

# Vivado-Einführungskurs

## ***Teil 1: Projekterstellung, Simulation, RTL-Analyse, Synthese, Definition von Randbedingungen, Implementierung, Funktionsprüfung auf der Hardware***

Im Rahmen dieses Einführungskurses lernen Sie den Entwurf digitaler Schaltungen mit der Software *Vivado* kennen. Sie legen ein Vivado-Projekt an, fügen das vorgegebene VHDL-Modell einer Zählerschaltung (siehe Abbildung 1) zum Projekt hinzu, simulieren, synthetisieren und implementieren Ihren Entwurf und testen die Schaltung schließlich auf der Laborhardware.

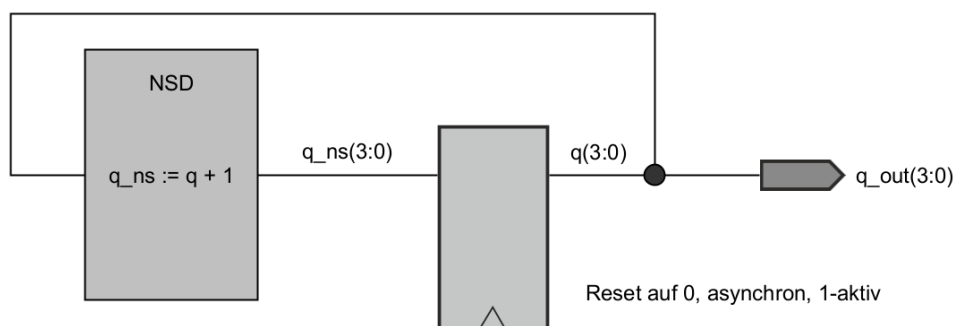


Abbildung 1: RTL-Blockschaltbild des 4-Bit-Zählers

## Inhalt

1	ANLEGEN EINES VIVADO-PROJEKTS .....	2
2	HINZUFÜGEN EINER VHDL-DATEI ZUM PROJEKT .....	5
3	SYNTAXPRÜFUNG .....	6
4	ERSTELLEN EINER TESTBENCH FÜR DIE SIMULATION .....	6
5	SIMULATION .....	9
6	VERIFIKATION DES ZÄHLERS MIT EINER SELBSTTESTENDEN TESTBENCH .....	14
7	ELABORATION / RTL-ANALYSE .....	15
8	SYNTHESE .....	16
9	DEFINITION VON RANDBEDINGUNGEN ( <i>CONSTRAINTS</i> ) .....	18
10	IMPLEMENTIERUNG .....	22
11	ERZEUGEN DES BITSTREAMS .....	23
12	KONFIGURIEREN DES FPGAS .....	23
13	FUNKTIONSPRÜFUNG AUF DER HARDWARE .....	24

# 1 Anlegen eines Vivado-Projekts

- Legen Sie mit Hilfe des Explorers unter `c:\userdata\<Anmeldename>` ein neues Stammverzeichnis für Ihre Vivado-Projekte an, z.B.

```
c:\userdata\musterma\vivado
```

- Erstellen Sie ein weiteres Verzeichnis für VHDL- und andere Vorlagen, so dass z.B. folgende Verzeichnisstruktur entsteht:

