

Vivado-Einführungskurs

Teil 2: Hierarchische VHDL-Modelle, Spezifikation, Testplan

Mit VHDL kann neben dem *Verhalten* digitaler Schaltungen auch deren hierarchische *Struktur* beschrieben werden. Im Zusammenhang mit der Simulation haben Sie in Teil 1 dieses Einführungskurses bereits eine Strukturbeschreibung kennengelernt: Der Zähler wird dort in der Testbench als untergeordnete Komponente erst deklariert, dann instanziiert und schließlich über Signale an die Prozesse der Testbench angeschlossen.

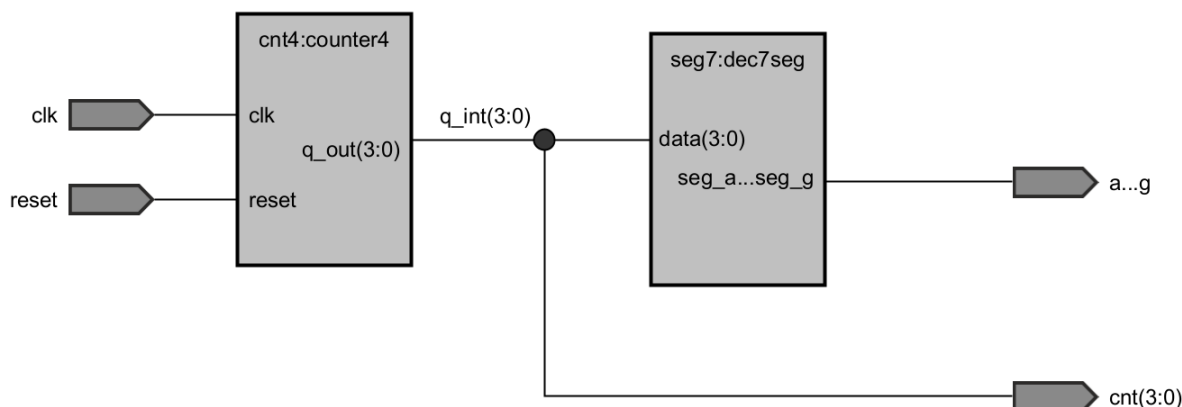
In den folgenden Abschnitten werden Sie den Zähler aus dem ersten Teil des Einführungskurses als Komponente in eine übergeordnete Strukturbeschreibung einbauen.

Inhalt

1	SPEZIFIKATION	2
2	ERSTELLEN EINES VIVADO-PROJEKTS	3
3	HINZUFÜGEN VON VHDL-DATEIEN ZUM PROJEKT.....	4
4	ERSTELLEN EINER STRUKTURBESCHREIBUNG	4
5	TESTPLAN.....	7
6	TESTBENCH	8
7	SIMULATION	9
8	ELABORATION / RTL-ANALYSE	11
9	SYNTHESE	12
10	DEFINITION VON RANDBEDINGUNGEN (<i>CONSTRAINTS</i>)	12
11	IMPLEMENTIERUNG.....	14
12	ERZEUGEN DES BITSTREAMS.....	14
13	KONFIGURIEREN DES FPGAS	14
14	FUNKTIONSPRÜFUNG AUF DER HARDWARE.....	14

1 Spezifikation

Eine VHDL-Strukturbeschreibung soll die Ausgänge des Zählers aus dem ersten Teil des Vivado-Einführungskurses mit den Eingängen eines Siebensegment-Dekoders verbinden. Neben den Signalen zur Ansteuerung einer Siebensegment-Anzeige (siehe Dokument *Technischer Anhang*) soll die Schaltung auch den Zählerstand im Binärcode ausgeben. Die nachfolgende Abbildung zeigt zur Verdeutlichung das **Blockschaltbild der Strukturbeschreibung**¹.



Die *Spezifikation* beschreibt u.a. das Verhalten und die Schnittstellen der Schaltung vollständig und korrekt. Für synchrone digitale Schaltungen (und nur um die geht es im Labor Hardwarebeschreibungssprachen) lässt sich eine solche Spezifikation relativ kompakt in tabellarischer Form darstellen:

reset	$cnt_t (3:0)$ <small>Binärcode, 1-aktiv</small>	$cnt_{t+1} (3:0)$ <small>Zustandswechsel mit steigender Taktflanke</small>
1	0000	Nicht definiert
0	cnt_t	$cnt_t + 1$

Hier wird zunächst das *sequentielle* Verhalten der Schaltung beschrieben – im konkreten Fall also das Verhalten des Zählers in seiner zeitlichen Abfolge. Bei `reset = 1` steht der Zählerausgang `cnt(3:0)` auf `00002`. Der Zustand des Ausgangs `cnt` nach der nächsten steigenden Taktflanke (Zeitpunkt $t+1$) ergibt sich durch Addition von 1 zum aktuellen Zählerstand.

Es ist nicht bekannt, wie lange das Resetsignal aktiv ist, von daher ist der Wert des Zählerausgangs im Zeitpunkt $t+1$ nicht definiert.

Aus der verwendeten Bitbreite (4) und der Binärcodierung folgt, dass auf den Zählerstand `11112` der Zählerstand `00002` folgen muss.

¹ Die hier dargestellten Blöcke repräsentieren **keine VHDL-Prozesse**, sondern **VHDL-Komponenten**. Es handelt sich also **nicht** um ein **RTL-Blockschaltbild**! Hinter jeder Komponente verbirgt sich wieder eine Verhaltens- oder Strukturbeschreibung.

