FPGA-Design und VHDL Grundkurs

Jan Grapengeter

May 2, 2019

Abstract

In dieser Arbeit sollen die Grundlagen von VHDL und FPGA-Design anhand praktischer Übungen erarbeitet werden.

Inhaltsangabe

Bildverzeichnis

Tabellenverzeichnis

1 Vorwort

In dieser Arbeit sollen Übungsprojekte zum Erlernen von VHDL und FPGA-Design präsentiert werden. Es wird grundsätzliches Wissen um Logikgatter und Flip-Flops vorausgesetzt. Alle Projekte sollen mit dem Basys3-board von Digilent und dem Vivado-Design-Tool von Xilinx erarbeitet werden können. Die Projekte sollen so aufgebaut sein, dass keine externen Quellen oder Peripherie zur Bearbeitung der Projekte benötigt werden.

2 Übungen

2.1 Erstes Projekt

2.1.1 Anlegen eines Projektes in Vivado

-Starten Sie Vivado
-Wählen Sie "Create New Project"
-"Next"
-Project Name: Wählen Sie einen Namen für das Projekt und legen Sie einen Installationsordner für das Projekt auf Ihrer Festplatte an.
-"Next"
-Project Type: Wählen Sie "RTL Project" (voreingestellt), "Do not specify sources at this time" sollte nicht angewählt sein.
-"Next"
-Add Sources: Hier legen Sie ihre VHDL-Dateien an
- Klicken Sie auf "Create File"
-Wählen Sie VHDL als Dateityp (File type) und wählen Sie "main" als Namen der Datei, "File location" sollte auf "Local to Project" stehen (voreingestellt).
-"Next"
-"Add Existing IP" kann ignoriert werden. -"Next"
-Add Constraints: Hier legen Sie ein Constraints-File an, dessen Bedeutung wird später erklärt
-Wählen Sie "Create File"
-Wählen Sie XDC als Dateityp (File type) und wählen Sie "cons" als Namen der Datei, "File location" sollte auf "Local to Project" stehen (voreingestellt)
-"Next"
-Default Part: Hier wählen Sie den von Ihnen verwendeten FPGA-Chip aus
-Wählen Sie "xc7atcpg236-1" aus, dies ist die Bezeichnung des FPGA-Chips auf dem Basys3.
-New Project Summary: Hier wird eine Zusammenfassung des gerade angelegten Projektes angezeigt. Wenn Sie alles richtig gemacht haben, sollte diese aussehen wie folgt:

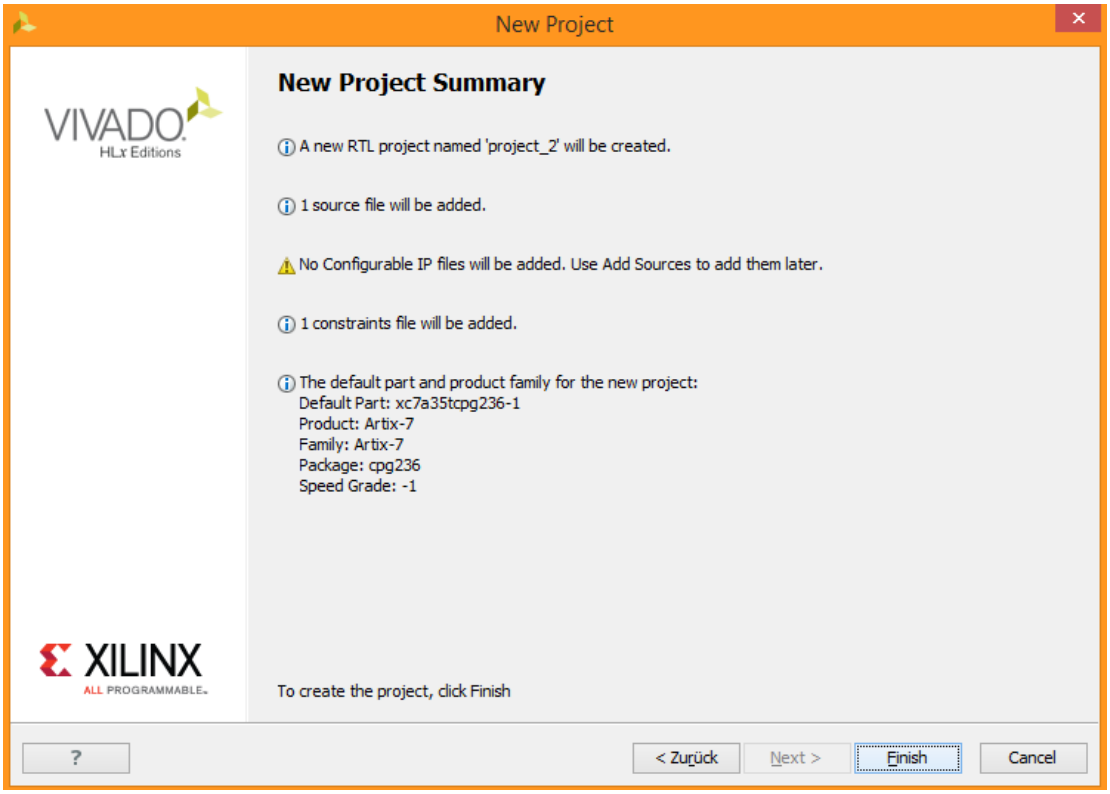

Figure 1: Projektzusammenfassung

-"Finish"

-Define Module: Hier tragen Sie die Ein- und Ausgänge ihres Projektes ein. Über diese Ports werden später Signale in das FPGA hinein und hinausgeführt.
-Wählen Sie "main" als "Entity name" und "Behavarioal" als "Archictecture Name" (voreingestellt).
-Unter "Port Name" tragen Sie "sw0" als Namen ein
-"Direction" sollte auf "in" stehen und "Bus" nicht aktiviert sein (voreingestellt).
-Klicken Sie auf das grüne Kreuz, um einen weiteren Port hinzuzufügen, tragen Sie "ld0" als Namen ein und ändern Sie "Direction auf "out", Bus sollte nicht aktiviert sein.
-"Ok"
-Wenn Sie alles richtig gemacht haben, sollten Sie nun im Projekt Manager sein und folgendes Bild sehen:

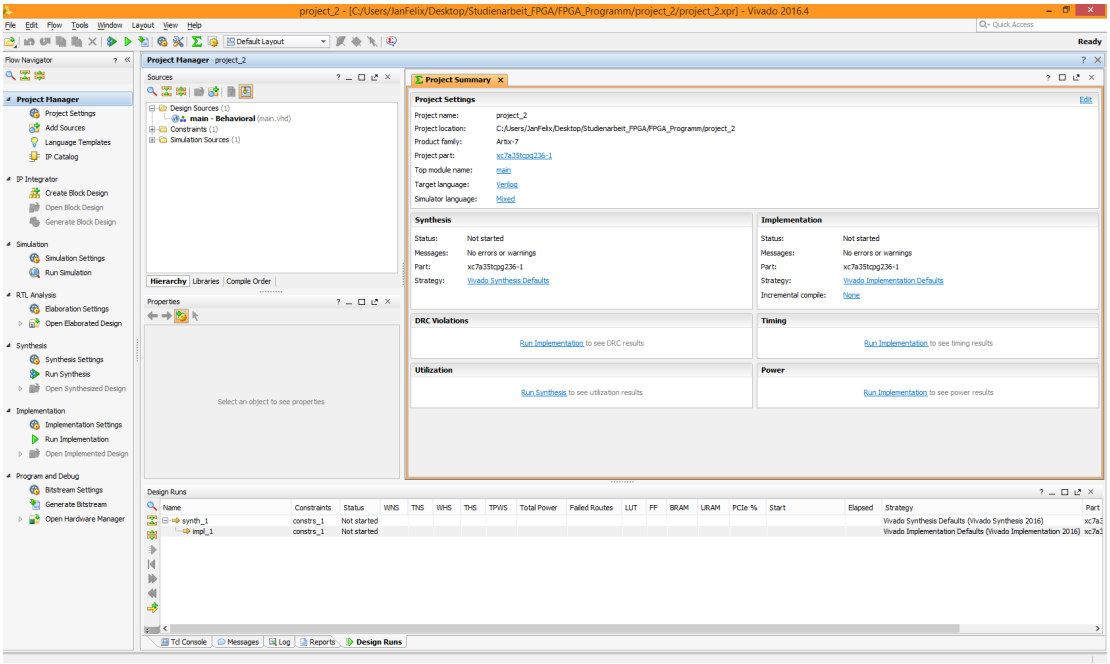

Figure 2: Projektmanager

-Damit haben Sie das Projekt erfolgreich angelegt.

2.1.2 Anlegen der VHDL-Main

-Doppelklicken Sie auf "main - Behavioral(main.vhd)" unter "Sources" (sollte orange umrandet sein)
-Im nun geöffneten Fenster "main.vhd" sollten Sie Ihre Main-Datei mit den beiden von Ihnen angelegten Ports sehen.

```
22  library IEEE;
23  use IEEE.STD_LOGIC_1164.ALL;
24
25  -- Uncomment the following library declaration if using
26  -- arithmetic functions with Signed or Unsigned values
27  --use IEEE.NUMERIC_STD.ALL;
28
29  -- Uncomment the following library declaration if instantiating
30  -- any Xilinx leaf cells in this code.
31  --library UNISIM;
32  --use UNISIM.VComponents.all;
33
34  entity main is
35      Port ( sw0 : in STD_LOGIC;
36             ld0 : out STD_LOGIC);
37  end main;
38
39  architecture Behavioral of main is
40
41  begin
42
43
44  end Behavioral;
```

Figure 3: VHDL main

-Unter "begin" tragen Sie:
ld0<=sw0;
ein. Damit wird der Wert des Eingangs "sw0" auf den Ausgang "ld0" geschrieben.
-Speichern Sie "main.vhd" ab.

2.1.3 Anlegend der Constraints-Datei

-Wieder im Reiter "Project Manager" doppelklicken Sie auf "Constraints"
-Doppelklicken Sie dann auf "cons.xdc", dies ist Ihre vorher angelegte Constraints-Datei.
-Sie sollten nun das noch leere Fenster "cons.xdc" sehen.
-Tragen Sie hier folgende Zeilen ein:

```
set_property -dict { PACKAGE_PIN V17   IOSTANDARD LVCMOS33 } [get_ports { sw0}];
set_property -dict { PACKAGE_PIN U16   IOSTANDARD LVCMOS33 } [get_ports { ld0}];
```

-Damit weisen Sie Ihren in der VHDL-Datei festgelegten Ein- und Ausgängen physikalische Ports auf dem FPGA-Chip zu. Hier wird dem Eingang "sw0" der Port "v17" zugewiesen und dem Ausgang "sd0" der Port "u16". Auf dem Basys3 sind die Ein- und Ausgänge des FPGA mit bestimmter Peripherie auf dem Board fest verdrahtet. Sie sollten, wenn Sie auf das Board schauen, auch die Namen der Peripherie und der angeschlossenen Ports finden. "sw0" mit Port

"v17" und "ld0" mit Port "u16" sind im rechten unteren Eck. (Im Bild bei Nummer 6)

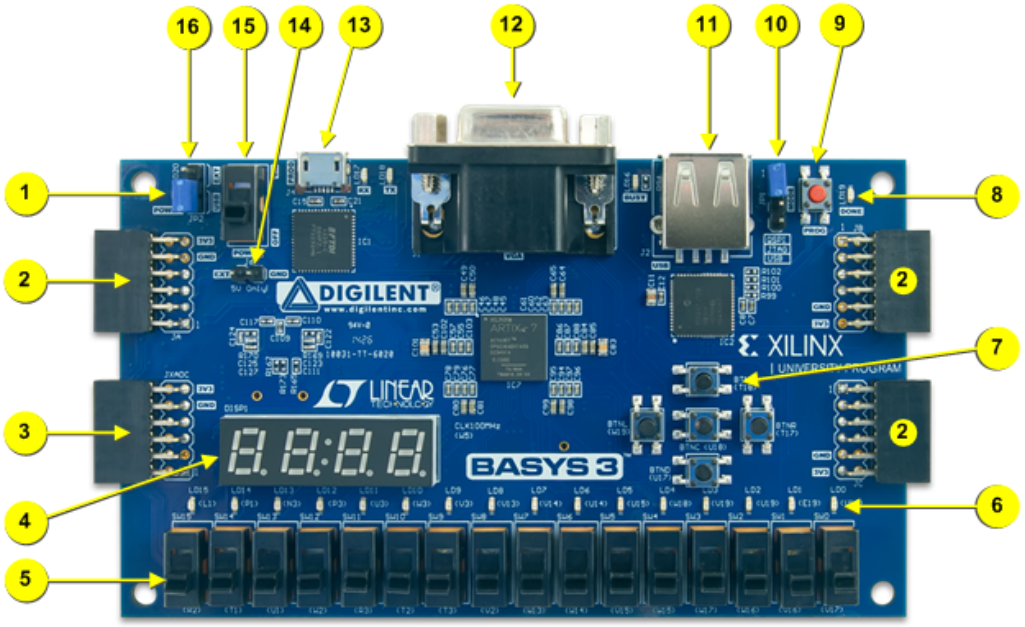

Figure 1. Basys3 board features

Callout	Component Description	Callout	Component Description
1	Power good LED	9	FPGA configuration reset button
2	Pmod connector(s)	10	Programming mode jumper
3	Analog signal Pmod connector (XADC)	11	USB host connector
4	Four digit 7-segment display	12	VGA connector
5	Slide switches (16)	13	Shared UART/ JTAG USB port
6	LEDs (16)	14	External power connector
7	Pushbuttons (5)	15	Power Switch
8	FPGA programming done LED	16	Power Select Jumper

Figure 4: Basys3 Frontansicht

2.1.4 Programmierung des Boards

-Um nun Ihr Projekt auf das Board zu überspielen, klicken Sie in der rechten Spalte (Flow Navigator) auf "Generate Bitstream", damit wird aus Ihrem Projekt ein für das FPGA lesbarer Code gemacht.
-No Implementation Results available: "Yes"
-Launch runs:
-Wählen Sie "Default directory" als "Launch directory"
-Wählen Sie "Launch runs on local host". Unter "Number of Jobs" können Sie die Anzahl der Prozessorkerne auswählen, die Vivado dafür einsetzen darf. Wählen Sie soviele wie möglich, da dieser Vorgang ansonsten einige Minuten dauern kann, insbesondere bei größeren Projekten.
-In der rechten oberen Ecke können Sie die einzelnen Schritte sehen, die Vivado zur Bitstream-

erstellung durchläuft.
-Bitstream Generation Completed:
-Wählen Sie "Open Hardware Manager"
-Unter "Flow Navigator" finden Sie nun ganz unten unter "Program and Debug" die Schaltfläche "Open Target", klicken Sie auf diese und wählen Sie "auto connect"
-Vivado sollte nun automatisch das angeschlossene Basys3 board finden.
-Wo vorher Ihr Reiter "Project Mananger" war, sollte nun der "Hardware Manager" geöffnet sein.
-Hier können Sie das FPGA Chip-Modell sehen.
-Unter "Flow Navigator - Program and Debug" wählen Sie nun "Program Device" und klicken Sie auf "xc7a35t_0" (Der Basys3 FPGA Chip).
-Program Device: Hier wählen Sie den von Ihnen generierten Bitstream aus.
-Da dieser automatisch ausgewählt sein sollte, klicken Sie auf "Program".
-Das FPGA sollte nun mit Ihrem VHDL Code programmiert worden sein.
-Testen Sie dies, indem Sie den Schalter 1 (sw0) umschalten, die LED0 (ld0) sollte an und ausgehen, abhängig von der Schalterposition.

2.1.5 Aufgaben

1.: Ändern Sie die Constraints so, dass nun Schalter "sw1" die Led "ld1" umschaltet. Nehmen Sie keine Änderungen an der VHDL Datei vor.
2.: Legen Sie in Ihrer Main zwei zusätzliche Ports an und verbinden Sie diese wie "sw0" und "ld0". Ändern Sie die constraints Datei danach so, dass Sie über den Knopf "BTND" (zu finden direkt oberhalb der LEDs) die LED 5 (ld5) umschalten können.
3.: Drücken Sie den Knopf "PROG" (zu finden rechts oben) auf Ihrem Board, dieser dient in der Standard-Stellung als Reset. Welches Programm läuft nun auf Ihrem FPGA? Was sagt Ihnen das über die interne Speicherstruktur eines FPGAs?

2.2 Logikgatter in VHDL und FPGAs

2.2.1 Einfache Gatter

Einfache Logikgatter lassen sich in VHDL wie folgt anlegen:

```
ld0<=sw0 and sw1;
```

Damit wird der Ausgang ld0 auf "1" geschaltet, wenn die Eingänge sw0 und sw1 beide auf "1" stehen. Wie in der letzten Übung müssen ld0, sw0 und sw1 mit entsprechender Peripherie auf dem Board verbunden werden. Die weiteren Logikgatter haben folgende Syntax:

Logikgatter	
Und-Gatter	and
Oder-Gatter	or
Nicht-Gatter	not
NAND-Gatter	nand
NOR-Gatter	nor
XOR-Gatter	xor
XNOR-Gatter	xnor

Table 1: Logikgatter

Aufgaben
1.: Legen Sie ein neues Projekt an und nennen Sie dieses Gatterlogik. Legen Sie sw0 und sw1 als Eingänge und ld0 als Ausgang fest. Verbinden Sie in der Constraints Datei sw0 mit Schalter 1 (sw0), sw1 mit Schalter 1(sw1) und ld0 mit LED 0 (ld0). LED 0 soll leuchten, wenn beide Schalter betätigt werden.
2.: Verbinden Sie je zwei benachbarte Schalter mit einem Logikgatter und weisen Sie den Ausgang des Gatters einer benachbarten LED zu.
3.: Klicken Sie unter "Flow Navigator" auf "Open Elaborated Design" und schauen Sie, welche Logikgatter Vivado aus ihrem VHDL Code gemacht hat.
4.: Klicken Sie unter "Flow Navigator" auf Synthesis/Schematic und sehen Sie sich an, wie die Ihre Schaltung nun aussieht. Was fällt Ihnen auf im Vergleich zum Elaborated Design?

2.2.2 Look Up Table

In einem FPGA sind keine Logikgatter fest verbaut. Alle Logikgatter werden über Look Up Tables (LUTs) realisiert. Wenn Sie in VHDL ein Logikgatter festlegen, wird dafür ein "Configurable Logic Block" (CLB) im FPGA angelegt, dessen Werte dem LUT des Gatters entsprechen. In dem Artix 7 A35T chip, der auf dem Basys3 verbaut ist, kann ein einzelner CLB maximal sechs Eingänge haben und daher maximal 2^6=64 verschiedene Kombinationen darstellen. Der LUT eines UND-Gatters ist hier dargestellt.

LUT Und-Gatter		
A1	A2	E
0	0	0
0	1	0
0	0	0
0	1	1

Table 2: LUT Und-Gatter

Aufgaben
1.: Notieren Sie sich die LUTs der anderen von Ihnen verwendeten Logikgatter.
2.: Notieren Sie sich ein nichtsymetrisches LUT mit zwei Eingängen. Überlegen Sie sich, wie Sie dieses in VHDL schreiben könnten.
3.: Notieren Sie sich ein LUT mit mehr als sechs Eingängen. Was macht Vivado aus diesem Code?

2.3 Unterfunktionen, Signale und std_logic_vector

2.3.1 Halbaddierer

Die LUT eines Halbaddierers ist:

Halbaddierer LUT			
A1	A2	Übertrag	Summe
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Table 3: Halbaddierer LUT

Aufgabe: Implementieren Sie einen Halbaddierer in VHDL, legen Sie dafür ein Projekt an und verbinden Sie die Ein- und Ausgänge mit sw0, sw1, ld0 und ld1.

2.3.2 Volladdierer

Die LUT eines Volladdierers ist:

Volladdierer LUT				
A1	A2	cin	Übertrag	Summe
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Table 4: Volladdierer LUT

Aufgabe: Implementieren Sie einen Volladdierer in VHDL, erweitern Sie dazu das Projekt des Halbaddieres, nutzen Sie sw15 als cin.

2.3.3 Entity und Component

Um später komplexere Schaltungen bauen zu können, ohne den Code zu unübersichtlich werden zu lassen, können Teile von Schaltungen in Module ausgegegliedert werden, um diese später im Hauptprogramm einbinden zu können. Dafür muss das Modul genau wie das Modul "main" definiert werden. Die Entity declaration muss im Code vor der architecture stehen, in der die Entity benutzt wird. Abfolge im Code:
1.: Libraries für "main"
2.: Entity declaration für "main" / Port map "main"
3.: Entity declaration für "subfunction(hier: volladdierer)" / Port map "subfunction"
4.: Architecture für "subfunction"
5.: Architecture für "main"
–> in 5. Component declaration
–> in 5. "begin"
–> in 5. Component instantiation
Die Syntax für Component declaration ist:
component KOMPONENTENNAME(hier: volladdierer) is
port(E1,E2,etc. : in std_logic;
A1,A2,etc. : out std_logic);
end component;

Die Syntax für Component instantiation ist:
INSTANZNAME(hier:voll1):KOMPONENTENNAME(hier:volladdierer)
port map(PORTNAME1 aus Component instantiation=>SIGNALNAME1 aus main,
wiederholen für alle Ports);
end component;
Damit wird ein Modul INSTANZNAME vom Typ KOMPONENTENNAME angelegt und dessen Ein- und Ausgänge mit Signalen der Haupfunktion verbunden.

Aufgabe:
Legen Sie ein neues Projekt in Vivado an und kopieren Sie Ihren Code für den Volladdierer aus der vorherigen Aufgabe. Legen Sie eine Entity und eine Component declaration an. Verbinden Sie die Ein- und Ausgänge der Komponente mit den gleichen Ein- und Ausgängen wie in der vorherigen Aufgabe.

2.3.4 Dateiausgliederung

Um den Code übersichtlicher zu halten, können Untermodule in eigene Dateien ausgegegliedert werden.
-Legen Sie ein neues Projekt an.
-Im Projektmanager rechtsklicken Sie auf "Design Sources" und wählen Sie "Add Source".
-Wählen Sie "Add or create design sources".
-Ẃahlen Sie "Create File".
Wählen SIe VHDL als Dateityp, Volladdierer als Dateinamen und "Local to project" als Speicherort.
-"Finish"
-Legen Sie die Ein- und Ausgänge wie in der Entity declaration in der vorherigen Aufgabe fest.
-Wechseln Sie im Project Manager auf "Compile Order"
-Rechtklicken Sie auf Ihre neu angelegte Datei und wählen Sie "Move to Top"
Sie haben jetzt eine Datei zu Ihrem Projekt hinzugefügt, die vor ihrem Hauptprogramm kompiliert wird.

Aufgaben:
1.: Kopieren Sie Ihren Code der Entity declaration und der Architecture aus der vorherigen Aufgabe in die neu angelegte Datei.
2.: Legen Sie, wie in der vorherigen Aufgabe, eine Component Instantiation in main und verbinden Sie die Ein- und Ausgänge wie in der vorherigen Aufgabe.
3.: Kompilieren Sie Ihr Programm und schreiben Sie es auf das Basys3-board. Wenn alles funktioniert hat, sollte es das gleiche Verhalten wie in der vorherigen Aufgabe zeigen.
4.: Überlegen Sie sich, warum die Compile Order geändert werden musste.
5.: Legen Sie eine zweite Component Instantiation Ihres Volladdieres an und verbinden Sie diese mit beliebigen unbenutzen Ports.
6.: Sehen Sie sich Ihre Schaltung mit "Elaborated Design" und "Synthesis/Schematic" an.

2.3.5 Signale

Signale sind interne Repräsentationen von Drähten und internen elektrischen Signalen. Sie werden in VHDL wie folgt angelegt:
signal SIGNALNAME1,SIGNALNAME2,etc. : SIGNALTYP;
Diese Deklaration muss in der Architecture vor "begin" stehen.
Beispiel:
signal carry1,carry2 : std_logic;
Signale können wie Outputports gesteuert werden.
signal1 <= signal2;
signal3 <= inputport;
outputport <= signal4;
Signalen kann ein Anfangswert mitgegeben werden:
signal1 <= '1';
singal2 <= '0';

Aufgaben:

1.: Legen Sie Signale für alle Ihre Ein- und Ausgänge aus Ihrem Volladdierer aus der vorherigen Aufgabe an und verbinden Sie diese so, dass die Schaltung danach die gleiche Funktion hat wie zuvor.
2.: Optional: Sollten Sie Ihren Volladdierer bisher über die dysjunkte Normalform beschrieben haben, überlegen Sie sich, wie Sie den Code mit Signalen abkürzen können. Kompilieren Sie Ihre neue Schaltung und testen Sie diese.
3.: Sehen Sie sich Ihre Schaltung mit "Elaborated Design" und "Synthesis/Schematic" an.

2.3.6 Verbindung von Komponenten

Das Blockschaltbild eines 4-bit Volladdierers ist:

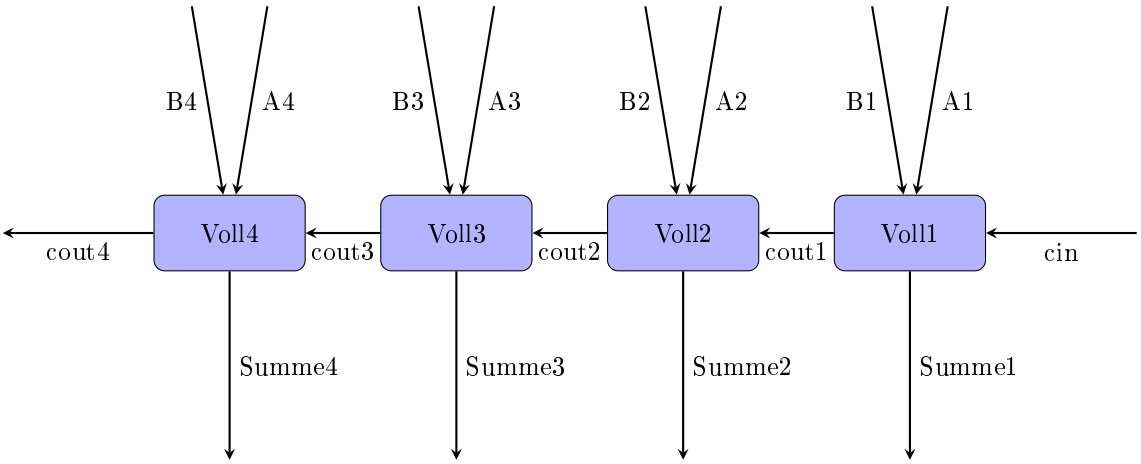

Figure 5: 4-bit Volladdierer

Aufgaben:
1.: Legen Sie ein neues Projekt an und implementieren Sie dort einen 4-bit Volladdierer. Kompilieren Sie Ihr Programm und testen Sie es auf dem Basys3-board.
2.: Sehen Sie sich Ihre Schaltung mit "Elaborated Design" und "Synthesis/Schematic" an.
3.: Welche Ein- und Ausgänge entsprechen welcher Zahl im Dezimalsystem?
4.: Erweitern Sie Ihre Schaltung auf alle Schalter und LEDs, die auf dem Basys3-board zu Verfügung stehen. Wie groß ist Ihr Volladdierer dann? Was ist das limitierende Kriterium?

