

Fachhochschule Bielefeld
Fachbereich Ingenieurwissenschaften und Mathematik
Studiengang Ingenieurinformatik

Implementierung eines VHDL-basierten, digitalen Funktionsgenerators auf dem Basys3-Board

Art der Ausarbeitung

Markus Hartlage 1103528

10. März 2022

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

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 der Generator zuverlässig in einem Frequenzbereich von 0.09Hz bis ... und einem Spannungsbereich von 0 bis 3,3V funktioniert. Außerdem gelingt die Konfiguration per UART-Schnittstelle.

This student research project describes the implementation of a function generator using the hardware description language VHDL. The structure of the generator is explained and its correct function is checked. In addition, its performance data is recorded and presented. It is shown that the generator works reliably in a frequency range from 0.09Hz to ..., when working on Voltage between 0 and 3,3V. Furthermore, the configuration via UART interface is tested successfully.

Inhaltsverzeichnis

Literaturverzeichnis	5
1 Einleitung	6
2 Konzept	7
2.1 Funktionsmerkmale	7
2.1.1 Funktionen	7
2.1.2 Konfiguration	8
2.2 Aufbau	8
3 Komponenten	10
3.1 Arithmetik	10
3.1.1 Zähler	10
3.1.2 Teiler	11
3.2 Takterzeugung	11
3.2.1 Clock-Enable	11
3.3 Funktionen	12
3.3.1 Konstante	13
3.3.2 Rechteck	13
3.3.3 Zick-Zack	13
3.3.4 Rampe	14
3.4 Konfigurationsschnittstelle	14
3.4.1 UART-Schnittstelle	14
3.4.2 Instruktionsauswertung	15
3.5 DAC-Konverter	15
3.5.1 DAC-Kanal	16
3.6 Clocking	16
4 Funktionstest	18
4.1 theoretische Limitierungen	18
4.1.1 Frequenzbereich	18
4.1.2 Auflösung	19
4.2 reales Verhalten	19
5 Fazit	20
6 Ausblick	21

Literaturverzeichnis

- [1] *DAC121S101/-Q1 12-Bit Micro Power, RRO Digital-to-Analog Converter*. DAC121S101. Texas Instruments. 2021. URL: <https://www.ti.com/lit/ds/symlink/dac121s101.pdf>.
- [2] *Division Algorithm*. Wikipedia. 28. Feb. 2022. URL: https://en.wikipedia.org/wiki/Division_algorithm (besucht am 05.03.2022).
- [3] *PmodDA2TM Reference Manual*. PmodDA2. digilent. 2022. URL: <https://digilent.com/reference/programmable-logic/basys-3/reference-manual>.
- [4] *UART Interface in VHDL for Basys3 Board*. digilent. 2. Nov. 2020. URL: <https://projects.digilentinc.com/alexey-sudbin/uart-interface-in-vhdl-for-basys3-board-eef170> (besucht am 05.03.2022).

1 Einleitung¹

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 Funktion. Z. B. kann ein Funktionsgenerator genutzt werden, um ein Rechteck-Signal mit einer bestimmten Frequenz auszugeben, das dann als Auslöser für eine Kamera fungiert, sodass die Kamera Bilder in einem bestimmten zeitlichen Abstand aufnimmt. Im Normalfall würde ein Funktionsgenerator 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 Konzept

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

2.1 Funktionsmerkmale

In diesem Abschnitt soll geschildert werden, was der Generator kann und wie er sich konfigurieren lässt.

2.1.1 Funktionen

Der Generator kann auf vier verschiedene Funktionsbausteine zurückgreifen, die jeweils einen anderen Spannungsverlauf ausgeben:

1. **Konstante Funktion**

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. Die Dauer des **high**-Pegels bzw. **low**-Pegels, T_h und T_l berechnen sich folgendermaßen:

$$T_l = T \cdot \frac{\text{dutycycle}}{255}, \text{dutycycle} \in \{0, 1, \dots, 255\} \quad (2.1)$$

$$T_h = T - T_l \quad (2.2)$$

Auf diese Art lassen sich 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.

