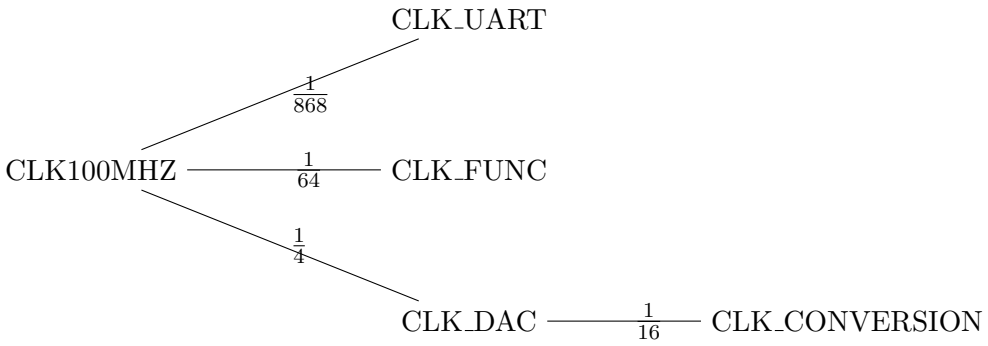


Abbildung 4.1: Clock-Tree