2.3.7 std_logic_vector

Wenn Sie viele Ein- und Ausgänge ansteuern, wird die Port Map Ihrer Schaltung sehr lang. Diese kann jedoch verkürzt werden, indem gleiche Signaltypen in einen Vektor zusammengefasst werden. Die Syntax in der Port Map dafür ist:
VEKTORNAME : in/out std_logic_vector(VEKTORLÄNGE-1 downto 0);
Syntax, wenn Vektor als "signal" angelegt wird:
signal VEKTORNAME : std_logic_vector(VEKTORLÄNGE-1 downto 0);
Downto(VEKTORLÄNGE-1 downto 0) legt die Länge des Vektors auf VEKTORLÄNGE fest und definiert die nullte Stelle als LSB. Vivado benötigt für Vektoren eine andere Syntax in der Constraints Datei. Beispiel:
set_property -dict { PACKAGE_PIN V17 IOSTANDARD LVCMOS33 } [get_ports sw[0]];

Aufgaben:
1.: Legen Sie die Ein- und Ausgänge Ihres 4-bit Volladdierers als Vektor an und ändern Sie die Constraints Datei entsprechend.
2.: Legen Sie die Signale für ihre Komponenten als Vektoren an.
3.: Testen Sie Ihre Schaltung.
4.: Sehen Sie sich Ihre Schaltung mit "Elaborated Design" und "Synthesis/Schematic" an.

2.3.8 Generate Statement

Größere Volladdierer lassen sich nur umständlich per Hand anlegen. Dies kann jedoch über das Generate Statement gelöst werden. Damit lassen sich in einer Schleife eine definierte Anzahl von Komponenten instanzieren. Die Syntax dafür ist:
LABEL:
for i in 0 to ANZAHLKOMPONENTEN+1 generate
LABELNAME : KOMPONENTENNAME port map
(Komponentenport1=>signal1(i);Komponentenport2=>signal2(i),etc.);
end generate LABEL;
signal1 und signal2 müssen als Vektor angelegt sein, um in der Schleife zugeordnet werden zu können.

Aufgaben:
1.: Ändern Sie Ihren 4-bit Volladdierer so, dass die Komponenteninstanzierung nun über ein Generate Statement erfolgt. Ändern Sie Ihre Signale falls nötig.
2.: Sehen Sie sich Ihre Schaltung mit "Elaborated Design" und "Synthesis/Schematic" an.

2.4 Process Structure und State Machines

2.4.1 Process Structure

Prozess Statements können benutzt werden, um VHDL Module zu strukturieren, außerdem erlaubt die Prozess Struktur die Verwendung weiterer Statements, die später noch eingeführt werden. Die Syntax für ein Process Statement ist:
LABEL(optional):process(sensitivity list(optional))
declarations
begin
SEQUENZ
end process;

Aufgabe:
Legen Sie ein neues Projekt an, definieren Sie die Schalter und LEDs des Basys3 als Ein- und Ausgänge. Verbinden Sie zwei Schalter innerhalb eines Prozesses mit einem Und-Gatter und legen Sie das Signal auf eine LED.

2.4.2 If Statement

If Statements sind nur innnerhalb von Prozessen erlaubt, mit ihnen lassen sich konditionale Strukturen realisieren. Die Syntax für ein If Statement ist:
if BEDINGUNG then
SEQUENZ
elsif BEDINGUNG then
SEQUENZ
else
SEQUENZ

end if;
Um den Wert eines Signals abzufragen, wird folgende Syntax für die Bedingung verwendet:
if(SIGNAL='1') then (statt 1 könnte hier auch 0 stehen, je nachdem was man abfragen möchte).
Um die Werte eines ganzen Vektors abzufragen, kann folgende Syntax verwendet werden:
VEKTOR(X downto Y)="1111"(Länge der Sequenz von 1 ist X-Y+1)
X ist hier der erste Wert, den man abfragt, Y der letzte, es lassen sich damit auch zusammenhängende Teile eines Vektors abfragen. Innerhalb einer Bedingung können Logikgatter verwendet werden.

Aufgaben:
1.: Realisieren Sie Ihre Schaltung aus der vorherigen Aufgabe mit einem If Statement.
2.: Realisieren Sie das unten stehende LUT mithilfe eines IF Statements. Verwenden Sie kein Elsif Statement.
3.: Verbinden Sie nun zusätzlich zwei weitere Schalter mit einer LED wie in der LUT angegeben. Verwenden Sie diesmal ein Elsif Statement.
4.: Sehen Sie sich Ihre Schaltung in "Elaborated Design" und "Synthesis/Schematic" an und vergleichen Sie die Ergebnisse. Was fällt Ihnen auf?

LUT Und-Gatter		
A1	A2	E
0	0	0
0	1	1
1	0	0
1	1	1

Table 5: LUT Und-Gatter

2.4.3 Case When Statement

Das Case When Statement ist eine weitere Möglichkeit Bedingungen abzufragen. Die Syntax für das Statement ist:
case SIGNAL is
when wert1 =>
SEQUENZ
when wert2 =>
SEQUENZ
end case;
"when other" kann als Standardfall eines Case When Statements verwendet werden und wird immer dann aktiv, wenn keiner der anderen Fälle eintritt.

Aufgaben:
1.: Realisieren Sie Ihre Schaltung aus der vorherigen Aufgabe mit einem Case When Statement.
2.: Sehen Sie sich Ihre Schaltung in "Elaborated Design" und "Synthesis/Schematic" an und vergleichen Sie die Ergebnisse. Was fällt Ihnen auf?
3.: Implementieren Sie ein XOR-Gatter mit einem If Statement, einem If Elsif Statement, einem Case When Statement und als einfaches Gatter und vergleichen Sie die Ergebnisse, indem Sie die Schaltung in "Elaborated Design" und "Synthesis/Schematic" ansehen.

2.4.4 State Machines

Hier ist das Blockschaltbild eines Endlichen Automaten.

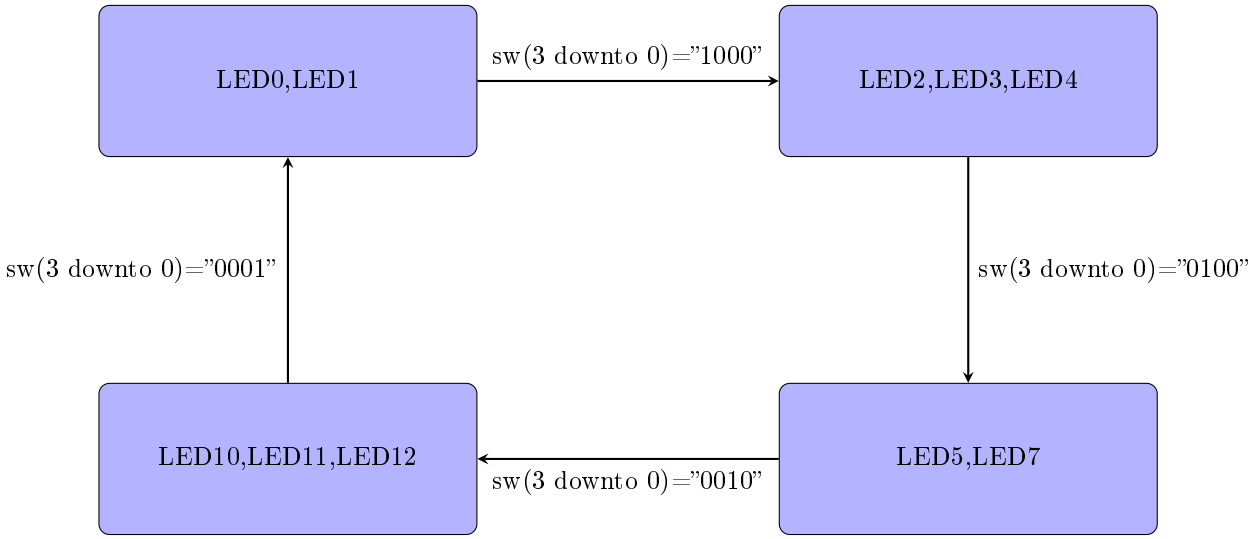

Figure 6: Endlicher Automat

Aufgabe:
1.: Implementieren Sie diesen Automaten in VHDL und bringen Sie Ihre Schaltung auf das FPGA. Zu Beginn sollte LED0 leuchten.
2.: Welche Probleme treten bei der Implementierung auf?
3.: Verwenden Sie Ihr Wissen um If Statements, um Ihren Mehrbit-Volladdierer aus der letzten Übung zu vereinfachen.

2.5 Clock Instantiation

2.5.1 Einlesen eines Frequenzsignals

Um einen endlichen Automaten richtig aufbauen zu können, ist ein Taktsignal vonnöten. Bei einer positiven Taktflanke werden dann die Signalzustände überprüft und dementsprechende Ausgabewerte generiert. Auf dem Basys3-board befindet sich ein 100MHz Oszillator. Dieser soll nun benutzt werden, um einen endlichen Automaten aufzubauen. Die Syntax, um eine Sequenz bei nach einer positiven Taktflanke zu triggern, ist:
if(rising_edge(uhr)) then
SEQUENZ
end if;

Aufgabe:
Implementieren Sie den Zustandsautomaten aus der letzten Übung erneut, fragen Sie die zum Umschalten notwendigen Zustände bei steigenden Taktflanken der Uhr ab. Erweitern Sie Ihre Constraints Datei entsprechend.

2.5.2 Frequenzteiler

Ein std_logic_vector kann mithilfe einiger Bibliotheken zu einem Frequenzteiler umfunktioniert werden. Legen Sie ein neues Projekt an und binden Sie folgende Bibliotheken in Ihr Projekt ein:
use ieee.std_logic_unsigned.all;
use IEEE.numeric_std.all;
Diese Bibliotheken erlauben Ihnen mathematische Operationen in Ihrem Code zu benutzen. Ein Frequenzteiler lässt sich dann wie folgt realisieren:
if(rising_edge(UHR)) then
TEILER<=TEILER+1;
end if;
Wobei TEILER vom Typ ein std_logic_vector ist.

Aufgaben:
1.: Implementieren Sie einen Frequenzteiler, sodass dieser das Frequenzsignal Ihrer Uhr auf etwas 1Hz herunterteilt. Legen Sie das heruntergeteilte Signal auf einen LED-Pin. Was ist die exakte Frequenz Ihres heruntergeteilten Signals?
2.: Ändern Sie ihren Frequenzteiler nun so, dass dessen Ausgangsfrequenz exakt 1Hz ist. Lassen Sie mit dem Signal wiederum eine LED blinken.
3.: Nutzen Sie Ihr Wissen um Pulsweitenmodulation, um die Helligkeit einer LED zu regulieren.

2.5.3 Synchrones und Asynchrones Design

Register/D-FFs lassen sich entweder synchron oder asynchron zurücksetzen. Die Syntax um die Register synchron zurückzusetzen, ist:

```
if(rising_edge(uhr)) then
if(reset='1') then
SEQUENZ_RESET
else
SEQUENZ
end if;
end if;
```

Die Syntax, um die Register asynchron zurückzusetzen, ist:

```
if(reset='1') then
SEQUENZ_RESET
elsif(rising_edge(uhr)) then
SEQUENZ
end if;
```

Xilinx empfiehlt für alle von Ihnen produzierten FPGAs synchrone Resets.[1]

Aufgaben:
1.: Überlegen Sie sich, welche Vor- und Nachteile synchrone gegenüber asynchronen Resets haben. Erläutern Sie dabei insbesondere die Auswirkungen auf die Metastabilität der Schaltung.
2.: Implementiere Sie eine Schaltung mit einem synchronen und einem asynchronen Reset. Sehen Sie sich die Schaltung in "Elaborated Design" und "Synthesis/Schematic" an. Was fällt Ihnen auf?

3.: Implementieren Sie die unten stehende State machine in VHDL mit einem synchronen Reset.

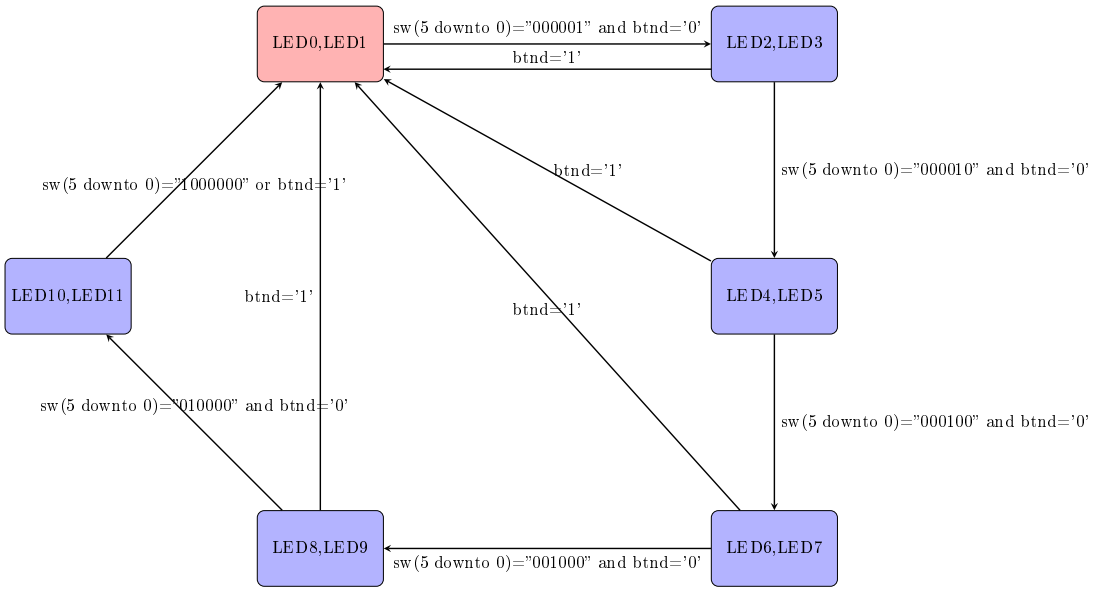

Figure 7: Endlicher Automat

4.: Implementieren Sie eine Stoppuhr mit Reset in VHDL. Stellen Sie Ihre Zeitauflösung auf 0.1s ein. Wie hoch ist die maximale Zeitauflösung, die Sie in Ihrem FPGA erreichen können? Was ist die maximale Zeitspanne, die Sie so messen können?

2.5.4 Gatterlaufzeiten

Bei zeitkritischen Designs sind die Verzögerungszeiten in Gattern und Leitungen zu beachten. Typische Verzögerungszeiten lassen sich dem Datenblatt des FPGA entnehmen. Man kann diese aber auch selbst messen.

Aufgabe:
Generieren Sie einen Ringoszillator und einen Frequenzteiler geeigneter Größe, um eine LED so umzuschalten, dass der Schaltvorgang mit dem Auge sichtbar ist. Messen Sie grob die Zeit, die ein Schaltvorgang dauert. Berechnen Sie daraus die Frequenz Ihres Ringoszillators und überlegen Sie sich, wie sich daraus die Gatter und Signal Laufzeiten abschätzen lassen.

2.5.5 Vivado Clock Wizard

Legen Sie ein neues Projekt an und wählen Sie unter "Project Manager" den "IP Catalog". Hier finden Sie vorgefertigte Module(Komponenten), die Sie direkt in Ihr Projekt einbinden können.
-Wählen Sie "FPGA Features and Design"
-Wählen Sie "Clocking Wizard"
-Im nun geöffneten Fenster wählen Sie als Optionen PLL, Frequency Synthesis, Phase Alignement, Balanced, Input Frequency =100(MHz)
-Unter "Output Clocks" wählen Sie clk_out1,Output Freq=320(MHz),reset,locked,Active High,Automatic Control On-Chip
-OK

-Out of Context for IP–>"Generate"
-Ok
-Öffnen Sie den Project Manager und wählen Sie Ihr neu erstelltes Modul "clk_wiz_0" und wechseln Sie zu "clk_wiz_0.v"
-Unter "module clk_wiz_0" finden Sie die Port map des Moduls, damit können Sie das Modul genau wie eine Komponente in Ihrem Hauptprogramm einbinden.

Aufgabe:
Instanzieren Sie die soeben angelegte Phasenregelschleife in Ihrem Hauptprogramm. Genererieren Sie dadurch eine schnelle Uhr und testen Sie, ob die Ausgangsfrequenz des Moduls tatsächlich 320MHz ist.

2.6 Arithmetik, Arrays und Variablen

2.6.1 ieee.numeric_std.all Bibliothek

In der ieee.numeric_std.all Bibliothek sind mathematische Funktionen enthalten. Funktionen können nicht sequentiell sein. Funktionen können verwendet werden, um Code zu vereinfachen. Außerdem sind die Funktionen aus der Bibliothek bereits optimiert. Man sollte diese verwenden, anstatt den Code selber zu schreiben.

Aufgaben:
1.: Sehen Sie sich die Funktionen "+" und "*" in der ieee.numeric_std.all Bibliothek an und vergleichen Sie die "+" Funktion mit Ihrem Volladdierer.
2.: Implementieren Sie einen Multiplikator mit der Funktion "*" aus der ieee.numeric_std.all Bibliothek und mit denen Ihnen bekannten Methoden ohne Verwendung der Bibliothek. Sehen Sie sich die Schaltungen in "Elaborated Design" und "Synthesizer/Schematic" an und vergleichen Sie die Ergebnisse.

2.6.2 Arithmetische Funktionen

Die ieee.numeric_std.all Bibliothek enthält Funktionen zu Typumwandlung und arithmetischen Operationen auf Integern. Funktionen haben in VHDL immer einen Rückgabeparameter und können mehrere Eingabeparameter haben. Der Rückgabewert kann wie ein Signal gespeichert werden. Die Syntax für die Typumwandlung zwischen std_logic_vector und Unsigned ist:

Signal a: std_logic_vector (Länge downto 0);
Signal b: unsigned (Länge downto 0);
b<=unsigned(a);
a und b müssen die gleiche Länge haben. Falls dies nicht der Fall ist, kann man mit der Funktion "&" zwei Vektoren oder Unsigned zusammenfügen. Syntax:
Signal c: unsigned (Länge*2 downto 0);
c<=b&b;
Zur Erinnerung die Syntax der For-Schleife:
for I in Länge to 0 loop
SEQUENZ
if(BEDINGUNG) then
exit;
end if;
end loop;
Das exit Statement kann benutzt werden, um die Schleife vorzeitig zu beenden.

Aufgaben:
1.: Sehen Sie sich die Definition der "Shift_Left"-Funktion unter der folgenden Adresse an. Benutzen Sie dies, um eine Zahl mit zwei zu multiplizieren. Testen Sie Ihre Schaltung.
https://www.csee.umbc.edu/portal/help/VHDL/packages/numeric_std.vhd
2.: Erweitern Sie Ihre Schaltung zu einem kombinatorischen 8x8 Multiplizierer. Sehen Sie sich die Schaltung in "Elaborated Design" und in "Synthesizer/Schematic" an und vergleichen Sie die Ergebnisse mit denen Ihres ersten Multiplizierers.
3.: Überlegen Sie sich, wie Sie eine Shift-Funktion für std_logic_vector implementieren könnten.

2.6.3 Arrays

Mit Arrays lässt sich VHDL-Code vereinfachen. Die Syntax, um in VHDL ein Array anzulegen, ist:
type TYPENNAME is array of (LÄNGE) of ELEMENTTYP;

Aufgabe:
Verwenden Sie Ihr Wissen um Arrays und For-loops, um Ihren kombinatorischen Multiplizierer zu vereinfachen. Sehen Sie sich Ihre Schaltung in "Elaborated Design" und "Synthesis/Schematic" an und vergleichen Sie die Ergebnisse mit denen aus der vorherigen Aufgabe.

2.6.4 Variablen

Signale werden in FPGAs, wenn nicht anders angelegt, parallel verarbeitet. Wenn man VHDL in serielle Strukturen zwingen möchte, kann man Variablen verwenden. Variablen können nur innerhalb von Prozessen angelegt werden. Die Syntax dafür ist:
variable VARIABLENNAME : VARIABLENTYP;
Die Variablentypen sind die gleichen wie bisher bei Signalen.

Aufgaben:
1.: Implementieren Sie eine einfache Schaltung, mit der Sie die Unterschiede zwischen Signalen und Variablen demonstrieren können.
2.: Sehen Sie sich Ihre Schaltung in Elaborated Design" und "Synthesis/Schematic" an und vergleichen Sie die Ergebnisse.

2.6.5 Vergleichsfunktionen

Um die Größe von Vektoren/Integern zu vergleichen, können Vergleichsfunktionen benutzt werden. Für std_logic_vector sind die Vergleichsfunktionen standardmäßig definiert, für Integer wird die ieee.numeric_std.all Bibliothek verwendet.

Aufgaben:
1.: Implementieren Sie eine Schaltung, um die Vergleichsfunktion zu demonstrieren.
2.: Implementieren Sie eine eigene Vergleichsfunktion für Std_logic_vector.

2.7 Serielle Kommunikation

2.7.1 UART

Das Basys3-board kann über eine JTAG-UART-Schnittstelle mit einem externen Gerät kommunizieren, beispielsweise einem PC. Über diese Schnittstelle haben Sie bisher das FPGA

programmiert. Die Daten werden byteweise seriell über die Schnittstelle versendet. Zusätzlich wird eine logische 1 als Start-Bit und eine logische 0 als Stop-Bit mitgesendet. Die Bits werden in einem Schieberegister gespeichert, bis sie gesendet werden können. In VHDL kann eine UART-Schnitstelle wie folgt instanziert werden:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.std_logic_unsigned.all;

entity UART_TX_CTRL is
    Port ( SEND : in  STD_LOGIC;
           DATA : in  STD_LOGIC_VECTOR (7 downto 0);
           CLK : in  STD_LOGIC;
           READY : out  STD_LOGIC;
           UART_TX : out  STD_LOGIC);
end UART_TX_CTRL;

architecture Behavioral of UART_TX_CTRL is

type TX_STATE_TYPE is (RDY, LOAD_BIT, SEND_BIT);

constant BIT_TMR_MAX : std_logic_vector(13 downto 0) := "10100010110000";
--Hiermit wird der Frequenzteiler für die BAUD rate eingestellt

constant BIT_INDEX_MAX : natural := 10;

signal bitTmr : std_logic_vector(13 downto 0) := (others => '0');
--Zählt mit Uhrfrequenz  hoch bis bit_tmr_max,
--signalisiert, wie lange das aktuelle bit stabil gehalten wurde

signal bitDone : std_logic;
--wird aktiv, wenn bittmr=bit_tmr_max, zeigt an,
 dass das nächste bit geladen werden kann

signal bitIndex : natural;
--speichert den Index des nächsten zu ladenen bits

signal txBit : std_logic := '1';
--speichert das gerade zu sendene bit

signal txData : std_logic_vector(9 downto 0);
--enthält das gesamte zu sendene Datenpaket,
--also Start-Bit, 1 byte Daten, Stop-Bit

signal txState : TX_STATE_TYPE := RDY;

begin

next_txState_process : process (CLK)
begin
if (rising_edge(CLK)) then
```

```
case txState is
when RDY =>
if (SEND = '1') then
txState <= LOAD_BIT;
end if;
when LOAD_BIT =>
txState <= SEND_BIT;
when SEND_BIT =>
if (bitDone = '1') then
if (bitIndex = BIT_INDEX_MAX) then
txState <= RDY;
else
txState <= LOAD_BIT;
end if;
end if;
when others=>
txState <= RDY;
end case;
end if;
end process;
--Dieser Prozess schaltet die einzelnen Zustände des Übertragungszustandes durch,
--ist ein simpler Zustandsautomat

bit_timing_process : process (CLK)
begin
if (rising_edge(CLK)) then
if (txState = RDY) then
bitTmr <= (others => '0');
else
if (bitDone = '1') then
bitTmr <= (others => '0');
else
bitTmr <= bitTmr + 1;
end if;
end if;
end if;
end process;
--Dieser Prozess zählt bitTmr hoch und setzt es zurück

bitDone <= '1' when (bitTmr = BIT_TMR_MAX) else '0';
--Zeigt an, dass das nächste bit geladen werden muss

bit_counting_process : process (CLK)
begin
if (rising_edge(CLK)) then
if (txState = RDY) then
bitIndex <= 0;
elsif (txState = LOAD_BIT) then
bitIndex <= bitIndex + 1;
end if;
end if;
end process;
```

```
--Dieser Prozess schaltet die einzelnen zu sendenen Bits (10 Stück)
--eines einzelnen Übertragungsvorganges durch

tx_data_latch_process : process (CLK)
begin
if (rising_edge(CLK)) then
if (SEND = '1') then
txData <= '1' & DATA & '0';
end if;
end if;
end process;
--Dieser Prozess lädt das aktuelle Datenbyte
--mit dem Start- und Stop-Bit in den zu sendenen Vektor

tx_bit_process : process (CLK)
begin
if (rising_edge(CLK)) then
if (txState = RDY) then
txBit <= '1';
elsif (txState = LOAD_BIT) then
txBit <= txData(bitIndex);
end if;
end if;
end process;
--Dieser Prozess lädt das aktuelle Datenbit aus dem gesamten Bitvektor