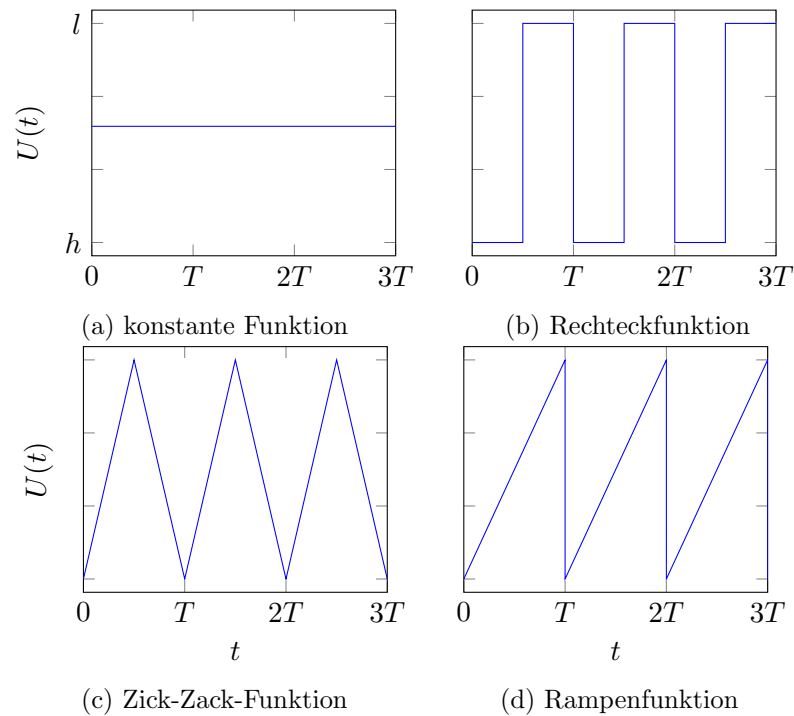


Abbildung 2.1: Funktionen, die der Funktionsgenerator ausgeben kann. Ihr Verlauf ist als Spannung $U(t)$ über der Zeit t aufgetragen. Die Zykluszeit und **high** bzw. **low** sind durch T , h und l gekennzeichnet.

2.1.2 Konfiguration

Der Funktionsgenerator muss, um extern konfiguriert werden zu können, über eine Möglichkeit für Nutzereingaben verfügen. Zu diesem Zweck wird die UART-Schnittstelle auf dem Basys3-Board genutzt. Über diese kann der Nutzer Konfigurationsbefehle per Computer versenden, die dann vom Generator interpretiert werden. Auch nutzerseitig wurde ein Python-Script geschrieben, mit dem die Kommunikation mit dem Generator vereinfacht wird.

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. Dieser erzeugt daraus wiederum ein analoges Signal, dass dann am Ausgang anliegt.