Die nächste Tabelle zeigt das *kombinatorische* Verhalten der Schaltung, hier den Zusammenhang zwischen Zählerstand und den Ausgängen des Siebensegment-Dekoders²:

cnt(3:0)	a b c d e f g
0000	0 0 0 0 0 0 1
0001	1 0 0 1 1 1 1
0010	0 0 1 0 0 1 0
0011	0 0 0 0 1 1 0
0100	1 0 0 1 1 0 0
0101	0 1 0 0 1 0 0
0110	0 1 0 0 0 0 0
0111	0 0 0 1 1 1 1
1000	0 0 0 0 0 0 0
1001	0 0 0 0 1 0 0
1010	0 0 0 1 0 0 0
1011	1 1 0 0 0 0 0
1100	0 1 1 0 0 0 1
1101	1 0 0 0 0 1 0
1110	0 1 1 0 0 0 0
1111	0 1 1 1 0 0 0

2 Erstellen eines Vivado-Projekts

- Legen Sie – sofern noch nicht geschehen – anhand der Beschreibung aus dem ersten Teil des Vivado-Einführungskurses ein Stammverzeichnis für Ihre Projekte und ein Vorlagenverzeichnis an, so dass z.B. folgende Verzeichnisstruktur entsteht:

```
c:\
├─ userdata
│   └─ musterma
│       ├── vivado      Stammverzeichnis für Vivado-Projekte
│       └─ vorlagen    Verzeichnis für Vorlagen
```

- Holen Sie die Datei **vorlagen_v2_vorbereitung.zip** von der Laborseite auf der E-Learning-Plattform der Hochschule und entpacken Sie das ZIP-Archiv im soeben erstellten Vorlagen-Verzeichnis.
- Starten Sie Vivado und legen Sie mit **Create Project...** ein neues Projekt **intro2** an. Wählen Sie als *Project location* das zuvor erstellte Stammverzeichnis für die Vivado-Projektdateien und aktivieren Sie die Option **Create project subdirectory**. Das Projekt sollte nun im Pfad **c:\userdata\<Anmeldename>\vivado\intro2** abgespeichert werden.

Achtung: Verwenden Sie in Pfad- oder Dateinamen **keine** Leer- und Sonderzeichen oder Umlaute!

Wählen Sie die übrigen Projekteinstellungen analog zum ersten Teil dieses Einführungskurses:

Project Type: **RTL Project**
Target language: **VHDL**
Simulator language: **Mixed**
Part: **xc7a35tcpg236-1**

² Zur Ansteuerung des Siebensegmentanzeige siehe das Dokument *Technischer Anhang*.

3 Hinzufügen von VHDL-Dateien zum Projekt

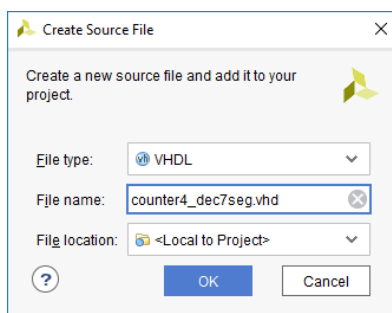
- Fügen Sie über **Flow Navigator > Project Manager > Add Sources > Add or create design sources** die Dateien `counter4.vhd` und `dec7seg.vhd` aus dem Vorlagenverzeichnis `vorlagen_v2` zum Projekt hinzu. Aktivieren Sie dabei die Option **Copy sources into project**.
- Die Datei `counter4.vhd` kennen sie bereits. Öffnen Sie die Datei `dec7seg.vhd` und studieren Sie deren Inhalt, das VHDL-Modell eines Siebensegment-Dekoders:
 - Wofür wird ein Siebensegment-Dekoder verwendet?
 - Welche Signale sind am Eingang des Siebensegment-Dekoders anzulegen, was wird am Ausgang ausgegeben?
 - Wie wurde das Verhalten des Siebensegment-Dekoders beschrieben?

Der Code des Siebensegment-Dekoders ist fehlerfrei, Sie brauchen ihn daher nicht zu verändern. Um den Umgang mit dem Simulator zu üben, enthält der Zähler einige Fehler, die Sie zum jetzigen Zeitpunkt bitte nicht korrigieren.

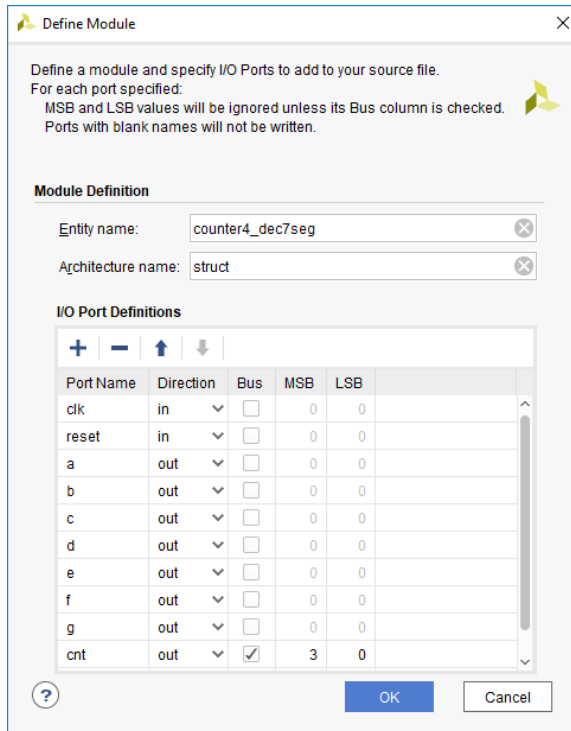
4 Erstellen einer Strukturbeschreibung

Die VHDL-Verhaltensbeschreibungen in den beiden Dateien `counter4.vhd` und `dec7seg.vhd` bilden die untere Hierarchieebene der Gesamtschaltung. In einer neuen VHDL-Strukturbeschreibung verbinden Sie nun auf der darüber liegenden Hierarchieebene die Ausgänge des Zählers mit den Eingängen des Siebensegment-Dekoders.

- Fügen Sie zum Erstellen der Strukturbeschreibung über **Flow Navigator > Project Manager > Add Sources > Add or create design sources > Create File** eine neue Datei zum Projekt hinzu.
- Geben Sie der neuen Datei den Namen `counter4_dec7seg.vhd` und schließen Sie den Dialog mit **Finish** ab.



- Der nächste Dialog fordert Sie zur Definition der Namen von Entity und Architecture, sowie zur Eingabe der Ein- und Ausgangsports auf. Den vorgegebenen Namen der Entity können Sie übernehmen. Der Name der Architecture ist frei wählbar, sollte sinnvollerweise aber **struct** heißen (es wird eine Strukturbeschreibung erstellt, keine Verhaltensbeschreibung!). Bezüglich der Ein- und Ausgangsports orientieren Sie sich am oben dargestellten Blockschaltbild.