UART_TX <= txBit;
--überträgt das aktuelle zu sendene Bit in das Hauptprogramm
READY <= '1' when (txState = RDY) else '0';
--Überträgt den Zustand "Ready" an das Hauptprogramm

end Behavioral;
```

[4]
Die Kommentare sollten dem Verständnis helfen. Verwenden Sie diese ab sofort, wenn Sie langen Code schreiben, um das Verständnis des Codes zu erleichtern.

2.7.2 BAUD rate

Die Baudrate gibt an, mit welcher Frequenz Datenbits über die serielle Schnittstelle übertragen werden. Für das Basys3-board wird eine BAUD rate von 9600 empfohlen. Das heißt, dass 9600 bits pro Sekunde übertragen werden.[5]

2.7.3 main

Im Hauptprogramm werden die zu sendenen Daten festgelegt und dann in der richtigen Reihenfolge zum Versenden in die UART-Komponente geladen.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.std_logic_unsigned.all;
```

```
entity main is
    Port ( CLK : in std_logic;
           UART_TXD  : out  STD_LOGIC;
           BTN  : in  STD_LOGIC_VECTOR (4 downto 0));
end main;

architecture Behavioral of main is

component UART_TX_CTRL
Port(
SEND : in std_logic;
DATA : in std_logic_vector(7 downto 0);
CLK : in std_logic;
READY : out std_logic;
UART_TX : out std_logic
);
end component;

type UART_STATE_TYPE is (RST_REG, LD_INIT_STR,
 SEND_CHAR, RDY_LOW, WAIT_RDY, WAIT_BTN, LD_BTN_STR);
--Hier werden die einzelnen Zustände der Übertragung festgelegt
type CHAR_ARRAY is array (integer range<>) of std_logic_vector(7 downto 0);

constant RESET_CNTR_MAX : std_logic_vector(17 downto 0) := "110000110101000000";
-- 100,000,000 * 0.002 = 200,000 = clk cycles per 2 ms
constant MAX_STR_LEN : integer := 27;
constant WELCOME_STR_LEN : natural := 27;
constant BTN_STR_LEN : natural := 24;

constant WELCOME_STR : CHAR_ARRAY(0 to 26) := (X"0A",X"0D",X"42",X"41",X"53",X"59",X"53",
X"33",X"20",X"47",X"50",X"49",X"4F",X"2F",X"55",X"41",X"52",X"54",X"20",X"44",X"45",X"4D",X"4F",
X"21",X"0A",X"0A",X"0D");
--Dies ist die Hexadezimaldarstellung von Daten in VHDL

constant BTN_STR : CHAR_ARRAY(0 to 23) := (X"42",X"75",X"74",X"74",X"6F",X"6E",X"20",X"70",X"72",
X"65",X"73",X"73",X"20",X"64",X"65",X"74",X"65",X"63",X"74",X"65",X"64",X"21",X"0A",X"0D");

signal sendStr : CHAR_ARRAY(0 to (MAX_STR_LEN - 1));
--Hier werden die zu sendenen Daten zwischengespeichert
signal strEnd : natural;
--Markiert das Ende der zu sendenen Daten
signal strIndex : natural;
--Hiermiet werden die einzelnen Byte der zu sendenen Daten durchgeschaltet

signal uartRdy : std_logic;
--Signalisiert, dass das nächste Bit gesendet werden kann
signal uartSend : std_logic := '0';
--Leitet Signal, dass das nächste bit gesendet werden kann an, an die UART-Komponente weiter
signal uartData : std_logic_vector (7 downto 0):= "00000000";
--speichert das aktuell zu sendene Datenbyte
signal uartTX : std_logic;
--Enthält das aktuell zu sendene Bit
```

```
signal uartState : UART_STATE_TYPE := RST_REG;
--Instanziert einen Zustandsautomaten mit den oben festgelegten Zuständen
signal reset_cntr : std_logic_vector (17 downto 0) := (others=>'0');
--Zählt hoch bis reset_cntr_max, um einen Reset auszulösen, falls nötig

signal btnDetect : std_logic;
--signalisiert, dass Knopf gedrückt wurde
signal btnDeBnc : std_logic_vector (4 downto 0);
--Normalerweise benutzt, um falsch positive Eingaben beim Knopfdruck zu vermeiden,
--hier der Einfachheit halber deaktiviert

begin

btnDetect<=btn(0);
btnDeBnc(4)<=btn(0);

process(CLK)
begin
  if (rising_edge(CLK)) then
    if ((reset_cntr = RESET_CNTR_MAX) or (uartState /= RST_REG)) then
      reset_cntr <= (others=>'0');
    else
      reset_cntr <= reset_cntr + 1;
    end if;
  end if;
end process;
--Dieser Prozess löst nach einer bestimmten Zeit einen reset aus, falls nötig

next_uartState_process : process (CLK)
begin
if (rising_edge(CLK)) then
if (btnDeBnc(4) = '1') then
uartState <= RST_REG;
else
case uartState is
when RST_REG =>
        if (reset_cntr = RESET_CNTR_MAX) then
          uartState <= LD_INIT_STR;
        end if;
when LD_INIT_STR =>
uartState <= SEND_CHAR;
when SEND_CHAR =>
uartState <= RDY_LOW;
when RDY_LOW =>
uartState <= WAIT_RDY;
when WAIT_RDY =>
if (uartRdy = '1') then
if (strEnd = strIndex) then
uartState <= WAIT_BTN;
else
uartState <= SEND_CHAR;
```

```
end if;
end if;
when WAIT_BTN =>
if (btnDetect = '1') then
uartState <= LD_BTN_STR;
end if;
when LD_BTN_STR =>
uartState <= SEND_CHAR;
when others=> --should never be reached
uartState <= RST_REG;
end case;
end if ;
end if;
end process;
--Dieser Prozess schaltet die einzelnen Zustände des Übertragungsvorganges durch

string_load_process : process (CLK)
begin
if (rising_edge(CLK)) then
if (uartState = LD_INIT_STR) then
sendStr <= WELCOME_STR;
strEnd <= WELCOME_STR_LEN;
elsif (uartState = LD_BTN_STR) then
sendStr(0 to 23) <= BTN_STR;
strEnd <= BTN_STR_LEN;
end if;
end if;
end process;
--Dieser Prozess wählt die zu sendenen Daten aus

char_count_process : process (CLK)
begin
if (rising_edge(CLK)) then
if (uartState = LD_INIT_STR or uartState = LD_BTN_STR) then
strIndex <= 0;
elsif (uartState = SEND_CHAR) then
strIndex <= strIndex + 1;
end if;
end if;
end process;
--Dieser Prozess inkrementiert den Index für die zu sendenen Byte

char_load_process : process (CLK)
begin
if (rising_edge(CLK)) then
if (uartState = SEND_CHAR) then
uartSend <= '1';
uartData <= sendStr(strIndex);
else
uartSend <= '0';
end if;
end if;
```

```
end process;
--Dieser Prozess lädt das zu sendene Byte

Inst_UART_TX_CTRL: UART_TX_CTRL port map(
SEND => uartSend,
DATA => uartData,
CLK => CLK,
READY => uartRdy,
UART_TX => uartTX
);

UART_TXD <= uartTX;
--Instanzierung der UART Komponente

end Behavioral;
```

2.7.4 Teraterm und Ascii

Um die gesendeten Daten auf Ihrem Computer zu sehen, benötigen Sie ein serielles Interface, das die Nachricht sichtbar macht. Digilent empfiehlt für das Basys3-board Teraterm. Realterm hat jedoch den Vorteil, dass man einstellen kann, wie Daten dargestellt werden.

2.7.5 Aufgaben

Aufgaben:
1.: Senden Sie eine andere Nachricht als die voreingestellte an Ihren Computer.
2.: Ändern Sie die Baudrate in dem Projekt und beobachten Sie, welche Zeichen Sie dann an Ihrem PC empfangen. Was fällt Ihnen auf? Woran könnte das liegen?
3.: Implementieren Sie einen Debouncer. Wofür wird dieser benutzt?
4.: Implementieren Sie ein Frequenzmessgerät, übertragen Sie die gemessene Frequenz an Ihren Computer und überlegen Sie sich, wie Sie damit die Gatterlaufzeiten des FPGA bestimmen können. Sehen Sie sich die Schaltung in "Elaborated Design" und "Synthesis/Schematic" an.

2.8 Simulation

2.8.1 Behavioral Simulation

Laden Sie Ihr Projekt, in dem Sie Signale und Variablen verglichen haben.
-Wählen Sie im Flow Navigator "Run Simulation"
-Wählen Sie "Run Behavioral Simulation"
-Im nun offenen Fenster sehen Sie die Signale und Variablen der von Ihnen geladenen Entity (in diesem Projekt nur "main")
-Unter "Untitled" rechtsklicken Sie auf Ihre Schalter (sw[15:0]) und wählen Sie "Force Constant" und setzen Sie alle Schalter auf "1"
-Unter "Untitled" rechtsklicken Sie auf Ihren Knopf (btnd) und wählen Sie "Force Clock"
-"Value Radix": binary, "Leading edge value": 1, "Trailing edge value": 0, "Period" : 10ns. Lassen Sie die anderen Werte unverändert
-Sie sollten unten in der "Tcl Console" die beiden Befehle
add_force {/main/sw} -radix bin {1111111111111111 0ns} und
add_force {/main/btnd} -radix bin {1 0ns} {0 5000ps} -repeat_every 10000ps sehen.
-Schreiben Sie "run 50 ns" in die Konsole.

-Sehen Sie sich das Ergebnis Ihrer Simulation an.

Aufgaben:
1.: Erklären Sie, was Sie gerade eben simuliert haben.
2.: Kopieren Sie Ihre Eingaben aus der Konsole in eine Textdatei und ändern Sie die Dateiendung in ".tcl". Schreiben Sie "Restart" in die erste Zeile. Laden Sie diese Datei dann in die Simulation, indem Sie den kompletten Dateipfad in die Konsole schreiben. (Hinweis: /–>\ ). Schalten Sie dann die Schalter ein und aus und beobachten Sie das Ergebnis.

2.8.2 Post-Synthesis Functional Simulation

-Wählen Sie im Flow Navigator "Run Simulation"
-Wählen Sie "Run Post-Synthesis Functional Simulation"
-Laden Sie Ihre tcl-Datei über die Konsole
-Verwenden Sie "Zoom Fit" im Simulationsfenster, falls nötig
-Rechtklicken Sie auf Ihr Hauptprogramm ("main") unter "Scopes"
-Wählen Sie "Go to Source Code"
-Sie können nun in "main(Hauptprogramm)_funct_synth.vhd" sehen, was Vivado aus Ihrem VHDL Code nach der Synthese gemacht hat

Aufgaben:
1.: Erklären Sie, was Sie gerade eben simuliert haben.
2.: Warum sind einige der Singnale am Anfang unbestimmt (mit X markiert)?
3.: Warum werden die internen Signale und Variablen nicht angezeigt?
4.: Erweitern Sie Ihre Simulation um weitere Schalterwechsel und beobachten Sie das Ergebnis.

2.8.3 Post-Synthesis Timing Simulation

-Wählen Sie im Flow Navigator "Run Simulation"
-Wählen Sie "Run Post-Synthesis Timing Simulation"
-Laden Sie Ihre tcl-Datei über die Konsole
-Verwenden Sie "Zoom Fit" im Simulationsfenster, falls nötig

Aufgaben
1.: Erklären Sie, was Sie gerade eben simuliert haben.
2.: Welches Ergebnis sehen Sie?
3.: Was ist der Unterschied zur vorherigen Simulation?

2.8.4 Post-Imlementation Functional Simulation

-Wählen Sie im Flow Navigator "Run Simulation"
-Wählen Sie "Run Post-Implementation Functional Simulation"
-Laden Sie Ihre tcl-Datei über die Konsole
-Verwenden Sie "Zoom Fit" im Simulationsfenster, falls nötig

Aufgaben
1.: Erklären Sie, was Sie gerade eben simuliert haben.
2.: Welches Ergebnis sehen Sie?
3.: Was ist der Unterschied zur Post-Synthesis Functional Simulation?

2.8.5 Post-Imlementation Timing Simulation

-Wählen Sie im Flow Navigator "Run Simulation"
-Wählen Sie "Run Post-Implementation Functional Simulation"
-Laden Sie Ihre tcl-Datei über die Konsole
-Verwenden Sie "Zoom Fit" im Simulationsfenster, falls nötig

Aufgaben
1.: Erklären Sie, was Sie gerade eben simuliert haben.
2.: Welches Ergebnis sehen Sie?
3.: Was ist der Unterschied zur Post-Implemenation Functional Simulation?
4.: Was ist der Unterschied zur Post-Synthesis Timing Simulation?

2.9 Digital Regelung

2.9.1 Phasen-Frequenz-Detektor

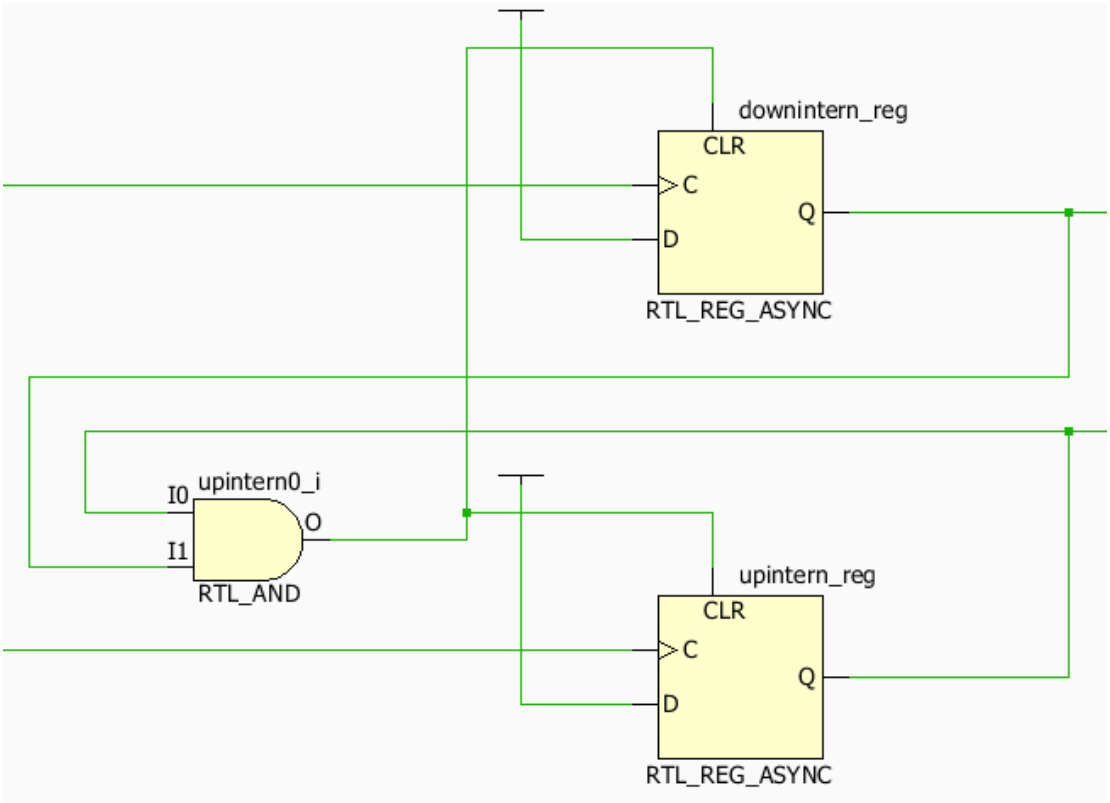

Figure 8: Blockschaltbild PFD

Ein simpler digitaler Phasen-Frequenz-Detektor hat zwei Eingänge für die beiden zu vergleichenden Eingangsfrequenzen und zwei Ausgänge für die Steuersignale der Ladungspumpe.[3]

Aufgaben:
1.: Erklären Sie anhand des Blockschaltbildes die Funktionsweise eines Phasen-Frequenz-Detektors.

2.: Implementieren Sie in VHDL ein Modul für einen PFD und testen Sie die Schaltung durch ein Simulationstool.

2.9.2 Digitale Ladungspumpe

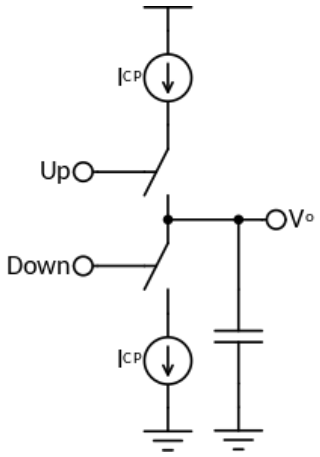

Figure 9: Blockschaltbild Analoge Ladungspumpe

Eine analoge Ladungspumpe, wie hier dargestellt, integriert einen Strom über einen Kondensator und stellt dadurch die Ausgangsspannung V_0 ein. Der Strom wird über die Schalter "Up" und "Down" gesteuert. Wenn "Up" geschlossen ist und "Down" offen, dann wird der Kondensator über die obere Stromquelle geladen. Wenn "Down" geschlossen ist und "Up" offen, wird der Kondensator über die untere Stromquelle entladen. In den beiden anderen Fällen bleibt die Spannung im Kondensator konstant. [7]

Aufgaben:
1.: Skizzieren Sie die Ladekurve für diese Art von Ladungspumpe. Wie würde die Kurve aussehen, wenn der Kondensator über eine Spannungsquelle geladen wird?
2.: Implementieren Sie eine digitale Ladungspumpe in VHDL. Was ist das Äquivalent des Kondensators und des Ladungsstromes und welchem der beiden analogen Ladungsmodelle entpricht dies?

2.9.3 Numerisch kontrollierter Oszillator

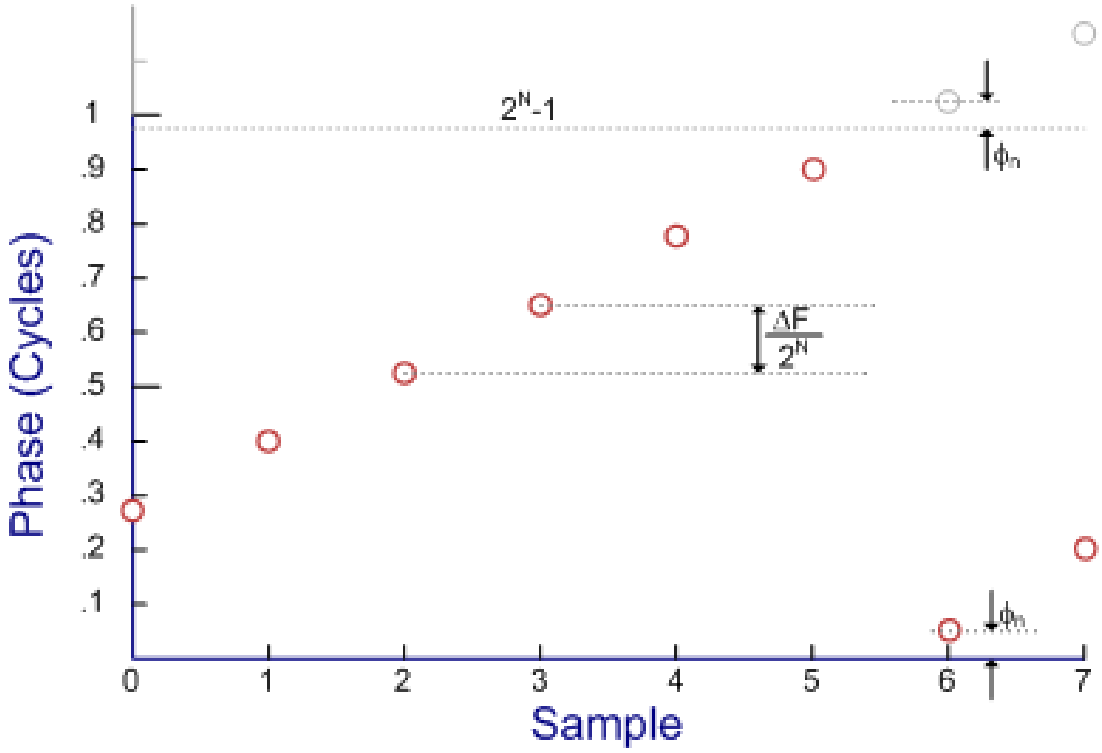

Figure 10: Phasenakkumulator

Ein Phasenakkumulator ist eine Inkrementier-Schaltung, die einen Registerwert mit jedem Taktzyklus um einen variablen Wert erhöht.[8]

Aufgaben:
1.: Erklären Sie anhand des Bildes, wie Sie einen Phasenakkumulator für eine regelbare Signalquelle nutzen können.
2.: Implementieren Sie einen NCO mit einem Rechtecksignal als Ausgang in VHDL.
3.: Überlegen Sie sich, wie Sie diese Schaltung erweitern können, um ein Sinussignal zu erzeugen.

2.9.4 Phasenregelschleife

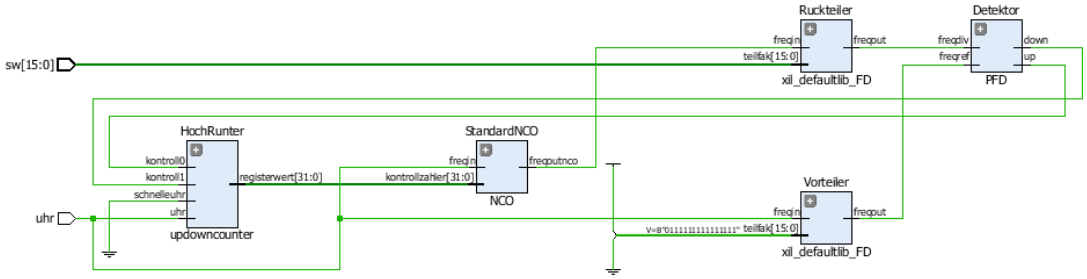

Figure 11: Blockschaltbild PLL

Eine Phasenregelschleife nimmt eine Eingangsfrequenz entgegegen und synchronisiert ein Ausgangssignal mit diesem Eingangssignal. Durch den Einsatz von Frequenzteilern lassen sich dadurch verschiedene Frequenzen synthetisieren, deren Langzeitstabilität der des Eingangssignals entspricht.

Aufgabem:
1.: Implementieren Sie eine Phasenregelschleife aus den einzelnen Modulen, die Sie bisher erstellt haben. Testen Sie Ihre Ausgangsfrequenz.
2.: Erweitern Sie Ihre Schaltung um einen einstellbaren Vorteiler, der Ihre Eingangsfrequenz herunterteilt. Testen Sie Ihre Ausgangsfrequenz.
3.: Erweitern Sie Ihre Schaltung um einen eintellbaren Frequenzteiler in der Rückführung. Stellen die den Teilerfaktor Ihres Vorteilers auf 2^{-16} ein. Testen Sie Ihre Ausgangsfrequenz.

2.9.5 Digitale Filter

Wie an der vorherigen Aufgabe sichtbar wurde, ist das Signal sehr ungenau. Man kann das Signal jedoch mit Filtern stabilisieren. Die Formel für einen Filter mit endlicher Impulsantwort (FIR-Filter) ist:
$y[n] = \sum_{i=1}^{N} b_i * x[n-i]$
mit y[n] als Ausgangssignal/Ausgangssignalvektor, x[n] als Eingangssignal/Eingangssignalvektor [6]

Aufgaben:
1.: Um welche Art Filter handelt es sich hier? Was ist das analoge Äquivalent?
2.: Implementieren Sie einen solchen Filter in VHDL, setzen Sie $b_i = 0$. Überlegen Sie sich, an welcher Stelle Ihrer PLL dieser Filter eingebaut werden müsste.
3.: Integrieren Sie Ihren Filter in Ihre bestehende PLL und testen Sie die Frequenz.

3 Lösungen

3.1 Erstes Projekt

3.1.1 VHDL

```
library IEEE;
use IEEE.STD\_LOGIC_1164.ALL;
```

```
entity main is
    Port ( sw0 : in STD\_LOGIC;
           btnd : in STD\_LOGIC;
           ld0 : out STD\_LOGIC;
           ld5 : out STD\_LOGIC);
end main;

architecture Behavioral of main is
begin

ld0 <= sw0;
ld5 <= btnd;

end Behavioral;
verbatim

\subsubsection{Contraints}

\begin{verbatim}
set\_property -dict \{ PACKAGE\_PIN V16   IOSTANDARD LVCMOS33 \} [get\_ports \{ sw0\}];
set\_property -dict \{ PACKAGE\_PIN u17   IOSTANDARD LVCMOS33 \} [get\_ports \{ btnd\}];

set\_property -dict \{ PACKAGE\_PIN e19   IOSTANDARD LVCMOS33 \} [get\_ports \{ ld0\}];
set\_property -dict \{ PACKAGE\_PIN u15   IOSTANDARD LVCMOS33 \} [get\_ports \{ ld5\}];
```

3.1.2 Weitere Lösung

1.5.3: Das FPGA hat flüchtigen Speicher, wenn man die Spannung abschaltet wird das Programm gelöscht.