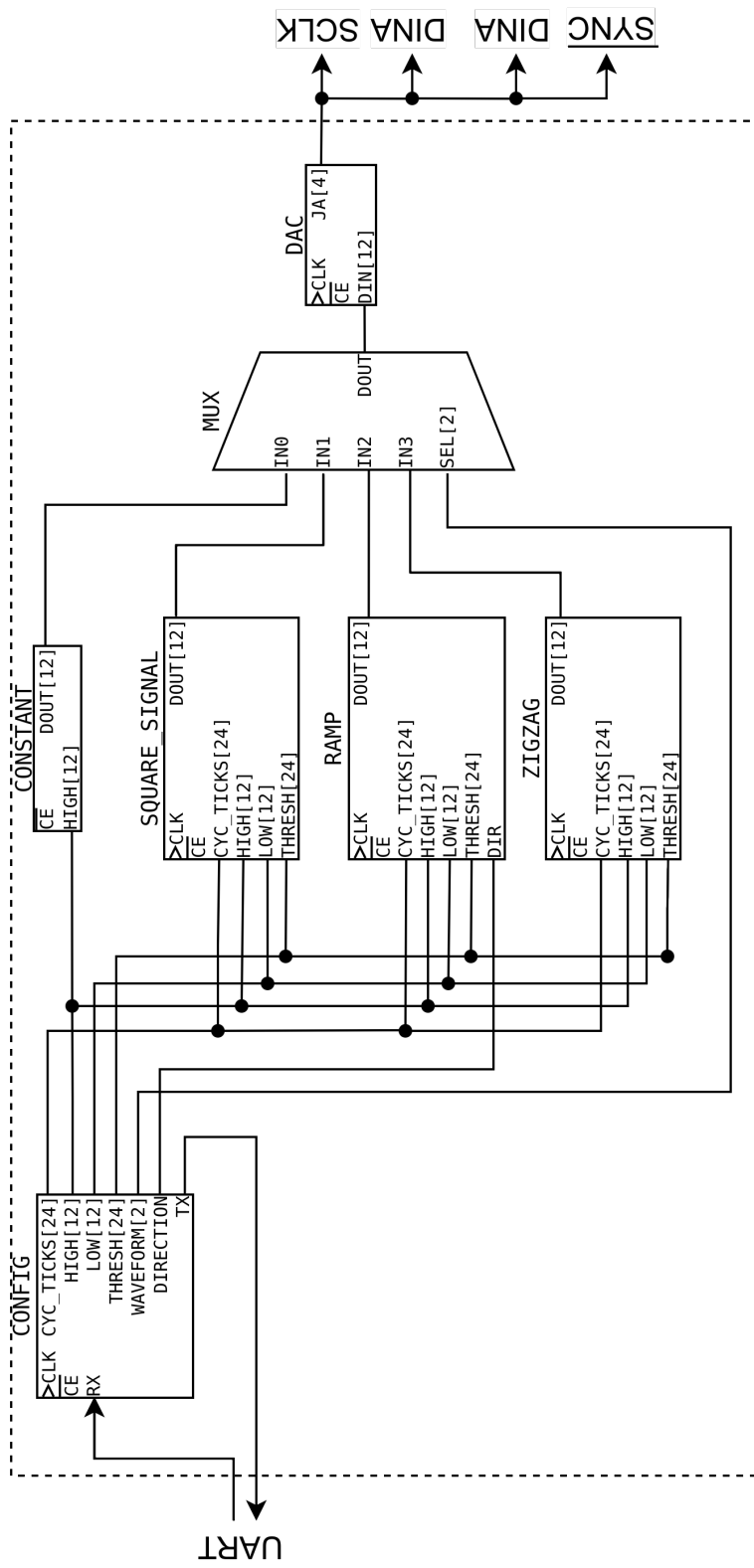


Abbildung 2.2: Diagramm des Funktionsgenerators, zusammengesetzt aus seinen Einzelkomponenten. Aus Gründen der Übersichtlichkeit sind die CLK Signale und CE Signale nicht angeschlossen dargestellt. Alle CLK Signale sind an den Systemtakt angeschlossen und alle CE Signale liegen auf Masse.

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. Vorab muss noch erwähnt werden, dass jede Komponente über ein `CLK` und ein `CE` Signal verfügen. `CE` ist der Chip-Enable Eingang. `CLK` ist der Eingang für das Taktsignal. Alle Komponenten führen einen Arbeitsschritt aus, wenn an `CE` `low` anliegt und `CLK` eine steigende Flanke detektiert.

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 $o_count = inc \cdot N \cdot (-1)^{1-D}$, $N \in \{0, 1, 2, \dots, max_ticks\}$ 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: `low` heißt aufwärts, `high` 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.

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 \bar{R} auf `low` 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 $Z(n-1)$ 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.