Define a module and specify I/O Ports to add to your source file.
For each port specified:
MSB and LSB values will be ignored unless its Bus column is checked.
Ports with blank names will not be written.

Module Definition

Entity name:

Architecture name:

I/O Port Definitions

Port Name	Direction	Bus	MSB	LSB
clk	in	<input type="checkbox"/>	0	0
reset	in	<input type="checkbox"/>	0	0
a	out	<input type="checkbox"/>	0	0
b	out	<input type="checkbox"/>	0	0
c	out	<input type="checkbox"/>	0	0
d	out	<input type="checkbox"/>	0	0
e	out	<input type="checkbox"/>	0	0
f	out	<input type="checkbox"/>	0	0
g	out	<input type="checkbox"/>	0	0
cnt	out	<input checked="" type="checkbox"/>	3	0

OK Cancel

- Öffnen Sie die Datei **counter4_dec7seg.vhd**. Auf Basis der soeben vorgenommenen Einstellungen wurden von Vivado eine Entity und eine leere Architecture erstellt. Erweitern Sie nun die Architecture folgendermaßen:
 - Deklarieren Sie die Komponenten `counter4` und `dec7seg`.
 - Deklarieren Sie Signale zum Herstellen von Verbindungen zwischen den Komponenten.
 - Instanziiieren und verbinden Sie die Komponenten `counter4` und `dec7seg`.
 - Stellen Sie eine Verbindung zwischen dem internen Signal `q_int` und dem Ausgangssignal `cnt` her.

Als Vorlage für die Strukturbeschreibung können Sie den nachfolgend abgedruckten VHDL-Code verwenden. Bitte studieren Sie die einzelnen Abschnitte genau – Sie werden im Laufe des Labors mehrere Strukturbeschreibungen erstellen und sollten wissen, was Sie tun.

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

entity counter4_dec7seg is
    Port ( clk : in STD_LOGIC;
          reset : in STD_LOGIC;
          a : out STD_LOGIC;
          b : out STD_LOGIC;
          c : out STD_LOGIC;
          d : out STD_LOGIC;
          e : out STD_LOGIC;
          f : out STD_LOGIC;
          g : out STD_LOGIC;
          cnt : out STD_LOGIC_VECTOR (3 downto 0));
end counter4_dec7seg;
```

```
architecture struct of counter4_dec7seg is
    -- Komponentendeklaration
    component counter4 is
        port (
            clk      : in std_logic;
            reset    : in std_logic;
            q_out    : out std_logic_vector(3 downto 0)
        );
    end component;

    component dec7seg is
        port (
            data : in std_logic_vector(3 downto 0);
            seg_a, seg_b, seg_c, seg_d: out std_logic;
            seg_e, seg_f, seg_g : out std_logic );
    end component;

    -- Deklaration des internen Signals
    signal q_int : std_logic_vector(3 downto 0);

begin
    -- Instanzieren und Verbinden der Komponenten
    cnt4: counter4 port map (
        clk      => clk,
        reset    => reset,
        q_out    => q_int
    );

    seg7: dec7seg port map (
        data => q_int,
        seg_a => a,
        seg_b => b,
        seg_c => c,
        seg_d => d,
        seg_e => e,
        seg_f => f,
        seg_g => g
    );

    -- Internes Signal mit Ausgangsport verbinden
    cnt <= q_int;