3.2 Logikgatter in VHDL und FPGAs

3.2.1 Einfache Gatter

VHDL_main:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity main is
    Port ( sw0 : in STD_LOGIC;
           sw1 : in STD_LOGIC;
           sw2 : in STD_LOGIC;
           sw3 : in STD_LOGIC;
           sw4 : in STD_LOGIC;
           sw5 : in STD_LOGIC;
           sw6 : in STD_LOGIC;
           sw7 : in STD_LOGIC;
           sw8 : in STD_LOGIC;
           sw9 : in STD_LOGIC;
           sw10 : in STD_LOGIC;
```

```
            sw11 : in STD_LOGIC;
            sw12 : in STD_LOGIC;
            sw13 : in STD_LOGIC;
            sw14 : in STD_LOGIC;
            ld0 : out STD_LOGIC;
            ld2 : out STD_LOGIC;
            ld4 : out STD_LOGIC;
            ld6 : out STD_LOGIC;
            ld8 : out STD_LOGIC;
            ld10 : out STD_LOGIC;
            ld12 : out STD_LOGIC;
            ld14 : out STD_LOGIC);
end main;

architecture Behavioral of main is
begin

ld0 <= sw0 and sw1;
ld2 <= sw2 or sw3;
ld4 <= sw4 nand sw5;
ld6 <= sw6 nor sw7;
ld8 <= sw8 xor sw9;
ld10 <= sw10 xnor sw11;
ld12 <= not sw12;

end Behavioral;
```

Constraints:

```
set_property -dict { PACKAGE_PIN V17   IOSTANDARD LVCMOS33 } [get_ports { sw0}];
set_property -dict { PACKAGE_PIN V16   IOSTANDARD LVCMOS33 } [get_ports { sw1}];
set_property -dict { PACKAGE_PIN w16   IOSTANDARD LVCMOS33 } [get_ports { sw2}];
set_property -dict { PACKAGE_PIN w17   IOSTANDARD LVCMOS33 } [get_ports { sw3}];
set_property -dict { PACKAGE_PIN w15   IOSTANDARD LVCMOS33 } [get_ports { sw4}];
set_property -dict { PACKAGE_PIN v15   IOSTANDARD LVCMOS33 } [get_ports { sw5}];
set_property -dict { PACKAGE_PIN w14   IOSTANDARD LVCMOS33 } [get_ports { sw6}];
set_property -dict { PACKAGE_PIN w13   IOSTANDARD LVCMOS33 } [get_ports { sw7}];
set_property -dict { PACKAGE_PIN v2   IOSTANDARD LVCMOS33 } [get_ports { sw8}];
set_property -dict { PACKAGE_PIN t3   IOSTANDARD LVCMOS33 } [get_ports { sw9}];
set_property -dict { PACKAGE_PIN t2   IOSTANDARD LVCMOS33 } [get_ports { sw10}];
set_property -dict { PACKAGE_PIN r3   IOSTANDARD LVCMOS33 } [get_ports { sw11}];
set_property -dict { PACKAGE_PIN w2   IOSTANDARD LVCMOS33 } [get_ports { sw12}];

set_property -dict { PACKAGE_PIN U16   IOSTANDARD LVCMOS33 } [get_ports { ld0}];
set_property -dict { PACKAGE_PIN u19   IOSTANDARD LVCMOS33 } [get_ports { ld2}];
set_property -dict { PACKAGE_PIN w18   IOSTANDARD LVCMOS33 } [get_ports { ld4}];
set_property -dict { PACKAGE_PIN u14   IOSTANDARD LVCMOS33 } [get_ports { ld6}];
set_property -dict { PACKAGE_PIN v13   IOSTANDARD LVCMOS33 } [get_ports { ld8}];
set_property -dict { PACKAGE_PIN w3   IOSTANDARD LVCMOS33 } [get_ports { ld10}];
set_property -dict { PACKAGE_PIN p3   IOSTANDARD LVCMOS33 } [get_ports { ld12}];
```

4.:

Jedes Logikgatter wird in eine LUT übersetzt. Das FPGA hat intern keine Logikgatter fest verbaut.

3.2.2 Look Up Table

1.: Wahrheitstabellen der Logikgatter.
2.: Einer der beiden Eingänge muss mit "not" invertiert werden. Ein nichtsymmetrisches LUT mit zwei Eingängen kann immer in ein Nicht-Gatter übersetzt werden.
3.: Es schaltet zwei CLBs in Reihe.

3.3 Unterfunktionen, Signale und std_logic_vector

3.3.1 Halbaddierer

VHDL_main:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity main is
    Port ( sw0 : in STD_LOGIC;
           sw1 : in STD_LOGIC;

           ld0 : out STD_LOGIC;
           ld1 : out STD_LOGIC);
end main;

architecture Behavioral of main is

begin

ld0 <= sw0 and sw1;
ld1 <= sw0 xor sw1;

end Behavioral;
```

Constraints:

```
set_property -dict { PACKAGE_PIN V17   IOSTANDARD LVCMOS33 } [get_ports { sw0}];
set_property -dict { PACKAGE_PIN V16   IOSTANDARD LVCMOS33 } [get_ports { sw1}];

set_property -dict { PACKAGE_PIN U16   IOSTANDARD LVCMOS33 } [get_ports { ld0}];
set_property -dict { PACKAGE_PIN e19   IOSTANDARD LVCMOS33 } [get_ports { ld1}];
```

3.3.2 Volladdierer

VHDL_main.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity main is
```

```
    Port ( sw0 : in STD_LOGIC;
           sw1 : in STD_LOGIC;
           sw15 : in STD_LOGIC;

           ld0 : out STD_LOGIC;
           ld1 : out STD_LOGIC);
end main;

architecture Behavioral of main is
begin

Summe <= (not cin and not A1 and A2) or (not cin and A1 and not A2)
or (cin and not A1 and not A2) or (cin and A1 and A2);
Ubertrag <= (not cin and A1 and A2) or (cin and not A1 and A2)
or (cin and A1 and not A2) or (cin and A1 and A2);

end Behavioral;
```

Contraints:

```
set_property -dict { PACKAGE_PIN V17   IOSTANDARD LVCMOS33 } [get_ports { sw0}];
set_property -dict { PACKAGE_PIN V16   IOSTANDARD LVCMOS33 } [get_ports { sw1}];
set_property -dict { PACKAGE_PIN r2   IOSTANDARD LVCMOS33 } [get_ports { sw15}];

set_property -dict { PACKAGE_PIN U16   IOSTANDARD LVCMOS33 } [get_ports { ld0}];
set_property -dict { PACKAGE_PIN e19   IOSTANDARD LVCMOS33 } [get_ports { ld1}];
```

3.3.3 Entity und Component

VHDL_main:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity main is
    Port ( sw0 : in STD_LOGIC;
           sw1 : in STD_LOGIC;
           sw15 : in STD_LOGIC;

           ld0 : out STD_LOGIC;
           ld1 : out STD_LOGIC);
end main;

architecture Behavioral of main is

component volladdierer is
port(A1 : in STD_LOGIC;
     A2 : in STD_LOGIC;
     cin : in std_logic;
     Ubertrag : out STD_LOGIC;
     Summe : out STD_LOGIC);
end component;
```

```
begin

voll1:volladdierer port map(A1=>sw0,A2=>sw1,cin=>sw15,Ubertrag=>ld0,Summe=>ld1);

end Behavioral;
```

Contraints:

```
set_property -dict { PACKAGE_PIN V17   IOSTANDARD LVCMOS33 } [get_ports { sw0}];
set_property -dict { PACKAGE_PIN V16   IOSTANDARD LVCMOS33 } [get_ports { sw1}];
set_property -dict { PACKAGE_PIN r2   IOSTANDARD LVCMOS33 } [get_ports { sw15}];

set_property -dict { PACKAGE_PIN U16   IOSTANDARD LVCMOS33 } [get_ports { ld0}];
set_property -dict { PACKAGE_PIN e19   IOSTANDARD LVCMOS33 } [get_ports { ld1}];
```

Volladdierer:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity volladdierer is
    Port ( A1 : in STD_LOGIC;
           A2 : in STD_LOGIC;
           cin : in STD_LOGIC;
           Ubertrag : out STD_LOGIC;
           Summe : out STD_LOGIC);
end volladdierer;

architecture Behavioral of volladdierer is

Summe <= (not cin and not A1 and A2) or (not cin and A1 and not A2)
or (cin and not A1 and not A2) or (cin and A1 and A2);
Ubertrag <= (not cin and A1 and A2) or (cin and not A1 and A2)
or (cin and A1 and not A2) or (cin and A1 and A2);

end Behavioral;
```

3.3.4 Dateiausgliederung

1., 2., 3.:
Die Lösung für den Code ist identisch mit dem aus der vorherigen Aufgabe, jedoch ist die Projektstruktur jetzt anders.
4.:
Die Definition eines Moduls muss im Code vor der Instanzierung des Moduls stehen.

3.3.5 Signale

VHDL_main:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity main is
    Port ( sw0 : in STD_LOGIC;
           sw1 : in STD_LOGIC;
           sw15 : in STD_LOGIC;

           ld0 : out STD_LOGIC;
           ld1 : out STD_LOGIC);
end main;

architecture Behavioral of main is

component volladdierer is
port(A1 : in STD_LOGIC;
     A2 : in STD_LOGIC;
     cin : in std_logic;
     Ubertrag : out STD_LOGIC;
     Summe : out STD_LOGIC);
end component;

begin

voll1:volladdierer port map(A1=>sw0,A2=>sw1,cin=>sw15,Ubertrag=>ld0,Summe=>ld1);

end Behavioral;
```

Contraints:

```
set_property -dict { PACKAGE_PIN V17   IOSTANDARD LVCMOS33 } [get_ports { sw0}];
set_property -dict { PACKAGE_PIN V16   IOSTANDARD LVCMOS33 } [get_ports { sw1}];
set_property -dict { PACKAGE_PIN r2   IOSTANDARD LVCMOS33 } [get_ports { sw15}];

set_property -dict { PACKAGE_PIN U16   IOSTANDARD LVCMOS33 } [get_ports { ld0}];
set_property -dict { PACKAGE_PIN e19   IOSTANDARD LVCMOS33 } [get_ports { ld1}];
```

Volladdierer:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity volladdierer is
    Port ( A1 : in STD_LOGIC;
           A2 : in STD_LOGIC;
           cin : in STD_LOGIC;
           Ubertrag : out STD_LOGIC;
           Summe : out STD_LOGIC);
end volladdierer;

architecture Behavioral of volladdierer is
```

```
signal wire1,wire2,wire3 : std_logic;

begin
wire1 <= A1 xor A2;
wire2 <= wire1 and cin;
wire3 <= A1 and A2;

Summe <= wire1 xor cin;
Ubertrag <= wire2 or wire3;

end Behavioral;
```

3.3.6 Verbindung von Komponenten

VHDL_main:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity main is
    Port ( sw0 : in STD_LOGIC;
           sw1 : in STD_LOGIC;
           sw2 : in STD_LOGIC;
           sw3 : in STD_LOGIC;
           sw4 : in STD_LOGIC;
           sw5 : in STD_LOGIC;
           sw6 : in STD_LOGIC;
           sw7 : in STD_LOGIC;
           sw8 : in STD_LOGIC;
           sw9 : in STD_LOGIC;
           sw10 : in STD_LOGIC;
           sw11 : in STD_LOGIC;
           sw12 : in STD_LOGIC;
           sw13 : in STD_LOGIC;
           sw14 : in STD_LOGIC;
           sw15 : in STD_LOGIC;

           ld0 : out STD_LOGIC;
           ld1 : out STD_LOGIC;
           ld2 : out STD_LOGIC;
           ld3 : out STD_LOGIC;
           ld4 : out STD_LOGIC;
           ld5 : out STD_LOGIC;
           ld6 : out STD_LOGIC;
           ld7 : out STD_LOGIC;
           ld8 : out STD_LOGIC;
           ld9 : out STD_LOGIC;
           ld10 : out STD_LOGIC;
           ld11 : out STD_LOGIC;
           ld12 : out STD_LOGIC;
           ld13 : out STD_LOGIC;
           ld14 : out STD_LOGIC);
```

```
end main;



architecture Behavioral of main is

component volladdierer is
port(A1 : in STD_LOGIC;
     A2 : in STD_LOGIC;
     cin : in std_logic;
     Ubertrag : out STD_LOGIC;
     Summe : out STD_LOGIC);
end component;

signal a1,b1,carry,u1,s1 : std_logic;
signal a2,b2,u2,s2 : std_logic;
signal a3,b3,u3,s3 : std_logic;
signal a4,b4,u4,s4 : std_logic;

begin

carry<=sw0;
a1<=sw1;
a2<=sw2;
a3<=sw3;
a4<=sw4;
b1<=sw5;
b2<=sw6;
b3<=sw7;
b4<=sw8;

voll1:volladdierer port map(A1=>a1,A2=>b1,cin=>carry,Ubertrag=>u1,Summe=>s1);
voll2:volladdierer port map(A1=>a2,A2=>b2,cin=>u1,Ubertrag=>u2,Summe=>s2);
voll3:volladdierer port map(A1=>a3,A2=>b3,cin=>u2,Ubertrag=>u3,Summe=>s3);
voll4:volladdierer port map(A1=>a4,A2=>b4,cin=>u3,Ubertrag=>u4,Summe=>s4);

ld1<=s1;
ld2<=s2;
ld3<=s3;
ld4<=s4;
ld5<=u4;

end Behavioral;
```

Constraints:

```
set_property -dict { PACKAGE_PIN V17   IOSTANDARD LVCMOS33 } [get_ports { sw0}];
set_property -dict { PACKAGE_PIN V16   IOSTANDARD LVCMOS33 } [get_ports { sw1}];
set_property -dict { PACKAGE_PIN w16   IOSTANDARD LVCMOS33 } [get_ports { sw2}];
set_property -dict { PACKAGE_PIN w17   IOSTANDARD LVCMOS33 } [get_ports { sw3}];
set_property -dict { PACKAGE_PIN w15   IOSTANDARD LVCMOS33 } [get_ports { sw4}];
set_property -dict { PACKAGE_PIN v15   IOSTANDARD LVCMOS33 } [get_ports { sw5}];
```

```
set_property -dict { PACKAGE_PIN w14   IOSTANDARD LVCMOS33 } [get_ports { sw6}];
set_property -dict { PACKAGE_PIN w13   IOSTANDARD LVCMOS33 } [get_ports { sw7}];
set_property -dict { PACKAGE_PIN v2   IOSTANDARD LVCMOS33 } [get_ports { sw8}];
set_property -dict { PACKAGE_PIN t3   IOSTANDARD LVCMOS33 } [get_ports { sw9}];
set_property -dict { PACKAGE_PIN t2   IOSTANDARD LVCMOS33 } [get_ports { sw10}];
set_property -dict { PACKAGE_PIN r3   IOSTANDARD LVCMOS33 } [get_ports { sw11}];
set_property -dict { PACKAGE_PIN w2   IOSTANDARD LVCMOS33 } [get_ports { sw12}];
set_property -dict { PACKAGE_PIN u1   IOSTANDARD LVCMOS33 } [get_ports { sw13}];
set_property -dict { PACKAGE_PIN t1   IOSTANDARD LVCMOS33 } [get_ports { sw14}];
set_property -dict { PACKAGE_PIN r2   IOSTANDARD LVCMOS33 } [get_ports { sw15}];

set_property -dict { PACKAGE_PIN U16   IOSTANDARD LVCMOS33 } [get_ports { ld0}];
set_property -dict { PACKAGE_PIN e19   IOSTANDARD LVCMOS33 } [get_ports { ld1}];
set_property -dict { PACKAGE_PIN u19   IOSTANDARD LVCMOS33 } [get_ports { ld2}];
set_property -dict { PACKAGE_PIN v19   IOSTANDARD LVCMOS33 } [get_ports { ld3}];
set_property -dict { PACKAGE_PIN w18   IOSTANDARD LVCMOS33 } [get_ports { ld4}];
set_property -dict { PACKAGE_PIN u15   IOSTANDARD LVCMOS33 } [get_ports { ld5}];
set_property -dict { PACKAGE_PIN u14   IOSTANDARD LVCMOS33 } [get_ports { ld6}];
set_property -dict { PACKAGE_PIN v14   IOSTANDARD LVCMOS33 } [get_ports { ld7}];
set_property -dict { PACKAGE_PIN v13   IOSTANDARD LVCMOS33 } [get_ports { ld8}];
set_property -dict { PACKAGE_PIN v3   IOSTANDARD LVCMOS33 } [get_ports { ld9}];
set_property -dict { PACKAGE_PIN w3   IOSTANDARD LVCMOS33 } [get_ports { ld10}];
set_property -dict { PACKAGE_PIN u3   IOSTANDARD LVCMOS33 } [get_ports { ld11}];
set_property -dict { PACKAGE_PIN p3   IOSTANDARD LVCMOS33 } [get_ports { ld12}];
set_property -dict { PACKAGE_PIN n3   IOSTANDARD LVCMOS33 } [get_ports { ld13}];
set_property -dict { PACKAGE_PIN p1   IOSTANDARD LVCMOS33 } [get_ports { ld14}];
```

3.:
sw0,sw1,sw5,ld1 = 1
sw2,sw6,ld2 = 1
sw3,sw7,ld3 = 1
sw4,sw8,ld4 = 1
4.: Maximal 7 bit, limitierendes Kriterium ist die Anzahl der Schalter.

3.3.7 std_logic_vector

VHDL_main:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity main is
    Port ( sw : in STD_LOGIC_vector(15 downto 0);
           ld :out STD_LOGIC_vector(15 downto 0));
end main;

architecture Behavioral of main is

component volladdierer is
port(A1 : in STD_LOGIC;
     A2 : in STD_LOGIC;
     cin : in std_logic;
```

```
     Ubertrag : out STD_LOGIC;
     Summe : out STD_LOGIC);
end component;

Signal a,b,cin,u,s : std_logic_vector (3 downto 0);

begin

a(0)<=sw(1);
a(1)<=sw(2);
a(2)<=sw(3);
a(3)<=sw(4);
b(0)<=sw(5);
b(1)<=sw(6);
b(2)<=sw(7);
b(3)<=sw(8);

voll1:volladdierer port map(A1=>a(0),A2=>b(0),cin=>sw(0),Ubertrag=>u(0),Summe=>s(0));
voll2:volladdierer port map(A1=>a(1),A2=>b(1),cin=>u(0),Ubertrag=>u(1),Summe=>s(1));
voll3:volladdierer port map(A1=>a(2),A2=>b(2),cin=>u(1),Ubertrag=>u(2),Summe=>s(2));
voll4:volladdierer port map(A1=>a(3),A2=>b(3),cin=>u(2),Ubertrag=>u(3),Summe=>s(3));

ld(1)<=s(0);
ld(2)<=s(1);
ld(3)<=s(2);
ld(4)<=s(3);
ld(5)<=u(3);
end Behavioral;
```

Constraints:

```
set_property -dict { PACKAGE_PIN V17   IOSTANDARD LVCMOS33 } [get_ports  sw[0]];
set_property -dict { PACKAGE_PIN V16   IOSTANDARD LVCMOS33 } [get_ports  sw[1]];
set_property -dict { PACKAGE_PIN w16   IOSTANDARD LVCMOS33 } [get_ports  sw[2]];
set_property -dict { PACKAGE_PIN w17   IOSTANDARD LVCMOS33 } [get_ports  sw[3]];
set_property -dict { PACKAGE_PIN w15   IOSTANDARD LVCMOS33 } [get_ports  sw[4]];
set_property -dict { PACKAGE_PIN v15   IOSTANDARD LVCMOS33 } [get_ports  sw[5]];
set_property -dict { PACKAGE_PIN w14   IOSTANDARD LVCMOS33 } [get_ports  sw[6]];
set_property -dict { PACKAGE_PIN w13   IOSTANDARD LVCMOS33 } [get_ports  sw[7]];
set_property -dict { PACKAGE_PIN v2   IOSTANDARD LVCMOS33 } [get_ports  sw[8]];
set_property -dict { PACKAGE_PIN t3   IOSTANDARD LVCMOS33 } [get_ports  sw[9]];
set_property -dict { PACKAGE_PIN t2   IOSTANDARD LVCMOS33 } [get_ports  sw[10]];
set_property -dict { PACKAGE_PIN r3   IOSTANDARD LVCMOS33 } [get_ports  sw[11]];
set_property -dict { PACKAGE_PIN w2   IOSTANDARD LVCMOS33 } [get_ports  sw[12]];
set_property -dict { PACKAGE_PIN u1   IOSTANDARD LVCMOS33 } [get_ports  sw[13]];
set_property -dict { PACKAGE_PIN t1   IOSTANDARD LVCMOS33 } [get_ports  sw[14]];
set_property -dict { PACKAGE_PIN r2   IOSTANDARD LVCMOS33 } [get_ports  sw[15]];

set_property -dict { PACKAGE_PIN U16   IOSTANDARD LVCMOS33 } [get_ports  ld[0]];
set_property -dict { PACKAGE_PIN e19   IOSTANDARD LVCMOS33 } [get_ports  ld[1]];
set_property -dict { PACKAGE_PIN u19   IOSTANDARD LVCMOS33 } [get_ports  ld[2]];
set_property -dict { PACKAGE_PIN v19   IOSTANDARD LVCMOS33 } [get_ports  ld[3]];
```

```
set_property -dict { PACKAGE_PIN w18   IOSTANDARD LVCMOS33 } [get_ports  ld[4]];
set_property -dict { PACKAGE_PIN u15   IOSTANDARD LVCMOS33 } [get_ports  ld[5]];
set_property -dict { PACKAGE_PIN u14   IOSTANDARD LVCMOS33 } [get_ports  ld[6]];
set_property -dict { PACKAGE_PIN v14   IOSTANDARD LVCMOS33 } [get_ports  ld[7]];
set_property -dict { PACKAGE_PIN v13   IOSTANDARD LVCMOS33 } [get_ports  ld[8]];
set_property -dict { PACKAGE_PIN v3   IOSTANDARD LVCMOS33 } [get_ports  ld[9]];
set_property -dict { PACKAGE_PIN w3   IOSTANDARD LVCMOS33 } [get_ports  ld[10]];
set_property -dict { PACKAGE_PIN u3   IOSTANDARD LVCMOS33 } [get_ports  ld[11]];
set_property -dict { PACKAGE_PIN p3   IOSTANDARD LVCMOS33 } [get_ports  ld[12]];
set_property -dict { PACKAGE_PIN n3   IOSTANDARD LVCMOS33 } [get_ports  ld[13]];
set_property -dict { PACKAGE_PIN p1   IOSTANDARD LVCMOS33 } [get_ports  ld[14]];
set_property -dict { PACKAGE_PIN l1   IOSTANDARD LVCMOS33 } [get_ports  ld[15]];
```

Die Contraints Datei wird erst wieder neu eingetragen, wenn sich am Code etwas ändert.

3.3.8 Generate Statement

VHDL_main:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity main is
    Port ( sw : in STD_LOGIC_vector(15 downto 0);
           ld :out STD_LOGIC_vector(15 downto 0));
end main;

architecture Behavioral of main is

component volladdierer is
port(A1 : in STD_LOGIC;
     A2 : in STD_LOGIC;
     cin : in std_logic;
     Ubertrag : out STD_LOGIC;
     Summe : out STD_LOGIC);
end component;

Signal a,b,cin,u,s : std_logic_vector (3 downto 0);

begin

a(0)<=sw(1);
a(1)<=sw(2);
a(2)<=sw(3);
a(3)<=sw(4);
b(0)<=sw(5);
b(1)<=sw(6);
b(2)<=sw(7);
b(3)<=sw(8);

GENs_voll:
for i in 0 to 3 generate
```

```
vollex : volladdierer port map
(A1=>a(i),A2=>b(i),cin=>cin(i),Ubertrag=>u(i),Summe=>s(i));
end generate GENs_voll;

cin(3)<=u(2);
cin(2)<=u(1);
cin(1)<=u(0);
cin(0)<=sw(0);

ld(1)<=s(0);
ld(2)<=s(1);
ld(3)<=s(2);
ld(4)<=s(3);
ld(5)<=u(3);

end Behavioral;
```

3.4 Process Structure und State Machines

3.4.1 Process Structure

VHDL_main:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity main is
    Port ( sw : in STD_LOGIC_vector(15 downto 0);
           ld :out STD_LOGIC_vector(15 downto 0));
end main;

architecture Behavioral of main is
begin
process(sw)
begin

ld(0)<=sw(0) and sw(1);

end process;

end Behavioral;
```

3.4.2 If Statement

1.:
VHDL_main:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity main is
    Port ( sw : in STD_LOGIC_vector(15 downto 0);
```

```
           ld :out STD_LOGIC_vector(15 downto 0));
end main;

architecture Behavioral of main is

begin

process(sw)
begin

if(sw(1 downto 0)="11" ) then
ld(0)<='1';
else
ld(0)<='0';
end if;

end process;

end Behavioral;
```

2., 3.:
VHDL_main:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx leaf cells in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity main is
    Port ( sw : in STD_LOGIC_vector(15 downto 0);
           ld :out STD_LOGIC_vector(15 downto 0));
end main;

architecture Behavioral of main is

begin

process(sw)
begin

if(sw(1 downto 0)="11" or sw(1 downto 0)="10") then
ld(0)<='1';
else
ld(0)<='0';
end if;
```

```
if(sw(3 downto 2)="11") then
ld(2)<='1';
elsif(sw(3 downto 2)="10")then
ld(2)<='1';
else
ld(2)<='0';
end if;

end process;

end Behavioral;
```

4.: Funktional gleicher Code kann unterschiedliche Hardware instanzieren. Dies kann Auswirkungen auf die Funktion der Schaltung haben. Dies macht sich insbesondere bei der Statemachine Aufgabe in dieser Übung bemerkbar.

3.4.3 Case When Statement

1.:
VHDl_main:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity main is
    Port ( sw : in STD_LOGIC_vector(15 downto 0);
           ld :out STD_LOGIC_vector(15 downto 0));
end main;

architecture Behavioral of main is

begin

process(sw)
begin

case sw(1 downto 0) is
when "11" =>
ld(0)<='1';
when "10" =>
ld(0)<='1';
when others =>
ld(0)<='0';
end case;

end process;

end Behavioral;
```

