



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

1. März 2022

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

Eine kurze deutsche Zusammenfassung.

A short abstract in english.

Inhaltsverzeichnis

1	Einleitung	5
1.1	was kann das gerät	6
2	Komponenten	8
2.1	Arithmetik	8
2.1.1	Zähler	8
2.1.2	Teiler	9
2.2	Takterzeugung	10
2.2.1	Clock-Enable	10
2.3	Funktionen	11
2.3.1	Konstante	11
2.3.2	Rechteck	11
2.3.3	Zick-Zack	12
2.3.4	Rampe	12
2.4	Konfigurationsschnittstelle	12
2.4.1	UART-Schnittstelle	12
2.4.2	Instruktionsauswertung	13
2.5	DAC-Konverter	14
2.5.1	DAC-Kanal	14

1 Einleitung¹

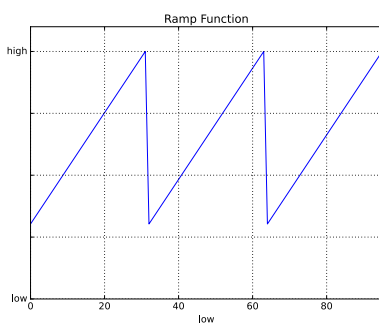
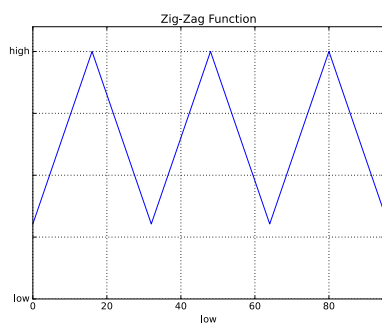
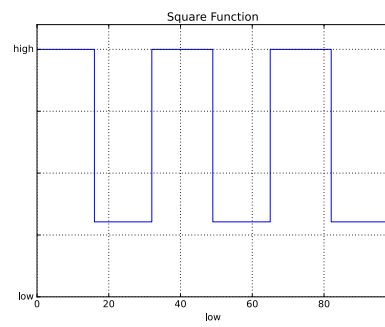
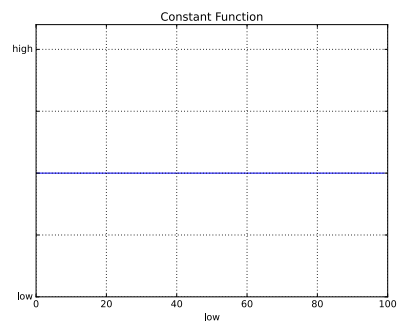
Diese Studienarbeit behandelt die Konzipierung und Implementierung eines digitalen Funktionsgenerators in der Hardwarebeschreibungssprache VHDL. Ein Funktionsgenerator ist ein elektronisches Bauteil, das in der Lage ist, verschiedene Spannungsverläufe an seinem Ausgang auszugeben. Diese Spannungsverläufe 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. 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.

Im folgenden soll das Konzept des Funktionsgenerators erläutert werden und seine Funktionsweise erklärt werden.

1.1 was kann das gerät

der generator kann 4 verschiedene Funktionen ausgeben,

1. Konstante Ein konstanter Wert liegt am Ausgang an.
2. Rechteck-Funktion Der Wert wechselt zwischen High- und Low-Pegel in der Frequenz f . Der Anteil der Zykluszeit T , in dem der Ausgang auf High ist, wird über den dutycycle eingestellt.
3. Zick-Zack-Funktion Der Analogwert steigt vom Low-Pegel bis zum High-Pegel linear an, erreicht er den High-Pegel, fällt der Analogwert wieder kontinuierlich auf Low ab. Somit schwankt der Pegel 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 Low zurück. Alternativ kann die Rampe auch vom High-Pegel her abfallen und bei Erreichen von Low wieder auf High zurück springen.



2 Komponenten

In diesem Kapitel soll der Aufbau des Funktionsgenerators anhand seiner digital-technischen Komponenten erläutert werden. Ihre Verschaltung im Generator ist in Abb. 2.1 zu sehen. Hier kann man gut erkennen, dass die verschiedenen Funktionen parallel arbeiten und von der Konfigurationsschnittstelle gesteuert werden. Diese sorgt auch dafür, dass die gewünschte Funktionskomponente auf den Ausgang durchgeschaltet wird.