end struct;
```

Beachten Sie, dass die Ausgangssignale des Zählers (`q_out`) bei der Instanziierung der Komponente nicht gleichzeitig an die Eingänge des Siebensegment-Dekoders (`data`) und den Ausgangsport `cnt` angeschlossen werden können, da die Richtung des Ports `cnt` auf `out` gesetzt wurde. Dieses Problem wird mit Hilfe des zusätzlichen Signals `q_int` gelöst.

- Speichern Sie zum Abschluss die neu erstellte Strukturbeschreibung und beobachten Sie das Fenster *Sources*: Der Zähler und der Siebensegment-Dekoder liegen jetzt eine Hierarchieebene unterhalb der Strukturbeschreibung.

5 Testplan

Mit einer Simulation bzw. einem Hardwaretest wird überprüft, ob die Funktion der Schaltung der Spezifikation entspricht. Dies geschieht durch das Anlegen von Eingangssignalen (Stimuli) und das Beobachten von Ausgangssignalen. Bei Bedarf können auch interne Signale untersucht werden.

Bei der Vorbereitung einer Simulation bzw. eines Tests können folgende Fragen hilfreich sein:

- Soll die Schaltung vollständig getestet werden, oder wird nur eine Teilfunktion untersucht?
- Welche Zustände der Schaltung müssen für den Test durchlaufen werden? Bei einem Zähler könnten das z.B. alle möglichen Zählerstände sein.
- Welche Eingangssignale sind für den Test relevant?
- Mit welcher Abfolge von Eingangssignalen kann der beabsichtigte Test durchgeführt werden?
- Ist die korrekte Arbeitsweise der Schaltung eindeutig aus den Werten und dem Verlauf der Ausgangssignale ableitbar?
- Wie lange (z.B. Anzahl von Taktzyklen) dauert der Test bei der gewählten Abfolge von Eingangssignalen?

Im vorliegenden Fall soll die Zusammenschaltung aus Zähler und Siebensegment-Dekoder vollständig getestet werden. Neben dem Takt besitzt die Schaltung nur das Eingangssignal `reset`. Für die beiden möglichen Werte dieses Signals müssen alle theoretisch möglichen Zustandsübergänge (sprich: Zählerstände) durchlaufen werden: insgesamt 16 Stück für `reset = 0` und einer für `reset = 1`. Der vollständige Test des Siebensegment-Dekoders ergibt sich automatisch, da seine Eingänge direkt mit den Ausgängen des Zählers verbunden sind.

Diese Vorüberlegungen werden in einem Testplan zusammengefasst:

Testumfang	vollständig (Zähler + Siebensegment-Dekoder)	
Eingangssignale ³	<code>reset</code>	asynchrones Resetsignal
	<code>clk</code>	Systemtakt, 100 MHz, steigende Flanke
Ausgangssignale ³	<code>cnt(3:0)</code>	Zählerstand, binär
	<code>a, b, c, d, e, f, g</code>	Ausgänge des Siebensegment-Dekoders
Testsequenz / Testdauer ⁴	1. Reset	1 Taktzyklus
	2. Durchlaufen aller Zustandsübergänge	16 Taktzyklen

Der Testplan bildet die Grundlage für die Erzeugung der Stimuli in der Testbench und für das Vorgehen beim späteren Test der Schaltung auf der Hardware.

³ Unter *Eingangssignale* bzw. *Ausgangssignale* sind nur die für den Test relevanten Signale aufgeführt.

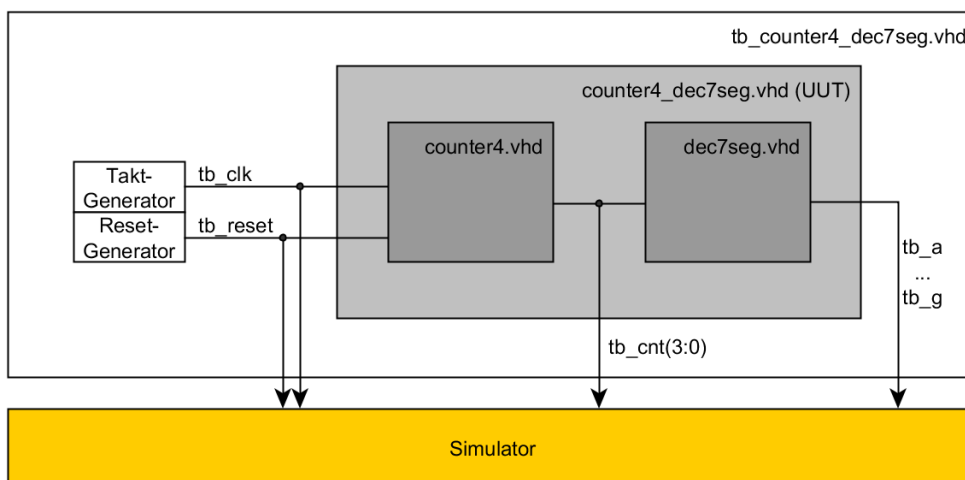
⁴ Die tatsächliche Testdauer ergibt sich aus der Periodendauer des verwendeten Taktsignals. Bei einer Taktfrequenz von 100 MHz wäre der o.g. Test nach 170 ns abgeschlossen.

6 Testbench

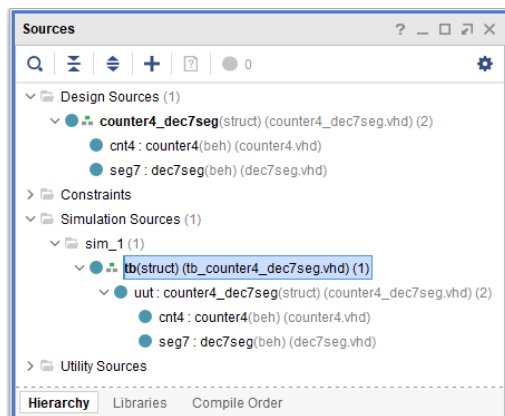
Nachdem nun geklärt ist, welche Abfolge von Eingangssignalen erzeugt werden muss, kann die VHDL-Testbench erstellt werden. Zum Erstellen einer Vorlage für die Testbench gehen Sie vor, wie im ersten Teil des Vivado-Einführungskurses beschrieben:

- Führen Sie in Vivado mit **Tools > Run Tcl Script** das Tcl-Script **testbench_generator/generate_testbench.tcl** aus.
- Falls eine Fehlermeldung angezeigt wird, schauen Sie in der *Tcl Console* nach und informieren sich anhand der dort angezeigten Meldungen über die Ursache. Korrigieren Sie die Fehler und führen Sie danach erneut das Tcl-Script aus.
- Editieren Sie den VHDL-Code der Testbench:
 - Passen Sie die Signalnamen des Takt- und des Resetsignals an. Beachten Sie, dass die Ports der Komponente `counter4_dec7seg` in der Testbench mit Signalen verbunden wurden, die den Präfix `tb_` tragen, z.B. `tb_clk`.
 - Bei welchem Wert des Reset-Signals soll der Zähler zurückgesetzt werden (aktiver Pegel)? Setzen Sie die Konstante `reset_active_level` auf den entsprechenden Wert.
 - Setzen Sie die Zeit, für die der Reset aktiv ist (`reset_active_time`), auf eine Taktperiode (siehe Testplan).
 - Die Schaltung soll in der Simulation mit einer Taktfrequenz von 100 MHz betrieben werden. Wählen Sie einen passenden Wert für `clk_period`.
 - Alle Eingangssignale der Schaltung außer dem Takt `clk` werden im Prozess `stimulus_process` erzeugt. Entspricht der Quellcode dem Testplan?

Die gesamte Schaltung aus Zähler, Siebensegment-Dekoder, Strukturbeschreibung und Testbench besitzt nun folgenden hierarchischen Aufbau: Auf der untersten Ebene liegen zwei Verhaltensbeschreibungen (`counter4.vhd` und `dec7seg.vhd`). Darüber angeordnet ist die Strukturbeschreibung `counter4_dec7seg.vhd` (bestehend aus zwei Komponenten) und darüber wiederum die Testbench `tb_counter4_dec7seg.vhd`. Die Testbench ist eine Mischung aus einer Verhaltens- und Strukturbeschreibung: In ihr ist eine Komponente instanziiert, zusätzlich enthält Sie Prozesse zum Erzeugen von Takt und Reset. Die nachfolgende Abbildung verdeutlicht diese Zusammenhänge (Achtung: Dies ist nur eine schematische Darstellung, kein RTL- oder Struktur-Blockschaltbild!).

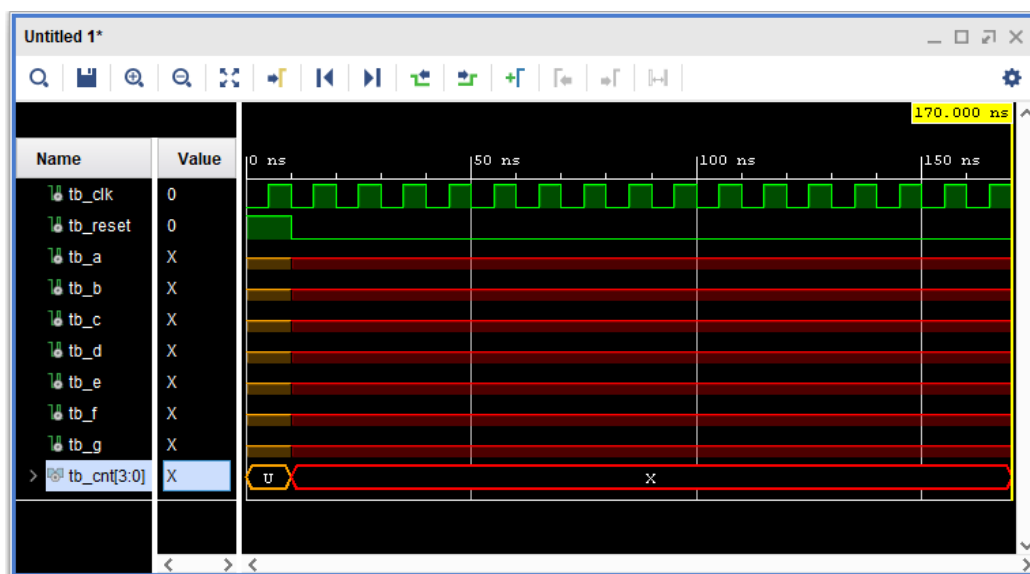


In Vivado finden Sie denselben hierarchischen Aufbau, wenn Sie im Panel **Sources** den Punkt **Simulation Sources > sim_1** erweitern:

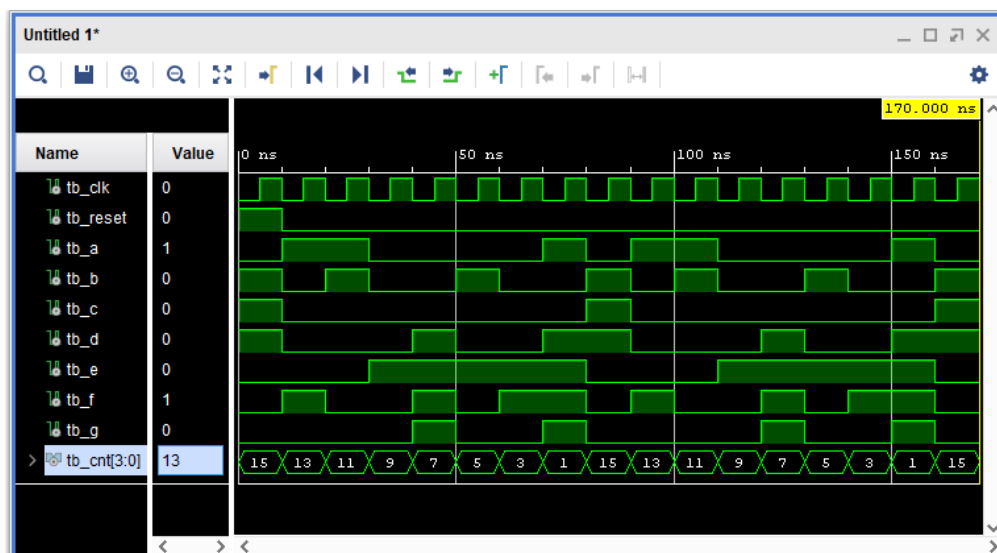


7 Simulation

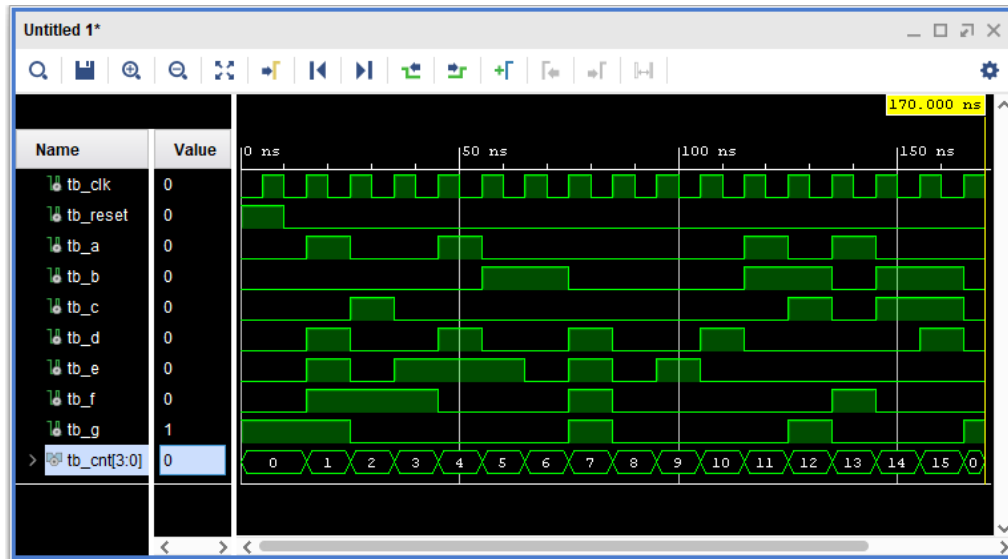
- Starten Sie den Simulator mit **Flow Navigator > Simulation > Run Simulation > Run Behavioral Simulation**. Falls die Simulation wegen Fehlern im Code nicht gestartet werden kann, hilft ein Blick auf die Reiter *Messages*, *Log* und *Tcl Console*.
- Ziehen Sie – sofern notwendig – anhand des Testplans alle Signale in die Signaldarstellung, die für die Überprüfung der Funktionalität erforderlich sind (beachten Sie den Präfix *tb_*). Die Konstanten *clk_period*, *reset_active_time* und *reset_active_level* können sie aus der Signaldarstellung entfernen.
- Ändern Sie den Radix des Signals *tb_cnt[3:0]* auf **Unsigned Decimal**.
- Setzen Sie die Simulationszeit auf 0 zurück, indem Sie in der *Tcl Console* das Kommando **restart** eingeben.
- Laut Testplan dauert eine vollständiger Simulationsdurchlauf 17 Taktzyklen, das entspricht 170 ns bei der gewählten Taktfrequenz. Simulieren Sie durch Eingabe des Befehls **run 170 ns** bis zum Zeitpunkt 170 ns. Die Signalverläufe sollten ungefähr folgendermaßen aussehen:



- Ganz offensichtlich gibt es mehrere Probleme:
 - Während der aktiven Phase des Resetsignals sind alle Ausgänge orange eingefärbt und besitzen den Wert „U“ (= Undefined). Der Reset scheint also keine Auswirkung zu haben, obwohl der Pegel des Resetimpulses korrekt ist.
→ kontrollieren Sie, ob der Zähler durch einen ‚1‘-Pegel des Resetsignals zurückgesetzt wird und korrigieren Sie ggf. den VHDL-Code.
 - Nach dem Resetimpuls sind alle Ausgänge rot eingefärbt. Das „X“ ist ein Hinweis darauf, dass ein Signal mehrere Treiber besitzt, also in mehreren Prozessen beschrieben wird.
→ Überprüfen Sie die Zuweisungsoperatoren (\leq) in den beiden Zählerprozessen. Steht in beiden Prozessen derselbe Signalname auf der linken Seite des Operators? Falls ja, sollten Sie dies korrigieren.
- Speichern Sie den geänderten VHDL-Code und starten Sie die Simulation neu mit **Run > Relaunch Simulation**. Geben Sie anschließend wieder `restart` und `run 170 ns` in der *Tcl Console* ein.



- Das sieht schon deutlich besser aus, allerdings nur auf den ersten Blick:
 - Klicken Sie auf das Signal `tb_cnt`, ganz links in dem Bereich, in dem der Signalwert aus Platzgründen ausgeblendet ist. Der Cursor springt zum Zeitpunkt 0 ns, in der Spalte *Value* sehen Sie den Signalwert 15. Durch den Resetimpuls wird der Zähler also nicht auf 0, sondern auf 15 zurückgesetzt.
 - Der Zähler zählt zwar, allerdings abwärts und in Zweierschritten.
 - Klicken Sie auf das Signal `tb_cnt`, genau auf die gekreuzten Linien zwischen zwei aufeinanderfolgenden Zählerständen. Anders als gefordert, ändert sich der Zählerstand mit der fallenden Taktflanke.
- Korrigieren Sie den VHDL-Code und starten Sie die Simulation neu.

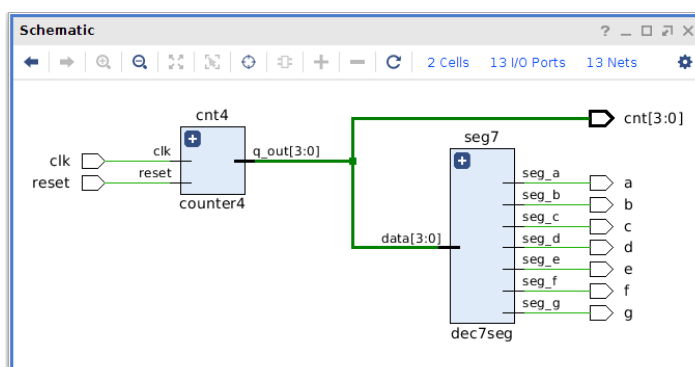


- Prüfen Sie nun noch einmal anhand der Spezifikation, ob alles (auch die Ausgangssignale des Siebensegment-Dekoders!) ordnungsgemäß funktioniert.
- Speichern Sie die Einstellungen der Signaldarstellung mit **File > Simulation Waveform > Save Configuration**, übernehmen Sie den vorgeschlagenen Dateinamen und fügen Sie die gespeicherte Datei auf Nachfrage zum Projekt hinzu.
- Wenn Sie die Simulation erfolgreich abgeschlossen haben, beenden Sie den Simulator durch einen Klick auf das Kreuz rechts oben im Titelbalken **SIMULATION**.

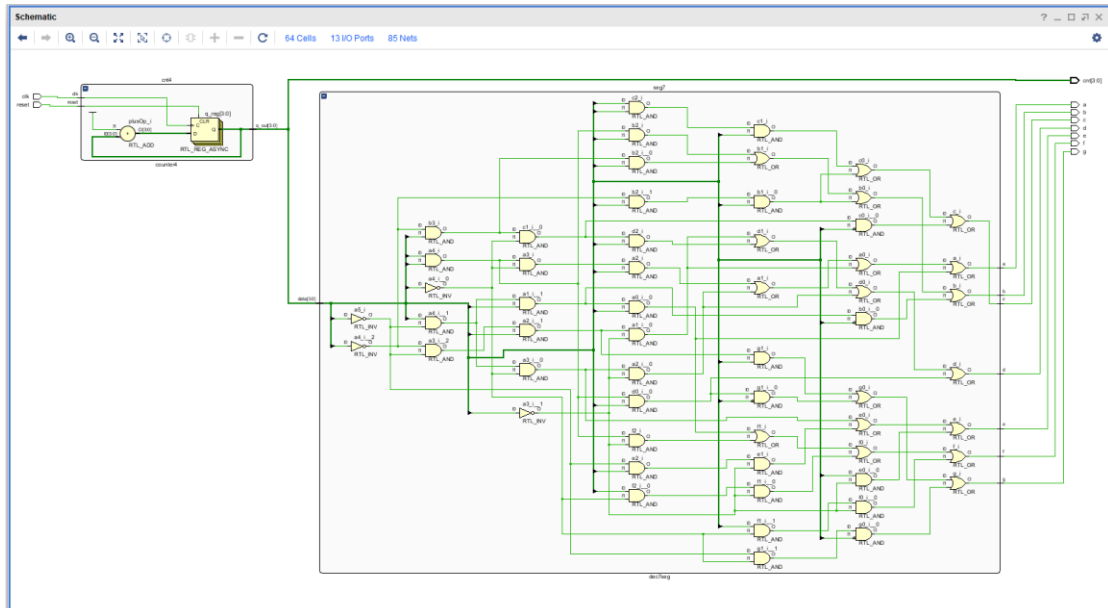
8 Elaboration / RTL-Analyse

- Starten Sie die RTL-Analyse mit **Flow Navigator > RTL Analysis > Open Elaborated Design** und bestätigen Sie die Rückfrage, ob das Werkzeug mit den Einstellungen zur Definition von Randbedingungen (z.B. Timing, I/O-Pins) gestartet werden soll, mit **OK**.

Vivado extrahiert aus dem VHDL-Code folgendes Blockschaltbild:



- Erweitern Sie die beiden Blöcke `cnt4` und `seg7`:



Das RTL-Blockschaltbild des Zählers `cnt4` finden Sie in der linken oberen Ecke, Sie kennen es bereits aus dem ersten Teil dieses Kurses. Der Siebensegment-Dekoder wurde mit Logikgattern modelliert und lässt sich so nur schwer „von Hand“ überprüfen.

- Schließen Sie das elaborierte Design.

9 Synthese

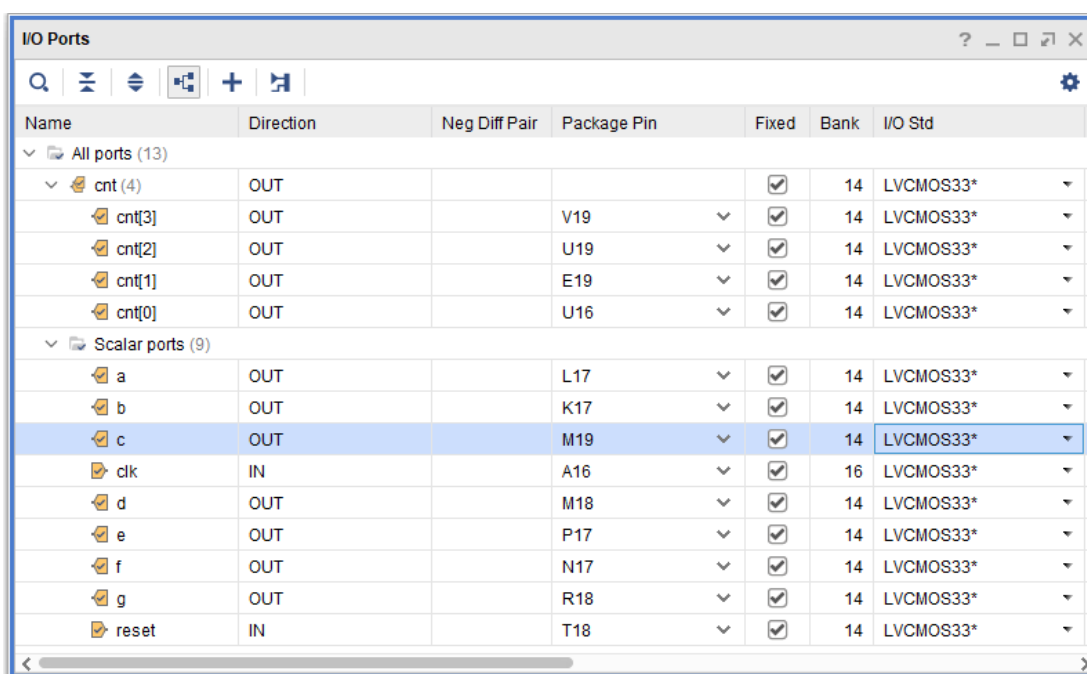
- Starten Sie die Synthese mit **Flow Navigator > Synthesis > Run Synthesis**.