3.:
VHDL_main:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity main is
    Port ( sw : in STD_LOGIC_vector(15 downto 0);
           ld :out STD_LOGIC_vector(15 downto 0));
end main;

architecture Behavioral of main is

begin

process(sw)
begin

case sw(1 downto 0) is
when "10" =>
ld(0)<='1';
when "01" =>
ld(0)<='1';
when others =>
ld(0)<='0';
end case;

if(sw(3 downto 2)="10" or sw(3 downto 2)="01") then
ld(2)<='1';
else
ld(2)<='0';
end if;

if(sw(5 downto 4)="10") then
ld(4)<='1';
elsif(sw(5 downto 4)="01") then
ld(4)<='1';
else
ld(4)<='0';
end if;

ld(6)<=sw(6) xor sw(7);

end process;

end Behavioral;
```

3.4.4 State Machines

1.:
VHDL_main:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
```

```
entity main is
    Port ( sw : in STD_LOGIC_vector(15 downto 0);
           btnd : in std_logic;
           ld :out STD_LOGIC_vector(15 downto 0));
end main;

architecture Behavioral of main is

Signal zustand : std_logic_vector (1 downto 0) := "00";

begin

process(zustand,sw,btnd)
begin

case zustand is
when "00" =>
ld(0)<='1';
ld(1)<='1';
ld(2)<='0';
ld(3)<='0';
ld(4)<='0';
ld(5)<='0';
ld(7)<='0';
ld(10)<='0';
ld(11)<='0';
ld(12)<='0';
case sw is
when "0001" =>
zustand(0)<='1';
when others =>

end case;

when "01" =>
ld(0)<='0';
ld(1)<='0';
ld(2)<='1';
ld(3)<='1';
ld(4)<='1';
ld(5)<='0';
ld(7)<='0';
ld(10)<='0';
ld(11)<='0';
ld(12)<='0';
case sw is
when "0010" =>
zustand(1)<='1';
when others =>

end case;
```

```
when "11" =>
ld(0)<='0';
ld(1)<='0';
ld(2)<='0';
ld(3)<='0';
ld(4)<='0';
ld(5)<='1';
ld(7)<='1';
ld(10)<='0';
ld(11)<='0';
ld(12)<='0';
case sw is
when "0100" =>
zustand(0)<='0';
when others =>

end case;

when "10" =>
ld(0)<='0';
ld(1)<='0';
ld(2)<='0';
ld(3)<='0';
ld(4)<='0';
ld(5)<='0';
ld(7)<='0';
ld(10)<='1';
ld(11)<='1';
ld(12)<='1';
case sw is
when "1000" =>
zustand(1)<='0';
when others =>

end case;
end case;
end process;
end Behavioral;
```

2.:
Die Umschaltung zwischen den einzelnen Zuständen funktioniert nur schlecht. Die Schaltung ist im Zustandswechsel nicht eindeutig festgelegt. Dies lässt sich in der nächsten Übung mit den clock-Statements lösen.
3.:
VHDL_main:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity main is
    Port ( sw : in STD_LOGIC_vector(15 downto 0);
           ld :out STD_LOGIC_vector(15 downto 0));
```

```
end main;

architecture Behavioral of main is

component volladdierer is
port(A1 : in STD_LOGIC;
     A2 : in STD_LOGIC;
     cin : in std_logic;
     Ubertrag : out STD_LOGIC;
     Summe : out STD_LOGIC);
end component;

Signal a,b,cin,u,s : std_logic_vector (3 downto 0);

begin

a(0)<=sw(1);
a(1)<=sw(2);
a(2)<=sw(3);
a(3)<=sw(4);
b(0)<=sw(5);
b(1)<=sw(6);
b(2)<=sw(7);
b(3)<=sw(8);

GENs_voll:
for i in 0 to 3 generate
voll1:
if (i=0) generate
full0 : volladdierer port map
(A1=>a(i),A2=>b(i),cin=>sw(0),Ubertrag=>u(i),Summe=>s(i));
end generate;
voll2:
if(i/=0) generate
full1 : volladdierer port map
(A1=>a(i),A2=>b(i),cin=>u(i-1),Ubertrag=>u(i),Summe=>s(i));
end generate;
end generate;

ld(1)<=s(0);
ld(2)<=s(1);
ld(3)<=s(2);
ld(4)<=s(3);
ld(5)<=u(3);

end Behavioral;
```

3.5 Clock Instantiation

3.5.1 Einlesen eines Frequenzsignals

1.:
VHDL_main:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_unsigned.ALL;
use IEEE.numeric_std.ALL;

entity main is
    Port ( sw : in STD_LOGIC_vector(15 downto 0);
           uhr : in std_logic;
           ld :out STD_LOGIC_vector(15 downto 0));
end main;

architecture Behavioral of main is

Signal zustand : std_logic_vector (1 downto 0):="00";
Signal teiler : std_logic_vector (32 downto 0);

begin

process(uhr,teiler)
begin
if(rising_edge(uhr)) then
teiler <= teiler+1;
end if;
end process;

process(zustand,sw,uhr,teiler)
begin
if(rising_edge(uhr)) then
case zustand is
when "00" =>
if (sw(0)='1') then
zustand(0)<='1';
end if;
when "01" =>
if (sw(1)='1') then
zustand(1)<='1';
end if;
when "11" =>
if (sw(2)='1') then
zustand(0)<='0';
end if;
when "10" =>
if (sw(3)='1') then
zustand(1)<='0';
end if;
```

```
end case;
end if;
end process;

process(zustand)
begin
case zustand is
when "00" =>
ld(0)<='1';
ld(1)<='1';
ld(2)<='0';
ld(3)<='0';
ld(4)<='0';
ld(5)<='0';
ld(7)<='0';
ld(10)<='0';
ld(11)<='0';
ld(12)<='0';
when "01" =>
ld(0)<='0';
ld(1)<='0';
ld(2)<='1';
ld(3)<='1';
ld(4)<='1';
ld(5)<='0';
ld(7)<='0';
ld(10)<='0';
ld(11)<='0';
ld(12)<='0';
when "11" =>
ld(0)<='0';
ld(1)<='0';
ld(2)<='0';
ld(3)<='0';
ld(4)<='0';
ld(5)<='1';
ld(7)<='1';
ld(10)<='0';
ld(11)<='0';
ld(12)<='0';
when "10" =>
ld(0)<='0';
ld(1)<='0';
ld(2)<='0';
ld(3)<='0';
ld(4)<='0';
ld(5)<='0';
ld(7)<='0';
ld(10)<='1';
ld(11)<='1';
ld(12)<='1';
end case;
```

```
end process;
end Behavioral;
```

Contraints:

```
set_property -dict { PACKAGE_PIN V17   IOSTANDARD LVCMOS33 } [get_ports  sw[0]];
set_property -dict { PACKAGE_PIN V16   IOSTANDARD LVCMOS33 } [get_ports  sw[1]];
set_property -dict { PACKAGE_PIN w16   IOSTANDARD LVCMOS33 } [get_ports  sw[2]];
set_property -dict { PACKAGE_PIN w17   IOSTANDARD LVCMOS33 } [get_ports  sw[3]];
set_property -dict { PACKAGE_PIN w15   IOSTANDARD LVCMOS33 } [get_ports  sw[4]];
set_property -dict { PACKAGE_PIN v15   IOSTANDARD LVCMOS33 } [get_ports  sw[5]];
set_property -dict { PACKAGE_PIN w14   IOSTANDARD LVCMOS33 } [get_ports  sw[6]];
set_property -dict { PACKAGE_PIN w13   IOSTANDARD LVCMOS33 } [get_ports  sw[7]];
set_property -dict { PACKAGE_PIN v2   IOSTANDARD LVCMOS33 } [get_ports  sw[8]];
set_property -dict { PACKAGE_PIN t3   IOSTANDARD LVCMOS33 } [get_ports  sw[9]];
set_property -dict { PACKAGE_PIN t2   IOSTANDARD LVCMOS33 } [get_ports  sw[10]];
set_property -dict { PACKAGE_PIN r3   IOSTANDARD LVCMOS33 } [get_ports  sw[11]];
set_property -dict { PACKAGE_PIN w2   IOSTANDARD LVCMOS33 } [get_ports  sw[12]];
set_property -dict { PACKAGE_PIN u1   IOSTANDARD LVCMOS33 } [get_ports  sw[13]];
set_property -dict { PACKAGE_PIN t1   IOSTANDARD LVCMOS33 } [get_ports  sw[14]];
set_property -dict { PACKAGE_PIN r2   IOSTANDARD LVCMOS33 } [get_ports  sw[15]];

set_property -dict { PACKAGE_PIN U16   IOSTANDARD LVCMOS33 } [get_ports  ld[0]];
set_property -dict { PACKAGE_PIN e19   IOSTANDARD LVCMOS33 } [get_ports  ld[1]];
set_property -dict { PACKAGE_PIN u19   IOSTANDARD LVCMOS33 } [get_ports  ld[2]];
set_property -dict { PACKAGE_PIN v19   IOSTANDARD LVCMOS33 } [get_ports  ld[3]];
set_property -dict { PACKAGE_PIN w18   IOSTANDARD LVCMOS33 } [get_ports  ld[4]];
set_property -dict { PACKAGE_PIN u15   IOSTANDARD LVCMOS33 } [get_ports  ld[5]];
set_property -dict { PACKAGE_PIN u14   IOSTANDARD LVCMOS33 } [get_ports  ld[6]];
set_property -dict { PACKAGE_PIN v14   IOSTANDARD LVCMOS33 } [get_ports  ld[7]];
set_property -dict { PACKAGE_PIN v13   IOSTANDARD LVCMOS33 } [get_ports  ld[8]];
set_property -dict { PACKAGE_PIN v3   IOSTANDARD LVCMOS33 } [get_ports  ld[9]];
set_property -dict { PACKAGE_PIN w3   IOSTANDARD LVCMOS33 } [get_ports  ld[10]];
set_property -dict { PACKAGE_PIN u3   IOSTANDARD LVCMOS33 } [get_ports  ld[11]];
set_property -dict { PACKAGE_PIN p3   IOSTANDARD LVCMOS33 } [get_ports  ld[12]];
set_property -dict { PACKAGE_PIN n3   IOSTANDARD LVCMOS33 } [get_ports  ld[13]];
set_property -dict { PACKAGE_PIN p1   IOSTANDARD LVCMOS33 } [get_ports  ld[14]];
set_property -dict { PACKAGE_PIN l1   IOSTANDARD LVCMOS33 } [get_ports  ld[15]];

set_property -dict { PACKAGE_PIN w5   IOSTANDARD LVCMOS33 } [get_ports  uhr];
```

3.5.2 Frequenzteiler

1.:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.std_logic_unsigned.all;
use IEEE.numeric_std.all;

entity main is
    Port ( sw : in STD_LOGIC_vector(15 downto 0);
```

```
           uhr : in std_logic;
           ld :out STD_LOGIC_vector(15 downto 0));
end main;

architecture Behavioral of main is

Signal teiler : std_logic_vector (32 downto 0);

begin

process(uhr,teiler)
begin
if(rising_edge(uhr)) then
teiler<=teiler+1;
end if;

ld(14)<= teiler(26);

end process;
end Behavioral;
```

Exakte Frequenz ist 100MHz/2n̂ (n=26,27)
2.:
VHDL_main:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.std_logic_unsigned.all;
use IEEE.numeric_std.all;

entity main is
    Port ( sw : in STD_LOGIC_vector(15 downto 0);
           uhr : in std_logic;
           ld :out STD_LOGIC_vector(15 downto 0));
end main;

architecture Behavioral of main is

Signal teiler : std_logic_vector (26 downto 0):="10111110101111000010000000";
signal a : std_Logic;

begin

process(uhr,teiler)
begin
if(rising_edge(uhr)) then
teiler<=teiler-1;
if(teiler="000000000000000000000000000") then
teiler<="10111110101111000010000000";
a<=not a;
end if;
```

```
end if;
end process;

ld(0)<=a;

end Behavioral;
```

3.:
VHDL_main:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.std_logic_unsigned.all;
use IEEE.numeric_std.all;

entity main is
    Port ( sw : in STD_LOGIC_vector(15 downto 0);
           uhr : in std_logic;
           ld :out STD_LOGIC_vector(15 downto 0));
end main;

architecture Behavioral of main is

Signal teiler : std_logic_vector (1 downto 0) :="11";

begin

process(uhr,teiler)
begin
if(rising_edge(uhr)) then
teiler<=teiler-1;
if(teiler="00") then
ld(0)<='1';
else
ld(0)<='0';
end if;
end if;

end process;
end Behavioral;
```

3.5.3 Synchrones und Asynchrones Design

1.:

Synchron vs Asynchron		
L	Synchron	Asynchron
Vorteile	Einsparung von FFs, kleinere Chips möglich	Reset wird schneller ausgeführt
Nachteile	Reset kann verzögert sein. Mögliche Metastabilität bei Assertierung des Reset-Signals	Mögliche Metastabilität bei Deassertierung des Reset-Signals

Table 6: Synchron vs Asynchron

[2]
2.:
VHDL_main:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.std_logic_unsigned.all;
use IEEE.numeric_std.all;

entity main is
    Port ( sw : in STD_LOGIC_vector(15 downto 0);
           uhr : in std_logic;
           ld :out STD_LOGIC_vector(15 downto 0));
end main;

architecture Behavioral of main is

signal reset1,reset2 : std_logic :='0';
signal temp1,temp2 : std_logic_vector(7 downto 0) :="00000000";

begin

reset1<=sw(0);
reset2<=sw(1);

process(uhr)
begin

if(rising_edge(uhr)) then
if(reset1='1') then
temp1<="00000000";
else
temp1<=temp1+1;
end if;
end if;

if(reset2='1') then
temp2<="00000000";
elsif(rising_edge(uhr)) then
temp2<=temp2+1;
end if;

end process;
ld(7 downto 0)<=temp1;
ld(15 downto 8)<=temp2;
end Behavioral;
```

3.:
VHDL_main:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_unsigned.ALL;
use IEEE.numeric_std.ALL;

entity main is
    Port ( sw : in STD_LOGIC_vector(15 downto 0);
           uhr : in std_logic;
           btnd : in std_logic;
           ld :out STD_LOGIC_vector(15 downto 0));
end main;

architecture Behavioral of main is

Signal zustand : std_logic_vector (2 downto 0):="000";
Signal teiler : std_logic_vector (32 downto 0);

begin

process(zustand,sw,uhr,teiler)
begin
if(rising_edge(uhr)) then
if(btnd='1') then
zustand<="000";
else
case zustand is
when "000" =>
if (sw="000001") then
zustand<="001";
end if;

when "001" =>
if (sw="000010") then
zustand<="010";
end if;

when "010" =>
if (sw="000100") then
zustand<="011";
end if;

when "011" =>
if (sw="001000") then
zustand<="100";
end if;

when "100" =>
if (sw="010000") then
zustand<="101";
end if;
```

```
when "101" =>
if (sw="100000") then
zustand<="000";
end if;
when others=>
zustand<="000";
end case;
end if;
end if;
end process;

process(zustand)
begin
case zustand is
when "000" =>
ld(0)<='1';
ld(1)<='1';
ld(2)<='0';
ld(3)<='0';
ld(4)<='0';
ld(5)<='0';
ld(6)<='0';
ld(7)<='0';
ld(8)<='0';
ld(9)<='0';
ld(10)<='0';
ld(11)<='0';
when "001" =>
ld(0)<='0';
ld(1)<='0';
ld(2)<='1';
ld(3)<='1';
ld(4)<='0';
ld(5)<='0';
ld(6)<='0';
ld(7)<='0';
ld(8)<='0';
ld(9)<='0';
ld(10)<='0';
ld(11)<='0';
when "010" =>
ld(0)<='0';
ld(1)<='0';
ld(2)<='0';
ld(3)<='0';
ld(4)<='1';
ld(5)<='1';
ld(6)<='0';
ld(7)<='0';
ld(8)<='0';
ld(9)<='0';
```

```
ld(10)<='0';
ld(11)<='0';
when "011" =>
ld(0)<='0';
ld(1)<='0';
ld(2)<='0';
ld(3)<='0';
ld(4)<='0';
ld(5)<='0';
ld(6)<='1';
ld(7)<='1';
ld(8)<='0';
ld(9)<='0';
ld(10)<='0';
ld(11)<='0';
when "100" =>
ld(0)<='0';
ld(1)<='0';
ld(2)<='0';
ld(3)<='0';
ld(4)<='0';
ld(5)<='0';
ld(6)<='0';
ld(7)<='0';
ld(8)<='1';
ld(9)<='1';
ld(10)<='0';
ld(11)<='0';
when "101" =>
ld(0)<='0';
ld(1)<='0';
ld(2)<='0';
ld(3)<='0';
ld(4)<='0';
ld(5)<='0';
ld(6)<='0';
ld(7)<='0';
ld(8)<='0';
ld(9)<='0';
ld(10)<='1';
ld(11)<='1';
when others=>
ld(0)<='1';
ld(1)<='0';
ld(2)<='1';
ld(3)<='0';
ld(4)<='1';
ld(5)<='0';
ld(6)<='1';
ld(7)<='0';
ld(8)<='1';
ld(9)<='0';
```

```
ld(10)<='1';
ld(11)<='0';
end case;
end process;
end Behavioral;
```

Contraints:

```
set_property -dict { PACKAGE_PIN V17   IOSTANDARD LVCMOS33 } [get_ports  sw[0]];
set_property -dict { PACKAGE_PIN V16   IOSTANDARD LVCMOS33 } [get_ports  sw[1]];
set_property -dict { PACKAGE_PIN w16   IOSTANDARD LVCMOS33 } [get_ports  sw[2]];
set_property -dict { PACKAGE_PIN w17   IOSTANDARD LVCMOS33 } [get_ports  sw[3]];
set_property -dict { PACKAGE_PIN w15   IOSTANDARD LVCMOS33 } [get_ports  sw[4]];
set_property -dict { PACKAGE_PIN v15   IOSTANDARD LVCMOS33 } [get_ports  sw[5]];
set_property -dict { PACKAGE_PIN w14   IOSTANDARD LVCMOS33 } [get_ports  sw[6]];
set_property -dict { PACKAGE_PIN w13   IOSTANDARD LVCMOS33 } [get_ports  sw[7]];
set_property -dict { PACKAGE_PIN v2   IOSTANDARD LVCMOS33 } [get_ports  sw[8]];
set_property -dict { PACKAGE_PIN t3   IOSTANDARD LVCMOS33 } [get_ports  sw[9]];
set_property -dict { PACKAGE_PIN t2   IOSTANDARD LVCMOS33 } [get_ports  sw[10]];
set_property -dict { PACKAGE_PIN r3   IOSTANDARD LVCMOS33 } [get_ports  sw[11]];
set_property -dict { PACKAGE_PIN w2   IOSTANDARD LVCMOS33 } [get_ports  sw[12]];
set_property -dict { PACKAGE_PIN u1   IOSTANDARD LVCMOS33 } [get_ports  sw[13]];
set_property -dict { PACKAGE_PIN t1   IOSTANDARD LVCMOS33 } [get_ports  sw[14]];
set_property -dict { PACKAGE_PIN r2   IOSTANDARD LVCMOS33 } [get_ports  sw[15]];

set_property -dict { PACKAGE_PIN U16   IOSTANDARD LVCMOS33 } [get_ports  ld[0]];
set_property -dict { PACKAGE_PIN e19   IOSTANDARD LVCMOS33 } [get_ports  ld[1]];
set_property -dict { PACKAGE_PIN u19   IOSTANDARD LVCMOS33 } [get_ports  ld[2]];
set_property -dict { PACKAGE_PIN v19   IOSTANDARD LVCMOS33 } [get_ports  ld[3]];
set_property -dict { PACKAGE_PIN w18   IOSTANDARD LVCMOS33 } [get_ports  ld[4]];
set_property -dict { PACKAGE_PIN u15   IOSTANDARD LVCMOS33 } [get_ports  ld[5]];
set_property -dict { PACKAGE_PIN u14   IOSTANDARD LVCMOS33 } [get_ports  ld[6]];
set_property -dict { PACKAGE_PIN v14   IOSTANDARD LVCMOS33 } [get_ports  ld[7]];
set_property -dict { PACKAGE_PIN v13   IOSTANDARD LVCMOS33 } [get_ports  ld[8]];
set_property -dict { PACKAGE_PIN v3   IOSTANDARD LVCMOS33 } [get_ports  ld[9]];
set_property -dict { PACKAGE_PIN w3   IOSTANDARD LVCMOS33 } [get_ports  ld[10]];
set_property -dict { PACKAGE_PIN u3   IOSTANDARD LVCMOS33 } [get_ports  ld[11]];
set_property -dict { PACKAGE_PIN p3   IOSTANDARD LVCMOS33 } [get_ports  ld[12]];
set_property -dict { PACKAGE_PIN n3   IOSTANDARD LVCMOS33 } [get_ports  ld[13]];
set_property -dict { PACKAGE_PIN p1   IOSTANDARD LVCMOS33 } [get_ports  ld[14]];
set_property -dict { PACKAGE_PIN l1   IOSTANDARD LVCMOS33 } [get_ports  ld[15]];

set_property -dict { PACKAGE_PIN w5   IOSTANDARD LVCMOS33 } [get_ports  uhr];
set_property -dict { PACKAGE_PIN u17   IOSTANDARD LVCMOS33 } [get_ports  btnd];
```

4.:
VHDL_main:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.std_logic_unsigned.all;
use IEEE.numeric_std.all;
```

```
entity main is
    Port ( sw : in STD_LOGIC_vector(15 downto 0);
           uhr : in std_logic;
           ld :out STD_LOGIC_vector(15 downto 0));
end main;

architecture Behavioral of main is

Signal teiler : std_logic_vector (23 downto 0):="100110001001011010000000";
signal start,stop,reset : std_Logic :='0';
signal stopuhr : std_logic_vector(15 downto 0):="0000000000000000";

begin

process(uhr,teiler,stopuhr,start,stop)
begin
if(rising_edge(uhr)) then
if(reset='1') then
reset<='0';
teiler<="100110001001011010000000";
stopuhr<="0000000000000000";
start<='0';
stop<='0';
else
if(start='1' and stop='0') then
if(teiler="000000000000000000000000") then
teiler<="100110001001011010000000";
stopuhr<=stopuhr+1;
else
teiler<=teiler-1;
end if;
end if;
end if;
end if;

if(rising_edge(uhr)) then
if(sw(0)='1') then
start<='1';
end if;
end if;

if(rising_edge(uhr)) then
if(sw(1)='1') then
stop<='1';
end if;
end if;

if(rising_edge(uhr)) then
if(sw(2)='1') then
reset<='1';
end if;
```

```
end if;

end process;

ld<=stopuhr;

end Behavioral;
```

Maximale Auflösung bei einer 100MHz Uhr: 10^-8. Durch Oversampling oder das zusätzliche Abfragen von fallenden Taktflanken, lässt sich die Auflösung jedoch erhöhen. Die maximale Auflösung entpricht dann der Gatterlaufzeit der D-FF.
Das Verwenden eines Ringoszillators mit einer Frequenz>100MHz erhöht ebenfalls die Auflösung. Kompliziertere Methoden wie Vernier-Delay-Lines sollten noch nicht versucht werden, mit diesen ist es jedoch möglich, die Zeitauflösung in den Pikosenkundenbereich zu drücken, also deutlich unterhalb der D-FF-Gatterlaufzeiten.
Maximale Zeit, die gemessen werden kann: ((2^17)-1)*0.1s

3.5.4 Gatterlaufzeiten

1.:
VHDL_main:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.std_logic_unsigned.all;
use IEEE.numeric_std.all;

entity main is
    Port ( sw : in STD_LOGIC_vector(15 downto 0);
           ld :out STD_LOGIC_vector(15 downto 0));
end main;

architecture Behavioral of main is

component Inverter is
    Port ( ein : in STD_LOGIC;
           aus : out STD_LOGIC);
end component;

Signal a,b : std_logic_vector (4 downto 0);
signal teiler : std_logic_vector (31 downto 0);

begin

GEN_INV:
for i in 0 to 4 generate
inve0:
if (i=0) generate
inv1 : Inverter port map
(ein=>b(4),aus=>b(i));
end generate;
inve1:
```

```
if(i/=0) generate
inv1 : Inverter port map
(ein=>b(i-1),aus=>b(i));
end generate;
end generate;

process(b,teiler)
begin

if(rising_edge(b(0))) then
teiler<=teiler+1;
end if;
ld(12)<=teiler(29);

end process;

end Behavioral;
```

Constraints:

```
set_property -dict { PACKAGE_PIN V17   IOSTANDARD LVCMOS33 } [get_ports  sw[0]];
set_property -dict { PACKAGE_PIN V16   IOSTANDARD LVCMOS33 } [get_ports  sw[1]];
set_property -dict { PACKAGE_PIN w16   IOSTANDARD LVCMOS33 } [get_ports  sw[2]];
set_property -dict { PACKAGE_PIN w17   IOSTANDARD LVCMOS33 } [get_ports  sw[3]];
set_property -dict { PACKAGE_PIN w15   IOSTANDARD LVCMOS33 } [get_ports  sw[4]];
set_property -dict { PACKAGE_PIN v15   IOSTANDARD LVCMOS33 } [get_ports  sw[5]];
set_property -dict { PACKAGE_PIN w14   IOSTANDARD LVCMOS33 } [get_ports  sw[6]];
set_property -dict { PACKAGE_PIN w13   IOSTANDARD LVCMOS33 } [get_ports  sw[7]];
set_property -dict { PACKAGE_PIN v2   IOSTANDARD LVCMOS33 } [get_ports  sw[8]];
set_property -dict { PACKAGE_PIN t3   IOSTANDARD LVCMOS33 } [get_ports  sw[9]];
set_property -dict { PACKAGE_PIN t2   IOSTANDARD LVCMOS33 } [get_ports  sw[10]];
set_property -dict { PACKAGE_PIN r3   IOSTANDARD LVCMOS33 } [get_ports  sw[11]];
set_property -dict { PACKAGE_PIN w2   IOSTANDARD LVCMOS33 } [get_ports  sw[12]];
set_property -dict { PACKAGE_PIN u1   IOSTANDARD LVCMOS33 } [get_ports  sw[13]];
set_property -dict { PACKAGE_PIN t1   IOSTANDARD LVCMOS33 } [get_ports  sw[14]];
set_property -dict { PACKAGE_PIN r2   IOSTANDARD LVCMOS33 } [get_ports  sw[15]];

set_property -dict { PACKAGE_PIN U16   IOSTANDARD LVCMOS33 } [get_ports  ld[0]];
set_property -dict { PACKAGE_PIN e19   IOSTANDARD LVCMOS33 } [get_ports  ld[1]];
set_property -dict { PACKAGE_PIN u19   IOSTANDARD LVCMOS33 } [get_ports  ld[2]];
set_property -dict { PACKAGE_PIN v19   IOSTANDARD LVCMOS33 } [get_ports  ld[3]];
set_property -dict { PACKAGE_PIN w18   IOSTANDARD LVCMOS33 } [get_ports  ld[4]];
set_property -dict { PACKAGE_PIN u15   IOSTANDARD LVCMOS33 } [get_ports  ld[5]];
set_property -dict { PACKAGE_PIN u14   IOSTANDARD LVCMOS33 } [get_ports  ld[6]];
set_property -dict { PACKAGE_PIN v14   IOSTANDARD LVCMOS33 } [get_ports  ld[7]];
set_property -dict { PACKAGE_PIN v13   IOSTANDARD LVCMOS33 } [get_ports  ld[8]];
set_property -dict { PACKAGE_PIN v3   IOSTANDARD LVCMOS33 } [get_ports  ld[9]];
set_property -dict { PACKAGE_PIN w3   IOSTANDARD LVCMOS33 } [get_ports  ld[10]];
set_property -dict { PACKAGE_PIN u3   IOSTANDARD LVCMOS33 } [get_ports  ld[11]];
set_property -dict { PACKAGE_PIN p3   IOSTANDARD LVCMOS33 } [get_ports  ld[12]];
set_property -dict { PACKAGE_PIN n3   IOSTANDARD LVCMOS33 } [get_ports  ld[13]];
set_property -dict { PACKAGE_PIN p1   IOSTANDARD LVCMOS33 } [get_ports  ld[14]];
```