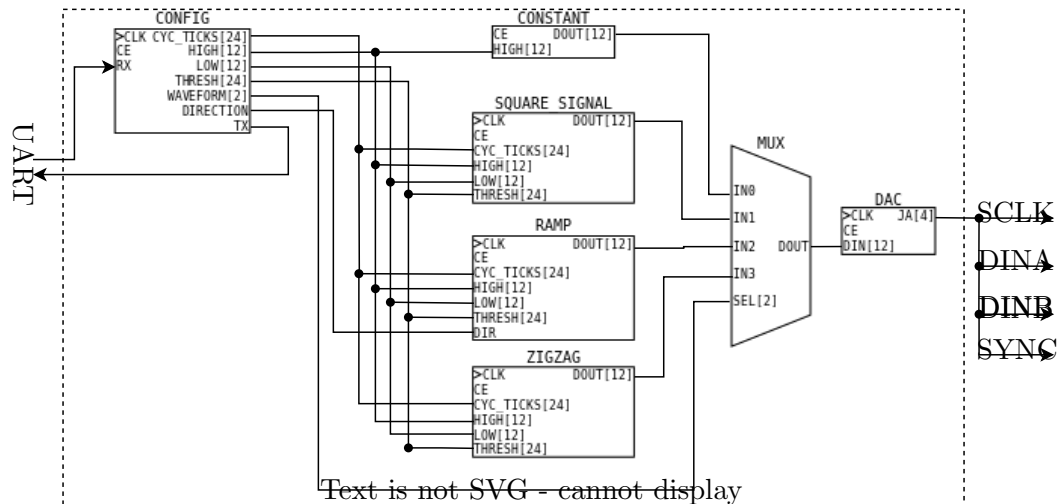


Abbildung 2.1: Diagramm des Funktionsgenerators, zusammengesetzt aus seinen Einzelkomponenten

2.1 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.

2.1.1 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: 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. 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 0 gesetzt wird.

2.1.2 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 2.3.3 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, ermittelt dabei pro Rechenschritt (d.h. pro Takt) eine Stelle des Ergebnisses. Er geht den Zähler von der größten bis zur kleinsten Stelle durch und schiebt die 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, ist Q an dieser Stelle

Eins, ansonsten ist Q Null. Der nachfolgende Pseudocode verdeutlicht den Algorithmus noch einmal. 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.

```
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 die Stelle des Zählers:
         $R(0) = Z(i)$ 
        wenn  $R \geq N$ , dann
            setze  $R = R - N$ 
            setze  $Q(i) = 1$  // die Stelle des Quotienten wird 1
        Ende der bedingten Anweisung
    springe zum Anfang der Schleife
Ende der bedingten Anweisung
Ende des Algorithmus
```

2.2 Takterzeugung

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

2.2.1 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=0` anliegt. Das Timingdiagramm in Abbildung Abb. 2.2 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.

2.3 Funktionen

Die Herzstücke des Funktionsgenerators bilden seine Funktionskomponenten. An ihren Ausgängen liegen die von ihnen berechneten Werte an, von denen einer zum

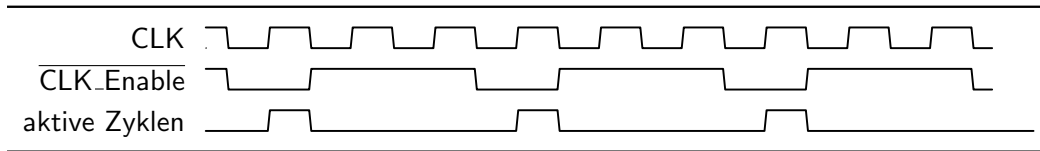


Abbildung 2.2: Beispielhaftes Timing-Diagramm. Jeder vierte Taktzyklus wird durch `CLK.Enable` aktiviert.

in Abschnitt 2.5 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.

2.3.1 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 `CE`, der bewirkt, dass der Ausgang asynchron auf 0 gesetzt wird.

2.3.2 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.

2.3.3 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 2.1.2 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.

2.3.4 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.

2.4 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.

2.4.1 UART-Schnittstelle

Die UART-Schnittstelle beruht auf dem **Universal-Asynchronous-Receiver-Transmission**-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:

1. **CLK**: Eingang, gibt die Taktfrequenz der Komponente vor
2. **CE**: Eingang, "chip-enable"-Signal, aktiviert die Komponente wenn es auf low gesetzt wird
3. **reset**: Eingang, die Schnittstelle wird auf den Initialisierungszustand zurückgesetzt und die aktuelle Übertragung bzw. aktuelle Empfangsprozesse werden abgebrochen.
4. **tx**: Ausgang, Das von der Schnittstelle versendete TX-Signal
5. **tx_start**: Eingang, wenn **tx_start** auf high gesetzt wird, wird mit der Übertragung von **data_in** begonnen
6. **data_in**: Eingang, ein 8-Bit breites Signal, dass das zu versendende Byte enthält.
7. **rx**: Eingang, das von der Schnittstelle empfangene RX-Signal
8. **data_out**: Ausgang, ein 8-Bit breites Signal, dass das zuletzt von der Schnittstelle empfangene Byte beinhaltet.
9. **rx_uart_rdy**: Ausgang, dieses Signal zeigt an, wenn ein komplettes Byte empfangen wurde und bereit ist, gelesen zu werden.

2.4.2 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 2.1 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

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 2.1: Befehlssatz inklusive Funktion der Befehle.

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.

2.5 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 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.

2.5.1 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.

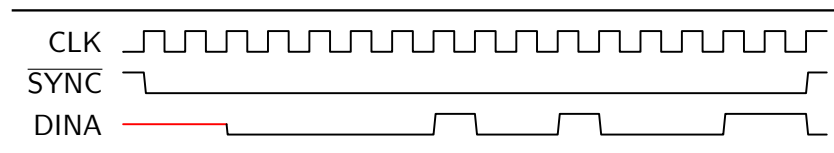


Abbildung 2.3: Signalverlauf bei der Übertragung des Werts 0x123 an den Kanal A des DACs