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 \bar{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 \bar{CE} `low` ist. Das Timingdiagramm in Abbildung Abb. 3.2 verdeutlicht, auf welche Taktzyklen die angeschlossene Komponente reagiert. Der große Vorteil dieser Methode liegt darin,

```

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 (vgl. [2]).

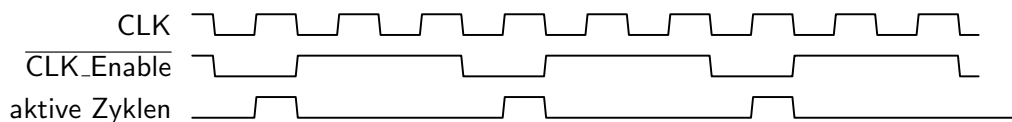


Abbildung 3.2: Beispielhaftes Timing-Diagramm. Jeder vierte Taktzyklus wird durch `CLK_Enable` aktiviert.

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 Komponenten-ausgangs 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 $\overline{\text{CE}}$, der bewirkt, dass der Ausgang asynchron auf **low** 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 3.1.2 beschriebene Teiler. Dieser muss das Ergebnis Takt für Takt berechnen, da bei der Implementierung festgestellt 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 Instruktionauswertung, die die empfangenen UART-Signale in Konfigurationsbefehle übersetzt.

3.4.1 UART-Schnittstelle

Die UART-Schnittstelle beruht auf dem **U**niversal-**A**synchronous-**R**ceiver-**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 (vgl. [4]). Sie beinhaltet sowohl eine Empfänger- als auch eine Sender-Komponente. Es gibt folgende Eingangssignale:

- **reset**: Eingang, die Schnittstelle wird auf den Initialisierungszustand zurückgesetzt und die aktuelle Übertragung bzw. aktuelle Empfangsprozesse werden abgebrochen.

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.

- **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
- **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 Basys 3 Board noch nicht vorhanden

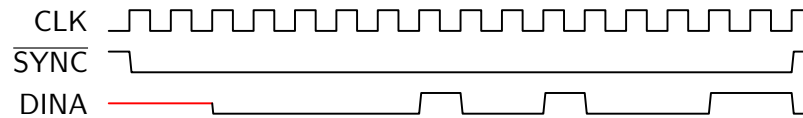


Abbildung 3.3: Signalverlauf bei der Übertragung des Werts 0x123 an den Kanal A des DACs

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. [1], [3]). Von diesen Kanälen sind zwei im Funktionsgenerator implementiert, es soll aber zunächst nur einer zum Einsatz kommen. Der Chip verfügt über ein Kommunikationsprotokoll ähnlich dem SPI-Protokoll. Es gibt ein $\overline{\text{SYNC}}$ Signal, vergleichbar mit dem $\overline{\text{SS}}$ 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 $\overline{\text{SYNC}}$ Signal 16 DAC-Takte lang auf `low` gehalten wurde. Sodann liegt am analogen Ausgang die entsprechende Spannung an.

3.6 Clocking

Wichtig für den richtigen Ablauf ist die Taktung der Komponenten untereinander. Dazu dienen die $\overline{\text{CE}}$ und CLK Eingänge der einzelnen Komponenten. Die Komponente Clock-Enable oder andere interne Prozesse schalten das $\overline{\text{CE}}$ Signal in dem erforderlichen Takt ein bzw. aus. Durch das Hintereinanderschalten der Komponenten ergibt sich ein Baum aus von einander abhängigen Takten. Beispielsweise ist die Frequenz der Funktionskomponenten von der Frequenz des DAC-Konverters abhängig. Der sogenannte Clock-Tree des Generators ist in Abb. 3.4 abgebildet. Vom Taktsignal des Systemtakts CLK100MHZ leiten sich folgende Signale ab:

- CLK_UARTx16: Dieses Taktsignal besitzt mit $115200 \frac{\text{Bit}}{\text{s}} \cdot 16 = 1,84 \frac{\text{MBit}}{\text{s}}$ die 16 Fache Frequenz der Baudrate, dies entspricht einem 54tel des Systemtakts. Es dient dazu, das eingehende Signal so fein abzutasten, dass rechtzeitig auf eine Änderung in der RX-Leitung reagiert werden kann.
 - CLK_UART: Dies ist der Takt, mit dem Bits über die UART-Schnittstelle

versendet und empfangen werden. Mit $115200 \frac{Bit}{s}$ entspricht er einem 16tel des CLK_UARTx16 Takts.

- **CLK_DAC:** Der Takt des DAC-Konverters ist laut Datenblatt auf maximal 30MHz beschränkt [1]. Da eine ganzzahlige Teilung des Systemtakts erforderlich ist, ist die maximale Frequenz, mit der der DAC am Basys 3 Board betrieben werden kann $\frac{f_{sys}}{4} = \frac{100MHz}{4} = 25MHz$.
 - **CLK_CONVERSION:** Eine Konvertierung von digital zu analog dauert 16 Takte, wie bereits in Abschnitt 3.5.1 beschrieben wurde. Damit beträgt der Takt, mit dem Konvertierungen erfolgen, ein 16tel des CLK_DAC Takts betragen.
 - **CLK_FUNCCOUNT:** Der interne Takt der Zähler in den Funktionsbausteinen läuft im selben Takt wie CLK_CONVERSION, sodass ein neuer Ausgangswert aus dem Zählstand errechnet und dieser dann konvertiert werden kann.

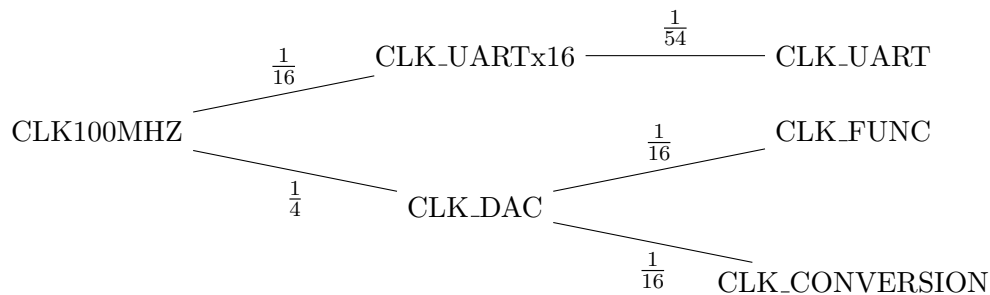


Abbildung 3.4: Clock-Tree des Funktionsgenerators.

4 Funktionstest

Nach der Implementierung des zuvor geschilderten Konzepts wurde ein Funktionstest durchgeführt. Dazu werden zunächst die theoretischen Grenzen der Konfiguration getestet und danach die tatsächlichen Ausgabewerte mit den theoretischen verglichen. Besonders interessant ist hierbei, in welchem Frequenzbereich der Generator zuverlässig funktioniert.

4.1 theoretische Limitierungen

Um den praktischen Nutzen des Funktionsgenerators einzuschätzen, werden die Grenzen der einstellbaren Frequenz, die Auflösung und der Spannungsbereich berechnet. Für die Spannung ergeben sich diese Werte aus dem Datenblatt des eingesetzten DACs, dieser kann in einem Bereich von 0 bis 5V eingesetzt werden [3]. Für den Betrieb auf dem Basys 3 Board läuft der Spannungsbereich von 0 bis 3,3 V, da hier die Ausgangsspannung des Boards der limitierende Faktor ist. Nachfolgend werden noch der Frequenzbereich und die Auflösung bestimmt.

4.1.1 Frequenzbereich

Der Systemtakt f_{sys} des Generators beträgt 100 MHz. Der Frequenzbereich der Funktionen, die er ausgeben kann, reicht von ca. 0,0931 Hz bis zu ca. 781 kHz. Die Frequenz f_{count} ist die Frequenz mit der der interne Zähler der Funktionsbausteine hochzählt und beträgt ein 64tel des Systemtakts f_{sys} (siehe Gleichung (4.1)). Die minimale Frequenz f_{min} errechnet sich aus dem maximalen Zählstand, der wiederum abhängig ist von seiner Bitbreite clk_width (siehe Gleichung (4.3)). Die maximale Frequenz ergibt sich aus dem Shannon'schen Abtasttheorem, nach dem die minimale Abtastfrequenz eines Signals doppelt so groß sein muss, wie die Frequenz des abgetasteten Signals [(|blabla). In diesem Fall entspricht die Abtastfrequenz f_{count} und das abgetastete Signal dem Ausgangssignal, woraus folgt, dass das ausgehende Signal nur halb so groß sein kann, wie f_{count} (siehe Gleichung (4.3)).