```
set_property -dict { PACKAGE_PIN l1   IOSTANDARD LVCMOS33 } [get_ports  ld[15]];

set_property ALLOW_COMBINATORIAL_LOOPS true [get_nets -of_objects [get_cells ld_OBUF[0]_inst_i_1]]

set_property SEVERITY {Warning}  [get_drc_checks LUTLP-1]

set_property SEVERITY {Warning} [get_drc_checks NSTD-1]
```

Inverter:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity Inverter is
    Port ( ein : in STD_LOGIC;
           aus : out STD_LOGIC);
end Inverter;

architecture Behavioral of Inverter is

begin

aus <= not ein;

end Behavioral;
```

Die Zeit, die die LED leuchtet x2, ist die Ausgangstaktperiode des Frequenzteilers. Es können natürlich auch mehrere Taktperioden gemessen werden und dann die durchschnittliche Zeit berechnet werden. Die Ausgangsfrequenz des Ringoszillators ergibt sich dann, indem man die Frequenz der blinkenden LED mit dem Teilerwert multipliziert. Aus der so berechneten Frequenz lässt sich die Periodendauer des Ringoszillators berechnen. Diese Zeit entspricht der Zeit, die das Signal braucht, um einmal durch alle Gatter zu laufen. Teilt man diese Zeit dann durch die Anzahl der verwendeten Gatter, erhält man grob die Durchlaufzeit eines einzelnen Gatters, wenn man die Verzögerungszeiten zwischen den Gattern vernachlässigt.
Ergebnis ungefähr: 1ns

3.5.5 Vivado Clock Manager

VHDL_main:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.std_logic_unsigned.all;
use IEEE.numeric_std.all;

entity main is
    Port ( sw : in STD_LOGIC_vector(15 downto 0);
           uhr : in std_logic;
           ld :out STD_LOGIC_vector(15 downto 0));
end main;

architecture Behavioral of main is
```

```
signal temp : std_logic_vector(15 downto 0) :="1111111111111111";
signal teiler : std_logic_vector(28 downto 0) :="10011000100101101000000000000";
signal schnelluhr,resets,locks : std_logic :='0';

component clk_wiz_0 is
port
(clk_out1 : out std_logic;
reset : in std_logic;
locked : out std_logic;
clk_in1 : in std_logic);
end component;

begin

UHR1: clk_wiz_0 port map(clk_out1=>schnelluhr,reset=>resets,locked=>locks,clk_in1=>uhr);

process(schnelluhr)
begin
if(rising_edge(schnelluhr)) then
if(teiler="00000000000000000000000000000") then
teiler<="10011000100101101000000000000";
temp<=temp+1;
else
teiler<=teiler-1;
end if;
end if;
end process;

ld<=temp;
end Behavioral;
```

3.6 Arithmetik und Variablen

3.6.1 ieee.numeric_std.all Bibliothek

1.:
Vergleich abhängig von der Implementierung.
2.:
VHDL_main:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.std_logic_unsigned.all;
use IEEE.numeric_std.all;

entity main is
    Port ( sw : in STD_LOGIC_vector(15 downto 0);
            uhr : in std_logic;
            btnd : in std_logic;
           ld :out STD_LOGIC_vector(15 downto 0));
end main;
```

```
architecture Behavioral of main is

Signal a,b,mult: std_logic_vector (7 downto 0);
Signal c: std_logic_vector (15 downto 0):="0000000000000000";
signal reset : std_logic :='0';

begin

a<=sw(15 downto 8);
b<=sw(7 downto 0);


process(uhr)
begin
reset<=btnd;
if(rising_edge(uhr)) then
if(reset = '1') then
reset <= '0';
c<="0000000000000000";
mult<=b;
elsif(mult /= "00000000") then
c<=c+a;
mult<=mult-1;
end if;
end if;
end process;

ld<=c;

end Behavioral;
```

Constraints:

```
set_property -dict { PACKAGE_PIN V17   IOSTANDARD LVCMOS33 } [get_ports  sw[0]];
set_property -dict { PACKAGE_PIN V16   IOSTANDARD LVCMOS33 } [get_ports  sw[1]];
set_property -dict { PACKAGE_PIN w16   IOSTANDARD LVCMOS33 } [get_ports  sw[2]];
set_property -dict { PACKAGE_PIN w17   IOSTANDARD LVCMOS33 } [get_ports  sw[3]];
set_property -dict { PACKAGE_PIN w15   IOSTANDARD LVCMOS33 } [get_ports  sw[4]];
set_property -dict { PACKAGE_PIN v15   IOSTANDARD LVCMOS33 } [get_ports  sw[5]];
set_property -dict { PACKAGE_PIN w14   IOSTANDARD LVCMOS33 } [get_ports  sw[6]];
set_property -dict { PACKAGE_PIN w13   IOSTANDARD LVCMOS33 } [get_ports  sw[7]];
set_property -dict { PACKAGE_PIN v2   IOSTANDARD LVCMOS33 } [get_ports  sw[8]];
set_property -dict { PACKAGE_PIN t3   IOSTANDARD LVCMOS33 } [get_ports  sw[9]];
set_property -dict { PACKAGE_PIN t2   IOSTANDARD LVCMOS33 } [get_ports  sw[10]];
set_property -dict { PACKAGE_PIN r3   IOSTANDARD LVCMOS33 } [get_ports  sw[11]];
set_property -dict { PACKAGE_PIN w2   IOSTANDARD LVCMOS33 } [get_ports  sw[12]];
set_property -dict { PACKAGE_PIN u1   IOSTANDARD LVCMOS33 } [get_ports  sw[13]];
set_property -dict { PACKAGE_PIN t1   IOSTANDARD LVCMOS33 } [get_ports  sw[14]];
set_property -dict { PACKAGE_PIN r2   IOSTANDARD LVCMOS33 } [get_ports  sw[15]];

set_property -dict { PACKAGE_PIN U16   IOSTANDARD LVCMOS33 } [get_ports  ld[0]];
```

```
set_property -dict { PACKAGE_PIN e19   IOSTANDARD LVCMOS33 } [get_ports  ld[1]];
set_property -dict { PACKAGE_PIN u19   IOSTANDARD LVCMOS33 } [get_ports  ld[2]];
set_property -dict { PACKAGE_PIN v19   IOSTANDARD LVCMOS33 } [get_ports  ld[3]];
set_property -dict { PACKAGE_PIN w18   IOSTANDARD LVCMOS33 } [get_ports  ld[4]];
set_property -dict { PACKAGE_PIN u15   IOSTANDARD LVCMOS33 } [get_ports  ld[5]];
set_property -dict { PACKAGE_PIN u14   IOSTANDARD LVCMOS33 } [get_ports  ld[6]];
set_property -dict { PACKAGE_PIN v14   IOSTANDARD LVCMOS33 } [get_ports  ld[7]];
set_property -dict { PACKAGE_PIN v13   IOSTANDARD LVCMOS33 } [get_ports  ld[8]];
set_property -dict { PACKAGE_PIN v3   IOSTANDARD LVCMOS33 } [get_ports  ld[9]];
set_property -dict { PACKAGE_PIN w3   IOSTANDARD LVCMOS33 } [get_ports  ld[10]];
set_property -dict { PACKAGE_PIN u3   IOSTANDARD LVCMOS33 } [get_ports  ld[11]];
set_property -dict { PACKAGE_PIN p3   IOSTANDARD LVCMOS33 } [get_ports  ld[12]];
set_property -dict { PACKAGE_PIN n3   IOSTANDARD LVCMOS33 } [get_ports  ld[13]];
set_property -dict { PACKAGE_PIN p1   IOSTANDARD LVCMOS33 } [get_ports  ld[14]];
set_property -dict { PACKAGE_PIN l1   IOSTANDARD LVCMOS33 } [get_ports  ld[15]];

set_property -dict { PACKAGE_PIN w5   IOSTANDARD LVCMOS33 } [get_ports  uhr];
set_property -dict { PACKAGE_PIN u17   IOSTANDARD LVCMOS33 } [get_ports  btnd];
```

3.6.2 Arithmetische Funktionen

1.:
VHDL_main:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.std_logic_unsigned.all;
use IEEE.numeric_std.all;

entity main is
    Port ( sw : in STD_LOGIC_vector(15 downto 0);
            uhr : in std_logic;
            btnd : in std_logic;
           ld :out STD_LOGIC_vector(15 downto 0));
end main;

architecture Behavioral of main is

Signal a,b: std_logic_vector (15 downto 0):="0000000000000000";
signal g,h : unsigned (15 downto 0);

begin

a<=sw;
g<=unsigned(a);
h<=shift_left(g,1);
b<=std_logic_vector(h);
ld<=b;

end Behavioral;
```

2.:

VHDL_main:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.std_logic_unsigned.all;
use IEEE.numeric_std.all;

entity main is
    Port ( sw : in STD_LOGIC_vector(15 downto 0);
            uhr : in std_logic;
            btnd : in std_logic;
           ld :out STD_LOGIC_vector(15 downto 0));
end main;

architecture Behavioral of main is

Signal a,b: std_logic_vector (7 downto 0);
Signal c,temp: std_logic_vector (15 downto 0):="0000000000000000";
signal reset : std_logic :='0';
signal f : unsigned (7 downto 0);
signal g : unsigned (15 downto 0);
signal d,e,g0,g1,g2,g3,g4,g5,g6,g7 : unsigned (15 downto 0);

begin

a<= sw(15 downto 8);
b<=sw(7 downto 0);

--d<=unsigned(a);
temp<= X"00" & a;
e<=unsigned(temp);
f<=unsigned(b);
d<="0000000000000000";

process
begin
if(f(0)='1') then
g0<=shift_left(e,0);
else
g0<=d;
end if;

if(f(1)='1') then
g1<=shift_left(e,1);
else
g1<=d;
end if;

if(f(2)='1') then
g2<=shift_left(e,2);
else
```

```
g2<=d;
end if;

if(f(3)='1') then
g3<=shift_left(e,3);
else
g3<=d;
end if;

if(f(4)='1') then
g4<=shift_left(e,4);
else
g4<=d;
end if;

if(f(5)='1') then
g5<=shift_left(e,5);
else
g5<=d;
end if;

if(f(6)='1') then
g6<=shift_left(e,6);
else
g6<=d;
end if;

if(f(7)='1') then
g7<=shift_left(e,7);
else
g7<=d;
end if;
end process;

ld<= std_logic_vector(g0+g1+g2+g3+g4+g5+g6+g7);

end Behavioral;
```

3.:
Shift-Module:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity shift is
    Port ( vin : in Std_logic_vector(15 downto 0);
            shamt : in Std_logic_vector(3 downto 0);
           vout : out Std_logic_vector(15 downto 0));
end shift;

architecture Behavioral of shift is
```

```
signal vtemp1,vtemp2 : std_logic_vector (15 downto 0);

begin

process
begin
case (shamt) is
when "0000"=>
vtemp1<=vin;
when "0001"=>
vtemp1<=vin(14 downto 0) & '0';
when "0010"=>
vtemp1<=vin(13 downto 0) & "00";
when "0011"=>
vtemp1<=vin(12 downto 0) & "00";
when "0100"=>
vtemp1<=vin(11 downto 0) & "00";
when "0101"=>
vtemp1<=vin(10 downto 0) & "00";
when "0110"=>
vtemp1<=vin(9 downto 0) & "00";
when "0111"=>
vtemp1<=vin(8 downto 0) & "00";
when "1000"=>
vtemp1<=vin(7 downto 0) & "00";
when "1001"=>
vtemp1<=vin(6 downto 0) & "00";
when "1010"=>
vtemp1<=vin(5 downto 0) & "00";
when "1011"=>
vtemp1<=vin(4 downto 0) & "00";
when "1100"=>
vtemp1<=vin(3 downto 0) & "00";
when "1101"=>
vtemp1<=vin(2 downto 0) & "00";
when "1110"=>
vtemp1<=vin(1 downto 0) & "00";
when "1111"=>
vtemp1<="0000000000000000";
end case;
end process;

end Behavioral;
```

3.6.3 Arrays

VHDL_main.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.std_logic_unsigned.all;
use IEEE.numeric_std.all;
```

```
entity main is
    Port ( sw : in STD_LOGIC_vector(15 downto 0);
           ld :out STD_LOGIC_vector(15 downto 0));
end main;

architecture Behavioral of main is

Signal temp: std_logic_vector (15 downto 0):="0000000000000000";
signal e,d : unsigned (15 downto 0);
signal i : natural:=7;
signal f : unsigned(7 downto 0);

type zahlvek is array (7 downto 0) of unsigned(15 downto 0);
signal gg : zahlvek;

begin

temp<= X"00" & sw(15 downto 8);
e<=unsigned(temp);
f<=unsigned(sw(7 downto 0));

process
begin

for i in 0 to 7 loop
if(f(i)='1') then
gg(i)<=shift_left(e,i);
else
gg(i)<=d;
end if;
end loop;
end process;

ld<= std_logic_vector(gg(0)+gg(1)+gg(2)+gg(3)+gg(4)+gg(5)+gg(6)+gg(7));

end Behavioral;
```

3.6.4 Variablen

VHDL_main:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.std_logic_unsigned.all;
use IEEE.numeric_std.all;

entity main is
    Port ( sw : in STD_LOGIC_vector(15 downto 0);
           uhr,btnd : in std_logic;
           ld :out STD_LOGIC_vector(15 downto 0));
end main;
```

```
architecture Behavioral of main is

signal a,b,c : std_logic_vector(7 downto 0);

begin

process (btnd)
begin
if (rising_edge(btnd)) then
a <= sw(7 downto 0);
b <= a;
end if;
end process;

process (btnd)
variable var : std_logic_vector (7 downto 0);
begin
if (rising_edge(btnd)) then
var := sw(15 downto 8);
c <= var;
end if;
end process;

ld(15 downto 8)<=c;
ld(7 downto 0)<=b;

end Behavioral;
```

3.6.5 Vergleichsfunktionen

1.:
VHDL_main:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity main is
    Port ( sw : in STD_LOGIC_vector(15 downto 0);
           uhr,btnd : in std_logic;
           ld :out STD_LOGIC_vector(15 downto 0));
end main;

architecture Behavioral of main is

signal a,b,c : std_logic_vector(7 downto 0);

begin

process (uhr)
begin
if(rising_edge(uhr)) then
```

```
if(sw(15 downto 8)>sw(7 downto 0)) then
c<="00000000";
b<="11111111";
else
c<="11111111";
b<="00000000";
end if;
end if;
end process;

ld(15 downto 8)<=c;
ld(7 downto 0)<=b;

end Behavioral;
```

2.:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity main is
    Port ( sw : in STD_LOGIC_vector(15 downto 0);
           uhr,btnd : in std_logic;
           ld :out STD_LOGIC_vector(15 downto 0));
end main;

architecture Behavioral of main is

signal a,b,c : std_logic_vector(7 downto 0);
signal vergleich : std_logic;

function vergleicher(vec1,vec2 : std_logic_vector(7 downto 0))
return std_logic is
variable temp : std_logic;
begin
for i in 7 downto 0 loop
if(vec1(i)='1' and vec2(i)='0') then
temp := '1';
exit;
elsif(vec1(i)='0' and vec2(i)='1') then
temp := '0';
exit;
else
temp := '0';
end if;
end loop;
return temp;
end vergleicher;


begin
```

```
a<=sw(15 downto 8);
b<=sw(7 downto 0);

ld(0)<=vergleicher(b,a);

end Behavioral;
```

Anmerkung: Funktionen sind nicht sequentiell, sequentielle Lösungen wären hier also falsch.

3.7 Serielle Kommunikation

3.7.1 Aufgaben

1.:
BTN_STR : CHAR_ARRAY verändern. Neue Zeichen aus ASCII Tabelle nehmen.
Wenn nötig, constant BTN_STR_LEN : natural := 24 ändern.
2.:
Nachricht wird bei falscher BAUD rate falsch übertragen. Teraterm ist standardmäßig auf 9600 Baud eingestellt. Wenn die BAUD rate in VHDL zu niedrig ist, werden die bits zu lange gehalten und dann doppelt übertragen. Bei zu hoher BAUD rate werden einige Bits nicht lange genug gehalten, um übertragen werden zu können. Dadurch fehlt ein Teil der Nachricht.
3.:
VHDL_main.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.std_logic_unsigned.all;

entity main is
    Port ( CLK : in std_logic;
           UART_TXD  : out  STD_LOGIC;
           BTN  : in  STD_LOGIC_VECTOR (4 downto 0));
end main;

architecture Behavioral of main is

component UART_TX_CTRL
Port(
SEND : in std_logic;
DATA : in std_logic_vector(7 downto 0);
CLK : in std_logic;
READY : out std_logic;
UART_TX : out std_logic
);
end component;

component debouncer
Generic(
        DEBNC_CLOCKS : integer;
        PORT_WIDTH : integer);
Port(
SIGNAL_I : in std_logic_vector(4 downto 0);
```

```
CLK_I : in std_logic;
SIGNAL_O : out std_logic_vector(4 downto 0)
);
end component;

type UART_STATE_TYPE is (RST_REG, LD_INIT_STR, SEND_CHAR, RDY_LOW, WAIT_RDY, WAIT_BTN, LD_BTN_STR);
type CHAR_ARRAY is array (integer range<>) of std_logic_vector(7 downto 0);

--constant TMR_CNTR_MAX : std_logic_vector(26 downto 0) := "101111101011110000100000000";
 --100,000,000 = clk cycles per second
--constant TMR_VAL_MAX : std_logic_vector(3 downto 0) := "1001"; --9
constant RESET_CNTR_MAX : std_logic_vector(17 downto 0) := "110000110101000000";
-- 100,000,000 * 0.002 = 200,000 = clk cycles per 2 ms
constant MAX_STR_LEN : integer := 27;
constant WELCOME_STR_LEN : natural := 27;
constant BTN_STR_LEN : natural := 24;

constant WELCOME_STR : CHAR_ARRAY(0 to 26) := (X"0A",X"0D",X"42",X"41",X"53",X"59",X"53",X"33",
X"20",X"47",X"50",X"49",X"4F",X"2F",X"55",X"41",X"52",X"54",X"20",X"44",X"45",X"4D",X"4F",X"21",
X"0A",X"0A",X"0D");
constant BTN_STR : CHAR_ARRAY(0 to 23) := (X"42",X"75",X"74",X"74",X"6F",X"6E",X"20",X"70",X"72",
X"65",X"73",X"73",X"20",X"64",X"65",X"74",X"65",X"63",X"74",X"65",X"64",X"21",X"0A",X"0D");

signal sendStr : CHAR_ARRAY(0 to (MAX_STR_LEN - 1));
signal strEnd : natural;
signal strIndex : natural;

signal uartRdy : std_logic;
signal uartSend : std_logic := '0';
signal uartData : std_logic_vector (7 downto 0):= "00000000";
signal uartTX : std_logic;

signal uartState : UART_STATE_TYPE := RST_REG;
--signal clk_cntr_reg : std_logic_vector (4 downto 0) := (others=>'0');
signal reset_cntr : std_logic_vector (17 downto 0) := (others=>'0');

signal btnDetect : std_logic;
signal btnDeBnc : std_logic_vector (4 downto 0):="00000";

signal btnReg : std_logic_vector (3 downto 0) := "0000";

begin

--btnDetect<=btn(0);
--btnDeBnc(4)<=btn(0);

btn_reg_process : process (CLK)
begin
if (rising_edge(CLK)) then
btnReg <= btnDeBnc(3 downto 0);
end if;
end process;
```

```
btnDetect <= '1' when ((btnReg(0)='0' and btnDeBnc(0)='1') or
(btnReg(1)='0' and btnDeBnc(1)='1') or
(btnReg(2)='0' and btnDeBnc(2)='1') or
(btnReg(3)='0' and btnDeBnc(3)='1')  ) else
  '0';



process(CLK)
begin
  if (rising_edge(CLK)) then
    if ((reset_cntr = RESET_CNTR_MAX) or (uartState /= RST_REG)) then
      reset_cntr <= (others=>'0');
    else
      reset_cntr <= reset_cntr + 1;
    end if;
  end if;
end process;

next_uartState_process : process (CLK)
begin
if (rising_edge(CLK)) then
if (btnDeBnc(3) = '1') then
uartState <= RST_REG;
else
case uartState is
when RST_REG =>
        if (reset_cntr = RESET_CNTR_MAX) then
          uartState <= LD_INIT_STR;
        end if;
when LD_INIT_STR =>
uartState <= SEND_CHAR;
when SEND_CHAR =>
uartState <= RDY_LOW;
when RDY_LOW =>
uartState <= WAIT_RDY;
when WAIT_RDY =>
if (uartRdy = '1') then
if (strEnd = strIndex) then
uartState <= WAIT_BTN;
else
uartState <= SEND_CHAR;
end if;
end if;
when WAIT_BTN =>
if (btnDetect = '1') then
uartState <= LD_BTN_STR;
end if;
when LD_BTN_STR =>
uartState <= SEND_CHAR;
```

```
when others=> --should never be reached
uartState <= RST_REG;
end case;
end if ;
end if;
end process;

string_load_process : process (CLK)
begin
if (rising_edge(CLK)) then
if (uartState = LD_INIT_STR) then
sendStr <= WELCOME_STR;
strEnd <= WELCOME_STR_LEN;
elsif (uartState = LD_BTN_STR) then
sendStr(0 to 23) <= BTN_STR;
strEnd <= BTN_STR_LEN;
end if;
end if;
end process;

char_count_process : process (CLK)
begin
if (rising_edge(CLK)) then
if (uartState = LD_INIT_STR or uartState = LD_BTN_STR) then
strIndex <= 0;
elsif (uartState = SEND_CHAR) then
strIndex <= strIndex + 1;
end if;
end if;
end process;

char_load_process : process (CLK)
begin
if (rising_edge(CLK)) then
if (uartState = SEND_CHAR) then
uartSend <= '1';
uartData <= sendStr(strIndex);
else
uartSend <= '0';
end if;
end if;
end process;

Inst_btn_debounce: debouncer
    generic map(
        DEBNC_CLOCKS => (2**16),
        PORT_WIDTH => 5)
    port map(
SIGNAL_I => BTN,
CLK_I => CLK,
SIGNAL_O => btnDeBnc
);
```

```
Inst_UART_TX_CTRL: UART_TX_CTRL port map(
SEND => uartSend,
DATA => uartData,
CLK => CLK,
READY => uartRdy,
UART_TX => uartTX
);

UART_TXD <= uartTX;

end Behavioral;
```

Debouncer.vhd:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.std_logic_unsigned.all;
USE IEEE.NUMERIC_STD.ALL;
use IEEE.math_real.all;

entity debouncer is
    Generic ( DEBNC_CLOCKS : INTEGER range 2 to (INTEGER'high) := 2**16;
              PORT_WIDTH : INTEGER range 1 to (INTEGER'high) := 5);
    Port ( SIGNAL_I : in  STD_LOGIC_VECTOR ((PORT_WIDTH - 1) downto 0);
           CLK_I : in  STD_LOGIC;
           SIGNAL_O : out  STD_LOGIC_VECTOR ((PORT_WIDTH - 1) downto 0));
end debouncer;

architecture Behavioral of debouncer is

constant CNTR_WIDTH : integer := natural(ceil(LOG2(real(DEBNC_CLOCKS))));
constant CNTR_MAX : std_logic_vector((CNTR_WIDTH - 1) downto 0) := std_logic_vector
(to_unsigned((DEBNC_CLOCKS - 1), CNTR_WIDTH));
type VECTOR_ARRAY_TYPE is array (integer range <>) of std_logic_vector((CNTR_WIDTH - 1) downto 0);

signal sig_cntrs_ary : VECTOR_ARRAY_TYPE (0 to (PORT_WIDTH - 1)) := (others=>(others=>'0'));

signal sig_out_reg : std_logic_vector((PORT_WIDTH - 1) downto 0) := (others => '0');

begin

debounce_process : process (CLK_I)
begin
   if (rising_edge(CLK_I)) then
   for index in 0 to (PORT_WIDTH - 1) loop
      if (sig_cntrs_ary(index) = CNTR_MAX) then
         sig_out_reg(index) <= not(sig_out_reg(index));
      end if;
   end loop;
   end if;
```

```
end process;

counter_process : process (CLK_I)
begin
if (rising_edge(CLK_I)) then
for index in 0 to (PORT_WIDTH - 1) loop

if ((sig_out_reg(index) = '1') xor (SIGNAL_I(index) = '1')) then
if (sig_cntrs_ary(index) = CNTR_MAX) then
sig_cntrs_ary(index) <= (others => '0');
else
sig_cntrs_ary(index) <= sig_cntrs_ary(index) + 1;
end if;
else
sig_cntrs_ary(index) <= (others => '0');
end if;

end loop;
end if;
end process;

SIGNAL_O <= sig_out_reg;

end Behavioral;
```

Constraints:

```
set_property PACKAGE_PIN W5 [get_ports CLK]
set_property IOSTANDARD LVCMOS33 [get_ports CLK]
create_clock -period 10.000 -name sys_clk_pin -waveform {0.000 5.000} -add [get_ports CLK]

set_property -dict { PACKAGE_PIN t18   IOSTANDARD LVCMOS33 } [get_ports {BTN[0]}];
set_property -dict { PACKAGE_PIN w19   IOSTANDARD LVCMOS33 } [get_ports {BTN[1]}];
set_property -dict { PACKAGE_PIN t17   IOSTANDARD LVCMOS33 } [get_ports {BTN[2]}];
set_property -dict { PACKAGE_PIN U17   IOSTANDARD LVCMOS33 } [get_ports {BTN[3]}];

set_property -dict { PACKAGE_PIN U18   IOSTANDARD LVCMOS33 } [get_ports {BTN[4]}];

##USB-RS232 Interface
##Bank = 16, Pin name = ,Sch name = UART_TXD_IN
#set_property PACKAGE_PIN B18 [get_ports RsRx]
#set_property IOSTANDARD LVCMOS33 [get_ports RsRx]
#Bank = 16, Pin name = ,Sch name = UART_RXD_OUT
set_property PACKAGE_PIN A18 [get_ports UART_TXD]
set_property IOSTANDARD LVCMOS33 [get_ports UART_TXD]

set_property BITSTREAM.GENERAL.COMPRESS TRUE [current_design]
set_property BITSTREAM.CONFIG.SPI_BUSWIDTH 4 [current_design]
set_property CONFIG_MODE SPIx4 [current_design]

set_property BITSTREAM.CONFIG.CONFIGRATE 33 [current_design]
```

```
set_property CONFIG_VOLTAGE 3.3 [current_design]
set_property CFGBVS VCCO [current_design]
```

Der Debouncer hält das Signal des Knopfes über mehrere Taktzyklen. Damit wird verhindert, dass die Nachricht bei einem einzelnen Knopfdruck doppelt gesendet wird.
4.:
VHDL_main:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.std_logic_unsigned.all;

entity main is
    Port ( CLK : in std_logic;
           UART_TXD  : out  STD_LOGIC;
           BTN  : in  STD_LOGIC_VECTOR (4 downto 0));
end main;

architecture Behavioral of main is

component UART_TX_CTRL
Port(
SEND : in std_logic;
DATA : in std_logic_vector(7 downto 0);
CLK : in std_logic;
READY : out std_logic;
UART_TX : out std_logic
);
end component;

component frequenzmessung is
    Port ( uhr_ref,uhr_mess : in std_logic;TR);
type CHAR_ARRAY is array (integer range<>) of std_logic_vector(7 downto 0);
           frequenz : out std_logic_vector(63 downto 0));
end component;

type UART_STATE_TYPE is (RST_REG, LD_INIT_STR, SEND_CHAR, RDY_LOW, WAIT_RDY, WAIT_BTN, LD_BTN_S

constant RESET_CNTR_MAX : std_logic_vector(17 downto 0) := "110000110101000000";
-- 100,000,000 * 0.002 = 200,000 = clk cycles per 2 ms
constant MAX_STR_LEN : integer := 27;
constant WELCOME_STR_LEN : natural := 27;
constant BTN_STR_LEN : natural := 24;

signal freq : std_logic_vector (63 downto 0);
alias freq0 is freq (7 downto 0);
alias freq1 is freq (15 downto 8);
alias freq2 is freq (23 downto 16);
alias freq3 is freq (31 downto 24);
alias freq4 is freq (39 downto 32);
alias freq5 is freq (47 downto 40);
alias freq6 is freq (55 downto 48);
```

```
alias freq7 is freq (63 downto 56);
signal zeilenende : std_logic_vector( 7 downto 0) :=X"0A";
signal FREQ_STR : CHAR_ARRAY(0 to 8):=(freq7,freq6,freq5,freq4,freq3,freq2,freq1,freq0,zeilenende);
constant FREQ_STR_LEN : natural := 9;

signal teiler : std_logic_vector (26 downto 0):="101111101011110000100000000";

constant WELCOME_STR : CHAR_ARRAY(0 to 26) := (X"0A",X"0D",X"42",X"41",X"53",X"59",X"53",
X"33",X"20",X"47",X"50",X"49",X"4F",X"2F",X"55",X"41",X"52",X"54",X"20",X"44",X"45",X"4D",
X"4F",X"21",X"0A",X"0A",X"0D");
constant BTN_STR : CHAR_ARRAY(0 to 23) := (X"42",X"75",X"74",X"74",X"6F",X"6E",X"20",X"70",
X"72",X"65",X"73",X"73",X"20",X"64",X"65",X"74",X"65",X"63",X"74",X"65",X"64",X"21",X"0A",X"0D");



signal sendStr : CHAR_ARRAY(0 to (MAX_STR_LEN - 1));
signal strEnd : natural;
signal strIndex : natural;

signal uartRdy : std_logic;
signal uartSend : std_logic := '0';
signal uartData : std_logic_vector (7 downto 0):= "00000000";
signal uartTX : std_logic;

signal uartState : UART_STATE_TYPE := RST_REG;
signal reset_cntr : std_logic_vector (17 downto 0) := (others=>'0');

signal btnDetect : std_logic;

signal btnReg : std_logic_vector (3 downto 0) := "0000";

begin

process(CLK)
begin
if (rising_edge(CLK)) then
if (teiler="00000000000000000000000000000") then
teiler<="101111101011110000100000000";
btnDetect<='1';
else
teiler<=teiler-1;
btnDetect<='0';
end if;
end if;
end process;

process(CLK)
begin
  if (rising_edge(CLK)) then
    if ((reset_cntr = RESET_CNTR_MAX) or (uartState /= RST_REG)) then
      reset_cntr <= (others=>'0');
    else
```

```
      reset_cntr <= reset_cntr + 1;
    end if;
  end if;
end process;

next_uartState_process : process (CLK)
begin
if (rising_edge(CLK)) then
case uartState is
when RST_REG =>
        if (reset_cntr = RESET_CNTR_MAX) then
          uartState <= LD_INIT_STR;
        end if;
when LD_INIT_STR =>
uartState <= SEND_CHAR;
when SEND_CHAR =>
uartState <= RDY_LOW;
when RDY_LOW =>
uartState <= WAIT_RDY;
when WAIT_RDY =>
if (uartRdy = '1') then
if (strEnd = strIndex) then
uartState <= WAIT_BTN;
else
uartState <= SEND_CHAR;
end if;
end if;
when WAIT_BTN =>
if (btnDetect = '1') then
uartState <= LD_BTN_STR;
end if;
when LD_BTN_STR =>
uartState <= SEND_CHAR;
when others=> --should never be reached
uartState <= RST_REG;
end case;
end if;
end process;

string_load_process : process (CLK)
begin
if (rising_edge(CLK)) then
if (uartState = LD_INIT_STR) then
sendStr <= WELCOME_STR;
strEnd <= WELCOME_STR_LEN;
elsif (uartState = LD_BTN_STR) then
sendStr(0 to 8) <= FREQ_STR;--sendStr(0 to 23) <= BTN_STR;
strEnd <= FREQ_STR_LEN;--strEnd <= BTN_STR_LEN;
end if;
end if;
end process;
```

```
char_count_process : process (CLK)
begin
if (rising_edge(CLK)) then
if (uartState = LD_INIT_STR or uartState = LD_BTN_STR) then
strIndex <= 0;
elsif (uartState = SEND_CHAR) then
strIndex <= strIndex + 1;
end if;
end if;
end process;

char_load_process : process (CLK)
begin
if (rising_edge(CLK)) then
if (uartState = SEND_CHAR) then
uartSend <= '1';
uartData <= sendStr(strIndex);
else
uartSend <= '0';
end if;
end if;
end process;

Inst_UART_TX_CTRL: UART_TX_CTRL port map(
SEND => uartSend,
DATA => uartData,
CLK => CLK,
READY => uartRdy,
UART_TX => uartTX
);

Inst_Frequenzmessung: Frequenzmessung port map(
        uhr_ref=>CLK,
        uhr_mess=>CLK,
        frequenz=>freq
    );

UART_TXD <= uartTX;

end Behavioral;
```

Constraints:

```
set_property PACKAGE_PIN W5 [get_ports CLK]
set_property IOSTANDARD LVCMOS33 [get_ports CLK]
create_clock -period 10.000 -name sys_clk_pin -waveform {0.000 5.000} -add [get_ports CLK]

set_property -dict { PACKAGE_PIN t18   IOSTANDARD LVCMOS33 } [get_ports {BTN[0]}];
set_property -dict { PACKAGE_PIN w19   IOSTANDARD LVCMOS33 } [get_ports {BTN[1]}];
set_property -dict { PACKAGE_PIN t17   IOSTANDARD LVCMOS33 } [get_ports {BTN[2]}];
set_property -dict { PACKAGE_PIN U17   IOSTANDARD LVCMOS33 } [get_ports {BTN[3]}];
```

```
set_property -dict { PACKAGE_PIN U18   IOSTANDARD LVCMOS33 } [get_ports {BTN[4]}];

##USB-RS232 Interface
##Bank = 16, Pin name = ,Sch name = UART_TXD_IN
#set_property PACKAGE_PIN B18 [get_ports RsRx]
#set_property IOSTANDARD LVCMOS33 [get_ports RsRx]
#Bank = 16, Pin name = ,Sch name = UART_RXD_OUT
set_property PACKAGE_PIN A18 [get_ports UART_TXD]
set_property IOSTANDARD LVCMOS33 [get_ports UART_TXD]

set_property BITSTREAM.GENERAL.COMPRESS TRUE [current_design]
set_property BITSTREAM.CONFIG.SPI_BUSWIDTH 4 [current_design]
set_property CONFIG_MODE SPIx4 [current_design]

set_property BITSTREAM.CONFIG.CONFIGRATE 33 [current_design]

set_property CONFIG_VOLTAGE 3.3 [current_design]
set_property CFGBVS VCCO [current_design]

set_property DONT_TOUCH true [get_cells Inst_Frequenzmessung/ringosz/GEN_INV[0].inve1.inv1/aus_i]
set_property DONT_TOUCH true [get_cells Inst_Frequenzmessung/ringosz/GEN_INV[1].inve1.inv1/aus_i]
set_property DONT_TOUCH true [get_cells Inst_Frequenzmessung/ringosz/GEN_INV[2].inve1.inv1/aus_i]
set_property DONT_TOUCH true [get_cells Inst_Frequenzmessung/ringosz/GEN_INV[3].inve1.inv1/aus_i]
set_property DONT_TOUCH true [get_cells Inst_Frequenzmessung/ringosz/GEN_INV[4].inve1.inv1/aus_i]

set_property DONT_TOUCH true [get_nets Inst_Frequenzmessung/ringosz/GEN_INV[0].inve1.inv1/aus]
set_property DONT_TOUCH true [get_nets Inst_Frequenzmessung/ringosz/GEN_INV[1].inve1.inv1/aus]
set_property DONT_TOUCH true [get_nets Inst_Frequenzmessung/ringosz/GEN_INV[2].inve1.inv1/aus]
set_property DONT_TOUCH true [get_nets Inst_Frequenzmessung/ringosz/GEN_INV[3].inve1.inv1/aus]
set_property DONT_TOUCH true [get_nets Inst_Frequenzmessung/ringosz/GEN_INV[4].inve1.inv1/aus]
set_property DONT_TOUCH true [get_nets Inst_Frequenzmessung/ringosz/GEN_INV[5].inve1.inv1/aus]
set_property DONT_TOUCH true [get_nets Inst_Frequenzmessung/ringosz/GEN_INV[6].inve1.inv1/aus]

set_property DONT_TOUCH true [get_nets Inst_Frequenzmessung/ringosz/GEN_INV[0].inve1.inv1/ein]
set_property DONT_TOUCH true [get_nets Inst_Frequenzmessung/ringosz/GEN_INV[1].inve1.inv1/ein]
set_property DONT_TOUCH true [get_nets Inst_Frequenzmessung/ringosz/GEN_INV[2].inve1.inv1/ein]
set_property DONT_TOUCH true [get_nets Inst_Frequenzmessung/ringosz/GEN_INV[3].inve1.inv1/ein]
set_property DONT_TOUCH true [get_nets Inst_Frequenzmessung/ringosz/GEN_INV[4].inve1.inv1/ein]
set_property DONT_TOUCH true [get_nets Inst_Frequenzmessung/ringosz/GEN_INV[5].inve1.inv1/aus]
set_property DONT_TOUCH true [get_nets Inst_Frequenzmessung/ringosz/GEN_INV[6].inve1.inv1/aus]

set_property DONT_TOUCH true [get_nets Inst_Frequenzmessung/ringosz/uhr_ring]
set_property DONT_TOUCH true [get_nets Inst_Frequenzmessung/ringosz/b_1]
set_property DONT_TOUCH true [get_nets Inst_Frequenzmessung/ringosz/b_2]
set_property DONT_TOUCH true [get_nets Inst_Frequenzmessung/ringosz/b_3]
set_property DONT_TOUCH true [get_nets Inst_Frequenzmessung/ringosz/b_4]
set_property DONT_TOUCH true [get_nets Inst_Frequenzmessung/ringosz/b_5]
set_property DONT_TOUCH true [get_nets Inst_Frequenzmessung/ringosz/b_6]

set_property ALLOW_COMBINATORIAL_LOOPS true [get_nets -of_objects
[get_cells Inst_Frequenzmessung/ringosz/GEN_INV[4].inve1.inv1/aus]]
set_property ALLOW_COMBINATORIAL_LOOPS true [get_nets -of_objects
```

```
[get_cells Inst_Frequenzmessung/ringosz/GEN_INV[1].inve1.inv1/aus_inferred__0_i_1]]
set_property ALLOW_COMBINATORIAL_LOOPS true [get_nets -of_objects
[get_cells Inst_Frequenzmessung/ringosz/GEN_INV[2].inve1.inv1/aus_inferred__0_i_1__0]]
set_property ALLOW_COMBINATORIAL_LOOPS true [get_nets -of_objects
[get_cells Inst_Frequenzmessung/ringosz/GEN_INV[3].inve1.inv1/aus_inferred__0_i_1__1]]
set_property ALLOW_COMBINATORIAL_LOOPS true [get_nets -of_objects
[get_cells Inst_Frequenzmessung/ringosz/GEN_INV[3].inve1.inv1/aus_inferred__0_i_1__2]]
set_property ALLOW_COMBINATORIAL_LOOPS true [get_nets -of_objects
[get_cells Inst_Frequenzmessung/ringosz/GEN_INV[3].inve1.inv1/aus_inferred__0_i_1__3]]

##set_property SEVERITY {Warning}  [get_drc_checks LUTLP-1]

##set_property SEVERITY {Warning} [get_drc_checks NSTD-1]
```

Frequenzmessung.vhd:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.std_logic_unsigned.all;
use IEEE.numeric_std.all;

entity frequenzmessung is
    Port ( uhr_ref,uhr_mess : in std_logic;
           frequenz : out std_logic_vector(63 downto 0));
end frequenzmessung;

architecture Behavioral of frequenzmessung is

component ring is
    Port (uhr_ring : out STD_LOGIC);
end component;

signal zahler : std_logic_vector (63 downto 0);
signal uhr_ring_1 : std_logic;

--1x pro Sekunde wird die Frequenz gemessen

begin

ringosz:ring port map(uhr_ring=>uhr_ring_1);

process(uhr_ring_1)
begin
if(rising_edge(uhr_ring_1)) then
zahler<=zahler+1;
end if;
end process;

process(uhr_ref)
variable zahltemp0,zahltemp1,zahltemp2: std_logic_vector (63 downto 0);
variable teiler : std_logic_vector(26 downto 0) :="101111101011110000011111111";
begin
```

```
if(falling_edge(uhr_ref)) then
if(teiler ="00000000000000000000000000000") then
zahltemp1:=zahler;
zahltemp0:=zahltemp1-zahltemp2;
zahltemp2:=zahltemp1;
teiler:="10111110101111000011111111";
else
teiler:=teiler-1;
end if;
end if;
frequenz<=zahltemp0;
end process;

end Behavioral;
```

Ring.vhd:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.std_logic_unsigned.all;
use IEEE.numeric_std.all;

entity ring is
    Port (uhr_ring : out std_logic);
end ring;

architecture Behavioral of ring is

component Inverter is
    Port ( ein : in STD_LOGIC;
           aus : out STD_LOGIC);
end component;

Signal a,b : std_logic_vector (6 downto 0);

begin

GEN_INV:
for i in 0 to 6 generate
inve0:
if (i=0) generate
inv1 : Inverter port map
(ein=>b(6),aus=>b(i));
end generate;
inve1:
if(i/=0) generate
inv1 : Inverter port map
(ein=>b(i-1),aus=>b(i));
end generate;
end generate;

uhr_ring<=b(0);
```

```
end Behavioral;
```

Inverter.vhd:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity Inverter is
    Port ( ein : in STD_LOGIC;
           aus : out STD_LOGIC);
end Inverter;

architecture Behavioral of Inverter is

begin

aus <= not ein;

end Behavioral;
```

UART.vhd:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.std_logic_unsigned.all;

entity UART_TX_CTRL is
    Port ( SEND : in  STD_LOGIC;
           DATA : in  STD_LOGIC_VECTOR (7 downto 0);
           CLK : in  STD_LOGIC;
           READY : out  STD_LOGIC;
           UART_TX : out  STD_LOGIC);
end UART_TX_CTRL;

architecture Behavioral of UART_TX_CTRL is

type TX_STATE_TYPE is (RDY, LOAD_BIT, SEND_BIT);

--constant BIT_TMR_MAX : std_logic_vector(13 downto 0) := "01010001011000";
constant BIT_TMR_MAX : std_logic_vector(13 downto 0) := "10100010110000";
constant BIT_INDEX_MAX : natural := 10;

signal bitTmr : std_logic_vector(13 downto 0) := (others => '0');

signal bitDone : std_logic;

signal bitIndex : natural;

signal txBit : std_logic := '1';

signal txData : std_logic_vector(9 downto 0);
```

```
signal txState : TX_STATE_TYPE := RDY;

begin

next_txState_process : process (CLK)
begin
if (rising_edge(CLK)) then
case txState is
when RDY =>
if (SEND = '1') then
txState <= LOAD_BIT;
end if;
when LOAD_BIT =>
txState <= SEND_BIT;
when SEND_BIT =>
if (bitDone = '1') then
if (bitIndex = BIT_INDEX_MAX) then
txState <= RDY;
else
txState <= LOAD_BIT;
end if;
end if;
when others=>
txState <= RDY;
end case;
end if;
end process;

bit_timing_process : process (CLK)
begin
if (rising_edge(CLK)) then
if (txState = RDY) then
bitTmr <= (others => '0');
else
if (bitDone = '1') then
bitTmr <= (others => '0');
else
bitTmr <= bitTmr + 1;
end if;
end if;
end if;
end process;

bitDone <= '1' when (bitTmr = BIT_TMR_MAX) else
'0';

bit_counting_process : process (CLK)
begin
if (rising_edge(CLK)) then
if (txState = RDY) then
bitIndex <= 0;
```

```
elsif (txState = LOAD_BIT) then
bitIndex <= bitIndex + 1;
end if;
end if;
end process;

tx_data_latch_process : process (CLK)
begin
if (rising_edge(CLK)) then
if (SEND = '1') then
txData <= '1' & DATA & '0';
end if;
end if;
end process;

tx_bit_process : process (CLK)
begin
if (rising_edge(CLK)) then
if (txState = RDY) then
txBit <= '1';
elsif (txState = LOAD_BIT) then
txBit <= txData(bitIndex);
end if;
end if;
end process;

UART_TX <= txBit;
READY <= '1' when (txState = RDY) else
'0';

end Behavioral;
```

Wie in der Aufgabe des Ringoszillators zuvor, lässt sich aus der gemessenen Frequenz des Ringoszillators die Periodendauer des Oszillators berechnen. Wenn man dann noch die Periodendauer durch die Anzahl der verwendeten Inverter-Gatter teilt, erhält man die Gatterlaufzeit eines einzelnen Gatters, Signallaufzeiten außerhalb der Gatter können wieder vernachlässigt werden.
Ungefähres Ergebnis: 1ns
Je nach Vivado Version muss die Constraints Datei ähnlich wie hier angepasst werden, da sonst der Ringoszillator wegoptimiert wird.
Die Implementierung des Frequenzmessgerätes ist hier eine einfache totzeitfreie Variante mit einer Zeitauflösung von 10ns.

3.8 Simulation

3.8.1 Behavioral Simulation

1.: Schalter auf "1" gesetzt und Knopf mehrfach gedrückt.
2.:
TCL_file:

```
restart
```

```
add_force {/main/btnd} -radix bin {1 0ns} {0 5000ps} -repeat_every 10000ps
run 10 ns
add_force {/main/sw} -radix bin {1111111111111111 0ns}
run 50 ns
add_force {/main/sw} -radix bin {0000000000000000 0ns}
run 50 ns
```

Simulationsfenster:

Figure 12: Behavioral_Sim

3.8.2 Post-Synthesis Functional Simulation

1.:
Funktionale Simulation der Schaltung nach der Synthese, also der Schaltung, die Vivado aus dem VHDL gemacht hat.
2.:
Die Signale sind noch nicht initialisiert worden. Der Simulator weiß daher nicht, welche Werte diese haben. Standardmäßig initialisiert Vivado Signale mit "0", dass ignoriert der Simulator aber.
3.:
Diese sind vom Synthese-Tool in Vivado wegoptimiert worden. Man kann sehen, dass diese in "main(Hauptprogramm)_funct_synth.vhd" nicht mehr vorkommen.
4.:

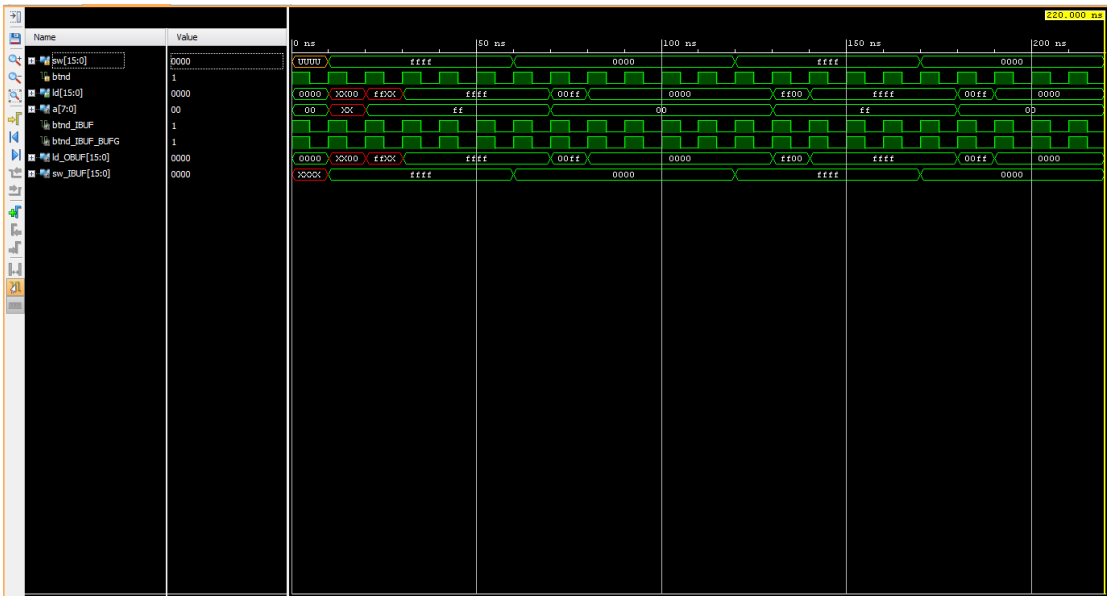

Figure 13: Post_Synth_Funct_Sim

3.8.3 Post-Synthesis Timing Simulation

1:
Simulation der Schaltung nach Synthese unter Annahme von typischen Verzögerungszeiten. Werte der Verzögerungen sind dem SDF (Standard Delay File) entnommen.[9]
2.:

Figure 14: Post_Synth_Timing_Sim

2.:
Die Schalter werden nun zu Beginn mit "Z" angenommen ("Z"= hochohmig), außerdem sind die Eingaben und Ausgaben um einige ns verschoben.

3.8.4 Post-Implementation Functional Simulation

1.:
Funktionale Simulation der Schaltung nach der Implementation, also der Schaltung auf der Hardware des ausgewählten Chips.
2.:

Figure 15: Post_Impl_Synth_Sim

3.:
Keine Unterschiede zur Post-Synthesis Functional Simulation, Unterschiede treten hier erst auf, wenn Sie Code schreiben, der auf dem im Projekt angegebenen Chip nicht funktioniert.

3.8.5 Post-Imlementation Timing Simulation

1.:
Simulation der Schaltung nach Implemenation unter Annahme von realen Verzögerungszeiten. Werte der Verzögerungen sind dem SDF (Standard Delay File) des im Projekt spezifizierten Chips entnommen.[9]
2.:

Figure 16: Post_Impl_Timing_Sim

3.:
Die Schalter werden nun zu Beginn mit "Z" angenommen ("Z"= hochohmig), außerdem sind die Eingaben und Ausgaben um einige ns verschoben.
4.:

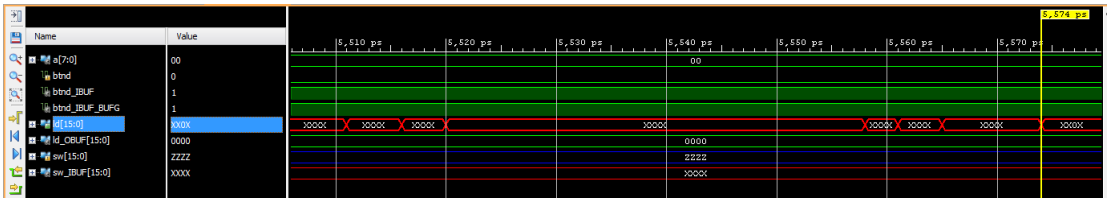

Figure 17: Post_Impl_Timing_Sim_Zoom

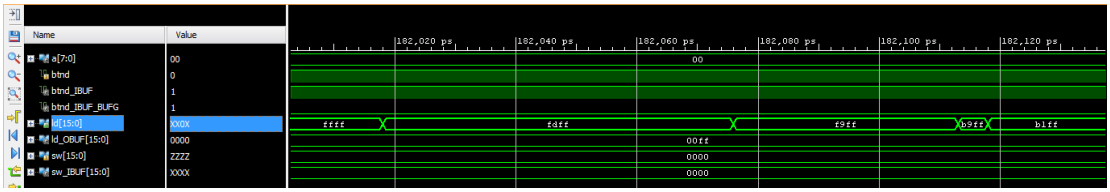

Figure 18: Post_Impl_Timing_Sim_Zoom_2

Die Verzögerungszeiten zu den einzelnen Schaltern und Leds sind nicht exakt gleich. Dadurch werden diese nicht zur exakt gleichen Zeit umgeschaltet, sondern weichen um einige ps ab.