- Klicken Sie nach dem Abschluss der Synthese auf **View Reports** und überprüfen Sie die Meldungen im Reiter **Messages**. Entfernen Sie das Häkchen bei **Info**, so dass nur die Warnungen sichtbar sind und löschen Sie ggf. vorhandene alte Meldungen durch Anklicken des „Mülleimers“. Korrigieren Sie ggf. Ihren VHDL-Code, falls z.B. unvollständige Sensitivitätslisten oder Latches infolge unvollständig auskodierter IF-Bedingungen gemeldet werden.
- Wählen Sie den Reiter **Project Summary** und klicken Sie unter **Utilization** auf **Table**. Der Siebensegment-Dekoder wird als Schaltnetz in LUTs realisiert, die vier Flip-Flops werden für das Register im 4-Bit-Zähler benötigt. Die Anzahl der I/O-Ports stimmt ebenfalls: Takt + Reset + vier Zählerausgänge + sieben Ausgänge zum Ansteuern der Siebensegment-Anzeige = 13 I/O-Ports.

10 Definition von Randbedingungen (Constraints)

Auf der Laborhardware sind Takt, Reset, die Zählerausgänge und die Ansteuersignale für die Siebensegment-Anzeige zu verdrahten:

- Takt: „Langsamer“ Takt des Peripherieboards, wird über BU2, Pin A16/CCIO an das FPGA angeschlossen
- Reset: BTNU auf dem BASYS3-Board
- Zählerausgänge: Leuchtdioden LD3 ... LD0 auf dem BASYS3-Board. Auf LD0 soll das LSB (Least Significant Bit = niederwertigstes Bit) ausgegeben werden.

- Ansteuersignale für Siebensegment-Dekoder: Es soll die Siebensegment-Anzeige auf dem Steckbrett des Peripherieboards zum Einsatz kommen. Die Ansteuersignale müssen also über BU2 auf das Peripherieboard geführt werden.
- Informieren Sie sich im Dokument **Technischer Anhang** über die FPGA-Pinnummern der jeweiligen Ein- und Ausgänge. Beachten Sie auch den **Bestückungsaufdruck** des Laborsystems.
 - Öffnen Sie das synthetisierte Design mit **Flow Navigator > Synthesis > Open Synthesized Design**.
 - Wählen Sie in der Werkzeugleiste die Ansicht **I/O Planning**.
 - Erweitern Sie im Reiter **I/O Ports** die Einträge **cnt** und **Scalar ports**. Tragen Sie unter **Package Pin** die Pinnummern ein, an die die jeweiligen Signale angeschlossen werden sollen. Die Pinnummern für die Ansteuersignale der Siebensegment-Anzeige können Sie auf BU2 frei wählen.
 - Ändern Sie bei allen Signalen den Wert in der Spalte **I/O Std** auf **LVC MOS33***.



Name	Direction	Neg Diff Pair	Package Pin	Fixed	Bank	I/O Std
All ports (13)						
cnt (4)						
cnt[3]	OUT		V19	<input checked="" type="checkbox"/>	14	LVC MOS33*
cnt[2]	OUT		U19	<input checked="" type="checkbox"/>	14	LVC MOS33*
cnt[1]	OUT		E19	<input checked="" type="checkbox"/>	14	LVC MOS33*
cnt[0]	OUT		U16	<input checked="" type="checkbox"/>	14	LVC MOS33*
Scalar ports (9)						
a	OUT		L17	<input checked="" type="checkbox"/>	14	LVC MOS33*
b	OUT		K17	<input checked="" type="checkbox"/>	14	LVC MOS33*
c	OUT		M19	<input checked="" type="checkbox"/>	14	LVC MOS33*
clk	IN		A16	<input checked="" type="checkbox"/>	16	LVC MOS33*
d	OUT		M18	<input checked="" type="checkbox"/>	14	LVC MOS33*
e	OUT		P17	<input checked="" type="checkbox"/>	14	LVC MOS33*
f	OUT		N17	<input checked="" type="checkbox"/>	14	LVC MOS33*
g	OUT		R18	<input checked="" type="checkbox"/>	14	LVC MOS33*
reset	IN		T18	<input checked="" type="checkbox"/>	14	LVC MOS33*

- Speichern Sie die Randbedingungen mit **File > Constraints > Save**. Bestätigen Sie die Meldung, dass die Syntheseresultate anschließend möglicherweise veraltet sein werden, mit **OK**. Geben Sie der Datei im folgenden Dialog den Namen **constr.xdc** (der Name ist beliebig, wichtig ist nur die Erweiterung **.xdc**).