$$f_{count} = \frac{f_{sys}}{64} \quad (4.1)$$

$$f_{max} = \frac{f_{count}}{2} \quad (4.2)$$

$$f_{min} = \frac{f_{count}}{2^{clk_width} - 1} \quad (4.3)$$

4.1.2 Auflösung

Die Auflösung des analogen Ausgangssignals hängt sowohl von der Geschwindigkeit ab, mit der das digitale Signal analogisiert werden kann, als auch von der maximalen Anzahl digital darstellbarer Werte. Überschreitet die Frequenz des analogen Signals f die Grenzfrequenz f_{grenz} , so fällt die Auflösung R reziprok zu f ab (siehe Abb. 4.1). Oberhalb von f_{grenz} ist die Auflösung durch die Bitbreite des ADCs begrenzt. Da es sich um einen 12-Bit Wandler handelt, beträgt die Anzahl darstellbarer Werte und damit auch die höchste Auflösung $2^{12} = 4096$. Diesen Wert nimmt die Auflösung an, wenn die Frequenz kleiner als f_{grenz} ist (siehe Gleichung (4.4)).

$$R = \begin{cases} 4095 & f \leq f_{grenz} \\ \frac{f_{count}}{f} & f > f_{grenz} = \frac{f_{count}}{4095} \end{cases} \quad (4.4)$$

Konkret bedeutet dass für den Funktionsgenerator, dass bei einer Amplitude von $U_{SS} = 3,3V$ und einer Frequenz von $f = 100kHz$, die Auflösung $4,54V^{-1}$ statt der maximalen Auflösung von $1241V^{-1}$ beträgt, das heißt, dass bei 100kHz 4,54 Werte statt 1241 Werte pro Volt gesampelt werden.

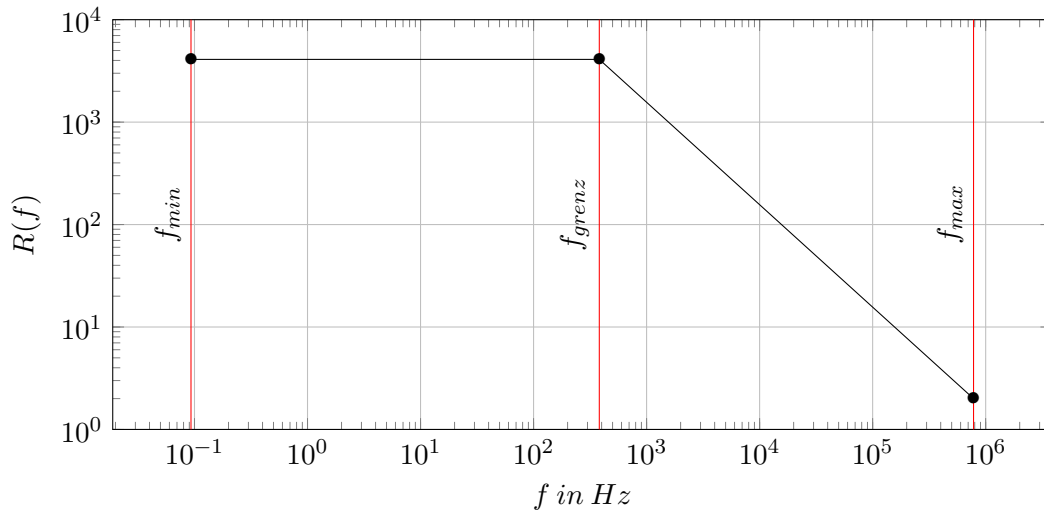


Abbildung 4.1: Doppelt logarithmisches Diagramm der Auflösung $R(f)$ über den Frequenzbereich f . Die Auflösung bleibt konstant bei 4095 1/tick bis sie schließlich bei f_{grenz} anfängt zu sinken.

4.2 reales Verhalten

Nun soll das reale Verhalten des Funktionsgenerators untersucht werden. Dazu wird das digitale Oszilloskop Analog Discovery 2 von digilent benutzt.

5 Fazit

Es wurde erfolgreich ein digitaler Funktionsgenerator auf dem Basys 3 Board implementiert. Dieser kann drei verschiedene Funktionen ausgeben und über eine UART Schnittstelle konfiguriert werden. Darüber hinaus konnte seine Funktionstüchtigkeit in dem Frequenzbereich ... bewiesen werden. Außerdem wurde gezeigt, dass die Konfiguration per UART-Interface gelingt.

6 Ausblick

Durch die Implementation der Funktionskomponenten als Bausteine ist es relativ leicht, die möglichen Funktionsformen zu erweitern. Z.B. könnte noch eine Sinusfunktion hinzugefügt werden. Theoretisch könnte auch ein nicht-periodisches Signal, wie ein zeitversetzter Sprung hinzugefügt werden.

Außerdem kann die UART-Schnittstelle noch um weitere Features ergänzt werden. Es wäre beispielsweise denkbar, die gegenwärtigen Funktionsparameter über den Transmitter an den User zurück zu schicken, sodass dieser sie leichter kontrollieren kann. Dazu müsste auch die Konfigurationsschnittstelle um Befehle zur Rückgabe der Funktionsparameter erweitert werden.

Darüber hinaus könnte auch noch der zweite Kanal auf dem DAC-Konverter genutzt werden um zwei verschiedene Signale gleichzeitig auszugeben.

Abschließend fehlt auch noch der Test in einer realen Anwendung, z.B. einem Trigger für eine Fotokamera, die in einer bestimmten Frequenz Bilder aufnehmen soll.