3.9 Digitale Regelung

3.9.1 Phasen-Frequenz-Detektor

1.:
Wenn eine positive Taktflanke in einem der beiden Flip-Flops detektiert wird, schaltet der entsprechende FF seinen Ausgang auf "1". Sobald beide Ausgänge auf "1" stehen, werden beide Ausgänge zurückgesetzt. Dadurch steht der FF, der mit einer höheren Frequenz betrieben wird, häufiger auf "1". Damit lässt sich die Frequenz und die Phase der beiden Signale vergleichen.

2.:
VHDL_PFD

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity PFD is
    Port ( freqref,freqdiv : in STD_LOGIC;
           up,down : out STD_LOGIC);
end PFD;

architecture Behavioral of PFD is

signal upintern,downintern : std_logic;

begin

process(upintern,downintern,freqref,freqdiv)
begin
if((upintern and downintern)='1') then
upintern<='0';
downintern<='0';
else
if(rising_edge(freqref)) then
upintern<='1';
end if;
if(rising_edge(freqdiv)) then
downintern<='1';
end if;
end if;
end process;

up<=upintern;
down<=downintern;

end Behavioral;
```

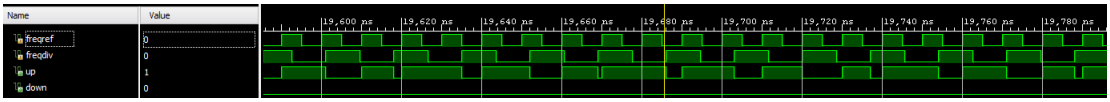

Figure 19: Simulation PFD

3.9.2 Digitale Ladungspumpe

1.:
Lineare Aufladung, kein Maximalwert. Bei Spannungsquelle: exponentieller Verlauf bis Maximalwert.

2.:
VHDL_CP:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.std_logic_unsigned.all;

entity updowncounter is
    Port ( uhr : in STD_LOGIC;
           schnelleuhr : in std_logic;
           kontroll0 : in STD_LOGIC;
           kontroll1 : in STD_LOGIC;
           registerwert : out STD_LOGIC_VECTOR (31 downto 0));
end updowncounter;

architecture Behavioral of updowncounter is

signal kontroll : std_logic_vector (1 downto 0);
signal registerintern : std_logic_vector (31 downto 0):="10000000000000000000000000000000";

begin

process(uhr,schnelleuhr,kontroll,kontroll0,kontroll1)
begin
if(rising_edge(uhr)) then
if(kontroll0='1' and kontroll1 ='0' and registerintern<"11111111111111111111111111111111") then
registerintern<=registerintern+1;
end if;
if(kontroll0='0' and kontroll1 ='1' and registerintern>"00000000000000000000000000000000") then
registerintern<=registerintern-1;
end if;
end if;
end process;

registerwert<=registerintern;

end Behavioral;
```

Entspricht Modell mit Stromquelle, Kondensator-Äquivalent: Register mit Inkrementierfunk-

tion.

3.9.3 Numerisch kontrollierter Oszillator

1.:
Bei jedem Überlauf des Registers muss man ein Signal invertieren. Die Anzahl der Taktzyklen, die benötigt werden, um den Überlauf zu erreichen, ist von einem Steuersignal abhängig. Der Registerwert wird mit jedem Taktzyklus um den Wert des Steuersignals inkrementiert. Durch Änderungen am Wert des Steuersignals lässt sich die Frequenz ändern.
2.:
VHDL_NCO:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.std_logic_unsigned.all;

entity NCO is
    Port ( freqin : in STD_LOGIC;
           kontrollzahler : in std_logic_vector(31 downto 0);
           freqoutnco : out STD_LOGIC);
end NCO;

architecture Behavioral of NCO is

signal kontrollregister : std_logic_vector(31 downto 0);
signal freqintern : std_logic;
--signal freqininvert : std_logic;

begin

--freqininvert<=not freqin;

process(freqin)
begin
if(rising_edge(freqin)) then
kontrollregister<=kontrollregister+kontrollzahler;
if((kontrollregister+kontrollzahler)<kontrollregister) then
freqintern <= not freqintern;
end if;
end if;
end process;

freqoutnco<=freqintern;

end Behavioral;
```

3.:
Man kann jedem Wert des NCO-Registern einen Wert der Sinuskurve zuweisen. Dies ist die Funktionsweise eines Phase-Amplitude-Converters. Diese Schaltung wäre allerdings zu groß um es auf dem hier verwendeten Chip zu realisieren.

3.9.4 Phasenregelschleife

1.,2.,3.:
VHDL_main:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.std_logic_unsigned.all;

entity main is
    Port (uhr : in std_logic;
          sw : in std_logic_vector(15 downto 0);
          ld : out std_logic_vector(15 downto 0);
          UART_TXD  : out  STD_LOGIC;
          BTN  : in  STD_LOGIC_VECTOR (4 downto 0)
          );
end main;

architecture Behavioral of main is

component FD is
    Port ( freqin : in STD_LOGIC;
           teilfak: in std_logic_vector (15 downto 0);
           freqout : out STD_LOGIC);
end component;

component PFD is
    Port ( freqref,freqdiv : in STD_LOGIC;
           up,down : out STD_LOGIC);
end component;

component NCO is
    Port ( freqin : in STD_LOGIC;
           kontrollzahler : in std_logic_vector(31 downto 0);
           freqoutnco : out STD_LOGIC);
end component;

component updowncounter is
    Port ( uhr : in STD_LOGIC;
           schnelleuhr : in std_logic;
           kontroll0 : in STD_LOGIC;
           kontroll1 : in STD_LOGIC;
           registerwert : out STD_LOGIC_VECTOR (31 downto 0));
end component;

component frequenzmessung is
    Port ( uhr_ref,uhr_mess : in std_logic;
           frequenz : out std_logic_vector(63 downto 0));
end component;

component UART_TX_CTRL is
    Port ( SEND : in  STD_LOGIC;
```

```
           DATA : in  STD_LOGIC_VECTOR (7 downto 0);
           CLK : in  STD_LOGIC;
           READY : out  STD_LOGIC;
           UART_TX : out  STD_LOGIC);
end component;

component LowPassFilter is
    Port ( regin : in STD_LOGIC_vector (31 downto 0);
           uhr : std_logic;
           regout : out STD_LOGIC_vector (31 downto 0));
end component;

signal freqaus,hoch,runter,freq11,freq22,freq33,freq44,freq55,freqnco,freqausgang,
freqtest,kontrolle,schnelluhr : std_logic;
signal teilfaktor1,teilfaktor2,kontroll,phasenwert : std_logic_vector (15 downto 0);
signal ncoregister,ncoregistertest : std_logic_vector(31 downto 0);

type UART_STATE_TYPE is (RST_REG, LD_INIT_STR, SEND_CHAR, RDY_LOW, WAIT_RDY, WAIT_BTN, LD_BTN_STR);
type CHAR_ARRAY is array (integer range<>) of std_logic_vector(7 downto 0);

constant RESET_CNTR_MAX : std_logic_vector(17 downto 0) := "110000110101000000";
-- 100,000,000 * 0.002 = 200,000 = clk cycles per 2 ms
constant MAX_STR_LEN : integer := 27;
constant WELCOME_STR_LEN : natural := 27;
constant BTN_STR_LEN : natural := 24;

signal freq : std_logic_vector (63 downto 0);
alias freq0 is freq (7 downto 0);
alias freq1 is freq (15 downto 8);
alias freq2 is freq (23 downto 16);
alias freq3 is freq (31 downto 24);
alias freq4 is freq (39 downto 32);
alias freq5 is freq (47 downto 40);
alias freq6 is freq (55 downto 48);
alias freq7 is freq (63 downto 56);
signal zeilenende : std_logic_vector( 7 downto 0) :=X"0A";
signal FREQ_STR : CHAR_ARRAY(0 to 8):=(freq7,freq6,freq5,freq4,freq3,freq2,freq1,freq0,zeilenende);
constant FREQ_STR_LEN : natural := 9;

signal teiler : std_logic_vector (26 downto 0):="101111101011110000100000000";

constant WELCOME_STR : CHAR_ARRAY(0 to 26) := (X"0A",X"0D",X"42",X"41",X"53",X"59",X"53",
X"33",X"20",X"47",X"50",X"49",X"4F",X"2F",X"55",X"41",X"52",X"54",X"20",X"44",X"45",X"4D",
X"4F",X"21",X"0A",X"0A",X"0D");
constant BTN_STR : CHAR_ARRAY(0 to 23) := (X"42",X"75",X"74",X"74",X"6F",X"6E",X"20",X"70",
X"72",X"65",X"73",X"73",X"20",X"64",X"65",X"74",X"65",X"63",X"74",X"65",X"64",X"21",X"0A",X"0D");



signal sendStr : CHAR_ARRAY(0 to (MAX_STR_LEN - 1));
signal strEnd : natural;
signal strIndex : natural;
```

```
signal uartRdy : std_logic;
signal uartSend : std_logic := '0';
signal uartData : std_logic_vector (7 downto 0):= "00000000";
signal uartTX : std_logic;

signal uartState : UART_STATE_TYPE := RST_REG;
signal reset_cntr : std_logic_vector (17 downto 0) := (others=>'0');

signal btnDetect : std_logic;

signal btnReg : std_logic_vector (3 downto 0) := "0000";

signal CLK,hochtest,runtertest : std_logic;

begin

--Frequenzgenerierung durch PLL hier

teilfaktor1<="0111111111111111";
Vorteiler: FD port map (freqin=>uhr,teilfak=>teilfaktor1,freqout=>freq11);

Detektor: PFD port map (freqref=>freq11,freqdiv=>freqausgang,up=>hoch,down=>runter);

HochRunter: updowncounter port map(uhr=>uhr,schnelleuhr=>schnelluhr,kontroll0=>hoch,
kontroll1=>runter,registerwert=>filterregister);

Filter: LowPassFilter port map(regin=>filterregister,uhr=>uhr,regout=>ncoregister);

StandardNCO: NCO port map(freqin=>uhr,kontrollzahler=>ncoregister,freqoutnco=>freqnco);

teilfaktor2<=sw;
Ruckteiler: FD port map (freqin=>freqnco,teilfak=>teilfaktor2,freqout=>freqausgang);




--Frequenzmessung beginnt hier

CLK<=uhr;

Inst_UART_TX_CTRL: UART_TX_CTRL port map(SEND=>uartSend,
DATA=>uartData,CLK=>CLK,READY=>uartRdy,UART_TX=>uartTX);

Inst_Frequenzmessung: Frequenzmessung port map(uhr_ref=>CLK,uhr_mess=>freqnco,frequenz=>freq);

process(CLK)
begin
if (rising_edge(CLK)) then
if (teiler="00000000000000000000000000000") then
teiler<="1011111010111100001000000000";
btnDetect<='1';
```

```
else
teiler<=teiler-1;
btnDetect<='0';
end if;
end if;
end process;

process(CLK)
begin
  if (rising_edge(CLK)) then
    if ((reset_cntr = RESET_CNTR_MAX) or (uartState /= RST_REG)) then
      reset_cntr <= (others=>'0');
    else
      reset_cntr <= reset_cntr + 1;
    end if;
  end if;
end process;

next_uartState_process : process (CLK)
begin
if (rising_edge(CLK)) then
case uartState is
when RST_REG =>
        if (reset_cntr = RESET_CNTR_MAX) then
          uartState <= LD_INIT_STR;
        end if;
when LD_INIT_STR =>
uartState <= SEND_CHAR;
when SEND_CHAR =>
uartState <= RDY_LOW;
when RDY_LOW =>
uartState <= WAIT_RDY;
when WAIT_RDY =>
if (uartRdy = '1') then
if (strEnd = strIndex) then
uartState <= WAIT_BTN;
else
uartState <= SEND_CHAR;
end if;
end if;
when WAIT_BTN =>
if (btnDetect = '1') then
uartState <= LD_BTN_STR;
end if;
when LD_BTN_STR =>
uartState <= SEND_CHAR;
when others=> --should never be reached
uartState <= RST_REG;
end case;
end if;
end process;
```

```
string_load_process : process (CLK)
begin
if (rising_edge(CLK)) then
if (uartState = LD_INIT_STR) then
sendStr <= WELCOME_STR;
strEnd <= WELCOME_STR_LEN;
elsif (uartState = LD_BTN_STR) then
sendStr(0 to 8) <= FREQ_STR;--sendStr(0 to 23) <= BTN_STR;
strEnd <= FREQ_STR_LEN;--strEnd <= BTN_STR_LEN;
end if;
end if;
end process;

char_count_process : process (CLK)
begin
if (rising_edge(CLK)) then
if (uartState = LD_INIT_STR or uartState = LD_BTN_STR) then
strIndex <= 0;
elsif (uartState = SEND_CHAR) then
strIndex <= strIndex + 1;
end if;
end if;
end process;

char_load_process : process (CLK)
begin
if (rising_edge(CLK)) then
if (uartState = SEND_CHAR) then
uartSend <= '1';
uartData <= sendStr(strIndex);
else
uartSend <= '0';
end if;
end if;
end process;

UART_TXD <= uartTX;

end Behavioral;
```

Um die Frequenz zu testen, wurde das Frequenzmessmodul aus der vorherigen Übung wiederverwendet.
VHDL_FD:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.std_logic_unsigned.all;

entity FD is
    Port ( freqin : in STD_LOGIC;
           teilfak: in std_logic_vector (15 downto 0);
           freqout : out STD_LOGIC);
```

```
end FD;

architecture Behavioral of FD is

signal teilzahl : std_logic_vector(15 downto 0);
signal freqintern : std_logic;

begin

process(freqin)
begin
if(rising_edge(freqin)) then
if(teilzahl="0000000000000000") then
teilzahl<=teilfak;
freqintern<=not freqintern;
else
teilzahl<=teilzahl-1;
end if;
end if;
end process;

freqout<=freqintern;

end Behavioral;
```

3.9.5 Digitale Filter

1.:
Tiefpassfilter, analoges Äquivalent zum Beispiel ein RC-Tiefpass
2.:
VHDL_Filter:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
--use ieee.std_logic_unsigned.all;
use IEEE.numeric_std.all;

entity LowPassFilter is
    Port ( regin : in STD_LOGIC_vector (31 downto 0);
           uhr : std_logic;
           regout : out STD_LOGIC_vector (31 downto 0));
end LowPassFilter;

architecture Behavioral of LowPassFilter is

type regarr is array (0 to 29) of unsigned (31 downto 0);
signal inte : regarr;

--signal zahler : std_logic_vector (4 downto 0);

begin
```

```
process(uhr)
variable inte1 : regarr;
variable zahler : integer;
variable schnitt : unsigned (63 downto 0);
--variable schnitt1 : unsigned (63 downto 0);
variable regarrlen : integer :=30;
begin
if(rising_edge(uhr)) then
for i in 0 to regarrlen-2 loop
inte1(i) := inte1(i+1);
end loop;
inte1(regarrlen-1) := unsigned(regin);
for i in 0 to regarrlen-1 loop
schnitt := schnitt+(inte1(i));
end loop;
schnitt := schnitt/regarrlen;
end if;
regout<=std_logic_vector(schnitt(31 downto 0));
end process;

end Behavioral;
```

3.:
VHDl_main:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.std_logic_unsigned.all;

entity main is
    Port (uhr : in std_logic;
          sw : in std_logic_vector(15 downto 0);
          ld : out std_logic_vector(15 downto 0);
          UART_TXD  : out  STD_LOGIC;
          BTN  : in  STD_LOGIC_VECTOR (4 downto 0)
          );
end main;

architecture Behavioral of main is

component FD is
    Port ( freqin : in STD_LOGIC;
           teilfak: in std_logic_vector (15 downto 0);
           freqout : out STD_LOGIC);
end component;

component PFD is
    Port ( freqref,freqdiv : in STD_LOGIC;
           up,down : out STD_LOGIC);
end component;

component NCO is
```

```
    Port ( freqin : in STD_LOGIC;
           kontrollzahler : in std_logic_vector(31 downto 0);
           freqoutnco : out STD_LOGIC);
end component;

component updowncounter is
    Port ( uhr : in STD_LOGIC;
           schnelleuhr : in std_logic;
           kontroll0 : in STD_LOGIC;
           kontroll1 : in STD_LOGIC;
           registerwert : out STD_LOGIC_VECTOR (31 downto 0));
end component;

component frequenzmessung is
    Port ( uhr_ref,uhr_mess : in std_logic;
           frequenz : out std_logic_vector(63 downto 0));
end component;

component UART_TX_CTRL is
    Port ( SEND : in  STD_LOGIC;
           DATA : in  STD_LOGIC_VECTOR (7 downto 0);
           CLK : in  STD_LOGIC;
           READY : out  STD_LOGIC;
           UART_TX : out  STD_LOGIC);
end component;

signal freqaus,hoch,runter,freq11,freq22,freq33,freq44,freq55,freqnco,freqausgang,
freqtest,kontrolle,schnelluhr : std_logic;
signal teilfaktor1,teilfaktor2,kontroll,phasenwert : std_logic_vector (15 downto 0);
signal ncoregister,ncoregistertest,filterregister : std_logic_vector(31 downto 0);

type UART_STATE_TYPE is (RST_REG, LD_INIT_STR, SEND_CHAR, RDY_LOW, WAIT_RDY, WAIT_BTN, LD_BTN_STR);
type CHAR_ARRAY is array (integer range<>) of std_logic_vector(7 downto 0);

constant RESET_CNTR_MAX : std_logic_vector(17 downto 0) := "110000110101000000";
-- 100,000,000 * 0.002 = 200,000 = clk cycles per 2 ms
constant MAX_STR_LEN : integer := 27;
constant WELCOME_STR_LEN : natural := 27;
constant BTN_STR_LEN : natural := 24;

signal freq : std_logic_vector (63 downto 0);
alias freq0 is freq (7 downto 0);
alias freq1 is freq (15 downto 8);
alias freq2 is freq (23 downto 16);
alias freq3 is freq (31 downto 24);
alias freq4 is freq (39 downto 32);
alias freq5 is freq (47 downto 40);
alias freq6 is freq (55 downto 48);
alias freq7 is freq (63 downto 56);
signal zeilenende : std_logic_vector( 7 downto 0) :=X"0A";
signal FREQ_STR : CHAR_ARRAY(0 to 8):=(freq7,freq6,freq5,freq4,freq3,freq2,freq1,freq0,zeilenende);
constant FREQ_STR_LEN : natural := 9;
```

```
signal teiler : std_logic_vector (26 downto 0):="10111110101111000010000000";

constant WELCOME_STR : CHAR_ARRAY(0 to 26) := (X"0A",X"0D",X"42",X"41",X"53",X"59",X"53",
X"33",X"20",X"47",X"50",X"49",X"4F",X"2F",X"55",X"41",X"52",X"54",X"20",X"44",X"45",X"4D",
X"4F",X"21",X"0A",X"0A",X"0D");
constant BTN_STR : CHAR_ARRAY(0 to 23) := (X"42",X"75",X"74",X"74",X"6F",X"6E",X"20",X"70",
X"72",X"65",X"73",X"73",X"20",X"64",X"65",X"74",X"65",X"63",X"74",X"65",X"64",X"21",X"0A",X"0D");



signal sendStr : CHAR_ARRAY(0 to (MAX_STR_LEN - 1));
signal strEnd : natural;
signal strIndex : natural;

signal uartRdy : std_logic;
signal uartSend : std_logic := '0';
signal uartData : std_logic_vector (7 downto 0):= "00000000";
signal uartTX : std_logic;

signal uartState : UART_STATE_TYPE := RST_REG;
signal reset_cntr : std_logic_vector (17 downto 0) := (others=>'0');

signal btnDetect : std_logic;

signal btnReg : std_logic_vector (3 downto 0) := "0000";

signal CLK,hochtest,runtertest : std_logic;

begin

--Frequenzgenerierung durch PLL hier

teilfaktor1<="0111111111111111";
Vorteiler: FD port map (freqin=>uhr,teilfak=>teilfaktor1,freqout=>freq11);

Detektor: PFD port map (freqref=>freq11,freqdiv=>freqausgang,up=>hoch,down=>runter);

HochRunter: updowncounter port map(uhr=>uhr,schnelleuhr=>schnelluhr,kontroll0=>hoch,
kontroll1=>runter,registerwert=>ncoregister);


StandardNCO: NCO port map(freqin=>uhr,kontrollzahler=>ncoregister,freqoutnco=>freqnco);

teilfaktor2<=sw;
Ruckteiler: FD port map (freqin=>freqnco,teilfak=>teilfaktor2,freqout=>freqausgang);




--Frequenzmessung beginnt hier
```

```
CLK<=uhr;

Inst_UART_TX_CTRL: UART_TX_CTRL port map(SEND=>uartSend,DATA=>uartData,
CLK=>CLK,READY=>uartRdy,UART_TX=>uartTX);

Inst_Frequenzmessung: Frequenzmessung port map(uhr_ref=>CLK,uhr_mess=>freqnco,frequenz=>freq);

process(CLK)
begin
if (rising_edge(CLK)) then
if (teiler="00000000000000000000000000") then
teiler<="10111110101111000010000000";
btnDetect<='1';
else
teiler<=teiler-1;
btnDetect<='0';
end if;
end if;
end process;

process(CLK)
begin
  if (rising_edge(CLK)) then
    if ((reset_cntr = RESET_CNTR_MAX) or (uartState /= RST_REG)) then
      reset_cntr <= (others=>'0');
    else
      reset_cntr <= reset_cntr + 1;
    end if;
  end if;
end process;

next_uartState_process : process (CLK)
begin
if (rising_edge(CLK)) then
case uartState is
when RST_REG =>
        if (reset_cntr = RESET_CNTR_MAX) then
          uartState <= LD_INIT_STR;
        end if;
when LD_INIT_STR =>
uartState <= SEND_CHAR;
when SEND_CHAR =>
uartState <= RDY_LOW;
when RDY_LOW =>
uartState <= WAIT_RDY;
when WAIT_RDY =>
if (uartRdy = '1') then
if (strEnd = strIndex) then
uartState <= WAIT_BTN;
else
uartState <= SEND_CHAR;
end if;
```

```
end if;
when WAIT_BTN =>
if (btnDetect = '1') then
uartState <= LD_BTN_STR;
end if;
when LD_BTN_STR =>
uartState <= SEND_CHAR;
when others=> --should never be reached
uartState <= RST_REG;
end case;
end if;
end process;

string_load_process : process (CLK)
begin
if (rising_edge(CLK)) then
if (uartState = LD_INIT_STR) then
sendStr <= WELCOME_STR;
strEnd <= WELCOME_STR_LEN;
elsif (uartState = LD_BTN_STR) then
sendStr(0 to 8) <= FREQ_STR;--sendStr(0 to 23) <= BTN_STR;
strEnd <= FREQ_STR_LEN;--strEnd <= BTN_STR_LEN;
end if;
end if;
end process;

char_count_process : process (CLK)
begin
if (rising_edge(CLK)) then
if (uartState = LD_INIT_STR or uartState = LD_BTN_STR) then
strIndex <= 0;
elsif (uartState = SEND_CHAR) then
strIndex <= strIndex + 1;
end if;
end if;
end process;

char_load_process : process (CLK)
begin
if (rising_edge(CLK)) then
if (uartState = SEND_CHAR) then
uartSend <= '1';
uartData <= sendStr(strIndex);
else
uartSend <= '0';
end if;
end if;
end process;

UART_TXD <= uartTX;

end Behavioral;
```

4 Fazit

In dieser Arbeit konnten 9 Projekte erarbeitet werden, mit denen die Grundlagen von VHDL und FPGA-Design präsentiert werden konnten. Es wurden Grundlagen zur Projektstrukturierung aufgezeigt und darauf aufbauend komplexe Schaltungen für spezifische Anwendungszwecke aufgebaut werden. Das letzte Projekt zeigte jedoch schon die Limitierung des Basys3-boards, da es selbst stark vereinfacht, kaum auf den Artix7-Chip passte. Noch komplexere Projekte sind daher nicht möglich.

Literatur

[1] balacha. Demystifying resets: Synchronous, asynchronous other design considerations... part 1. [https://forums.xilinx.com/t5/Adaptable-Advantage-Blog/Demystifying-Resets-Synchronous-Asynchronous-other-Design/ba-p/882252], 2018.

[2] Don Mills Clifford E. Cummings. Synchronous resets? asynchronous resets?i am so confused!how will i ever know which to use? [http://www.sunburst-design.com/papers/CummingsSNUG2002SJ_Resets.pdf], 2002.

[3] Analog Devices. Fundamentals of phase locked loops (plls). [https://www.analog.com/media/en/training-seminars/tutorials/MT-086.pdf], 2019.

[4] Digilent. Basys 3 general i/o demo. [https://reference.digilentinc.com/learn/programmable-logic/tutorials/basys-3-general-io/start], 2019.

[5] Digilent. Basys 3 general i/o demo. [https://reference.digilentinc.com/learn/programmable-logic/tutorials/basys-3-general-io/start], 2019.

[6] TU Wien. Fir-filter. [https://ti.tuwien.ac.at/cps/teaching/courses/dspv/files/FIRFilter.pdf], 2010.

[7] Wikipedia.org. Charge pump. [https://en.wikipedia.org/wiki/Charge_pump], 2019.

[8] Wikipedia.org. Numerically controlled oscillator. [https://en.wikipedia.org/wiki/Numerically_controlled_oscillator], 2019.

[9] Inc. Xilinx. Synchronous resets? asynchronous resets?i am so confused!how will i ever know which to use? [https://www.xilinx.com/support/documentation/sw_manuals/xilinx11/ise_c_simulation_timing.htm], 2009.