- Zur Definition der Randbedingung für den Systemtakt wählen Sie **Flow Navigator > Synthesis > Open Synthesized Design > Constraints Wizard > Next** und setzen die Frequenz des Taktsignals **clk** unter **Recommended Constraints > Frequency (MHz)** auf **100**. Überspringen Sie alle weiteren Dialogfenster mit **Skip to Finish >>**.
- Beenden Sie das Definieren der Randbedingungen durch einen Klick auf das Kreuz rechts oben im Titelfeld **SYNTHESIZED DESIGN**.
- Die erstellten Randbedingungen finden Sie im Fenster **Sources > Constraints**. Öffnen Sie die Datei **constr.xdc** und studieren Sie den Inhalt. Für den weiteren Entwurfsprozess ist bezüglich der Randbedingungen nur diese Datei relevant. Zur Definition der Randbedingungen hätte es also ausgereicht, diese Datei zu erstellen.

11 Implementierung

- Klicken Sie auf **Flow Navigator > Implementation > Run Implementation**. Bestätigen Sie die Meldung, dass die Synthese aufgrund veralteter Ergebnisse neu gestartet werden muss.
- Klicken Sie nach dem Abschluss der Implementierung auf **View Reports** und überprüfen Sie die Meldungen im Reiter **Messages**. Entfernen Sie ggf. die Häkchen bei *Info* und *Status*, setzen Sie die Häkchen bei *Error* und *Warning* und löschen Sie die veralteten Meldungen. Die jetzt verbliebenen Meldungen sollten Sie ernst nehmen und bei Bedarf Ihren VHDL-Code korrigieren.

12 Erzeugen des Bitstreams

- Klicken Sie auf **Flow Navigator > Program and Debug > Generate Bitstream**.
- Wählen Sie nach dem Generieren des Bitstreams die Option **Open Reports**. Ignorieren Sie die Warnmeldung im Reiter **Messages**, die Sie darauf hinweist, dass zwei Konfigurationsparameter nicht gesetzt sind (CFGBVS, CONFIG_VOLTAGE).

13 Konfigurieren des FPGAs

- Schalten Sie das BASYS3-Board noch nicht ein! Die Verdrahtung muss grundsätzlich stromlos erfolgen, um das Risiko von Beschädigungen am FPGA zu minimieren.
- Verdrahten Sie auf dem Peripherieboard den Ausgang des Taktgebers (BU3) auf das Pin A16 der Buchsenleiste BU2/JB (Eingangssignal `clk` des Zählers). Zeigen Sie das Taktsignal zusätzlich über eine der Leuchtdioden (BU5) auf dem Peripherieboard an.
- Stecken Sie die externe Siebensegment-Anzeige auf das Steckbrett. Informieren Sie sich anhand des Dokuments **Technischer Anhang** über die Position der Vorwiderstände sowie der Anschlüsse der Siebensegment-Anzeige. **Die Anzeige darf unter keinen Umständen ohne Vorwiderstände betrieben werden!** Schließen Sie erst die gemeinsame Anode an VCC an, verbinden Sie danach die Vorwiderstände mit den Segmentanschlüssen und stellen Sie danach eine Verbindung von den FPGA-Pins auf BU2 zu den Vorwiderständen her.
- Verbinden Sie das BASYS3-Board über ein USB-Kabel mit Ihrem PC und schalten Sie das BASYS3-Board ein.
- Klicken Sie auf **Flow Navigator > Program and Debug > Open Hardware Manager**. Im *Hardware Manager* klicken Sie auf **Open target > Auto Connect**.
- Klicken Sie auf **Program Device**. Übernehmen Sie im Dialog **Program Device** den vorgegebenen Dateinamen für den Bitstream und klicken Sie auf **Program**.

14 Funktionsprüfung auf der Hardware

- Folgen Sie beim Test auf der Hardware dem Testplan. Überprüfen Sie die Ergebnisse anhand der Spezifikation.