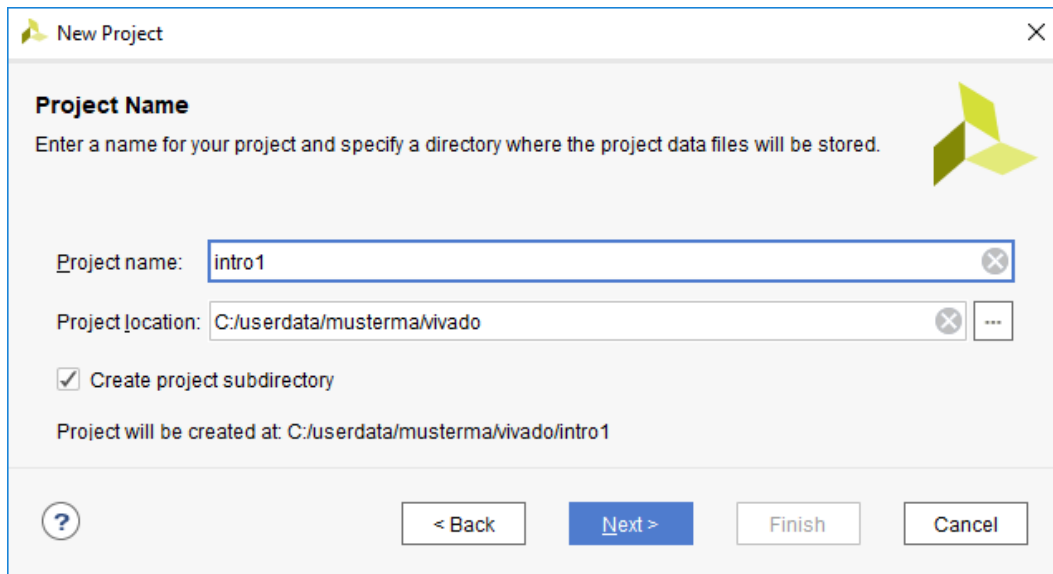
```
c:\
└─ userdata
    └─ musterma
        └─ vivado          Stammverzeichnis für Vivado-Projekte
        └─ vorlagen       Verzeichnis für Vorlagen
```

- Holen Sie die Datei **vorlagen\_v1\_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 über das Windows-Startmenü.



- Legen Sie mit **Create Project...** ein neues Projekt an. Geben Sie Ihrem Projekt einen nachvollziehbaren Namen (z.B. **intro1**). Wählen Sie als *Project location* das zuvor erstellte Stammverzeichnis für die Vivado-Projektdateien und aktivieren Sie die Option **Create project subdirectory**.

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



**New Project**

**Project Name**  
Enter a name for your project and specify a directory where the project data files will be stored.

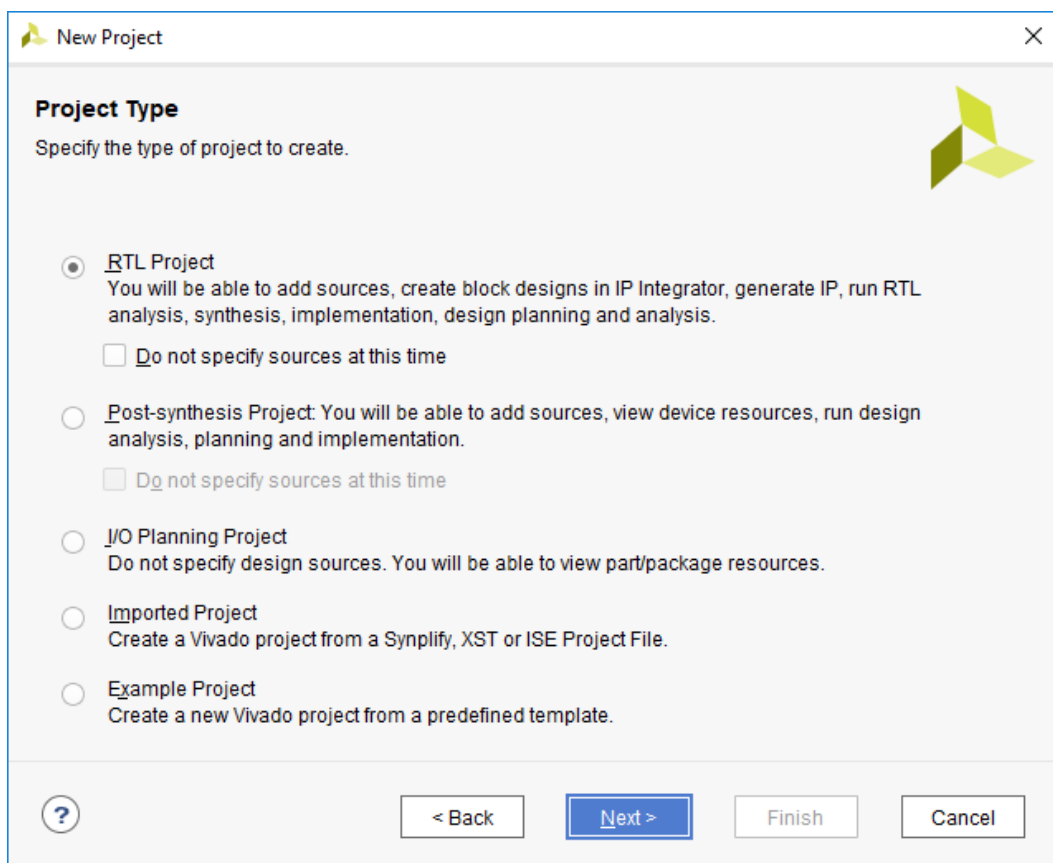
Project name:

Project location:

☒ Create project subdirectory

Project will be created at: C:/userdata/musterna/vivado/intro1

- Bestätigen Sie Ihre Eingabe durch einen Klick auf **Next**.
- Wählen Sie **RTL Project** als Projekttyp und klicken Sie auf **Next**.



**New Project**

**Project Type**  
Specify the type of project to create.

☒ **RTL Project**  
You will be able to add sources, create block designs in IP Integrator, generate IP, run RTL analysis, synthesis, implementation, design planning and analysis.  
☐ Do not specify sources at this time

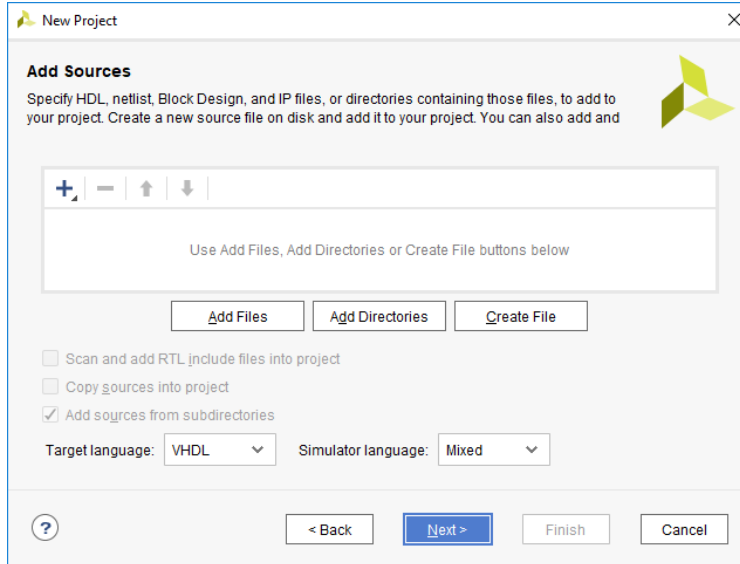
☐ **Post-synthesis Project**: You will be able to add sources, view device resources, run design analysis, planning and implementation.  
☐ Do not specify sources at this time

☐ **I/O Planning Project**  
Do not specify design sources. You will be able to view part/package resources.

☐ **Imported Project**  
Create a Vivado project from a Synplify, XST or ISE Project File.

☐ **Example Project**  
Create a new Vivado project from a predefined template.

- Setzen Sie im Dialog *Add Sources* den Wert von *Target language* auf **VHDL** und dem Wert von *Simulator language* auf **Mixed**. Klicken Sie auf **Next**.



**Add Sources**

Specify HDL, netlist, Block Design, and IP files, or directories containing those files, to add to your project. Create a new source file on disk and add it to your project. You can also add and

Use Add Files, Add Directories or Create File buttons below

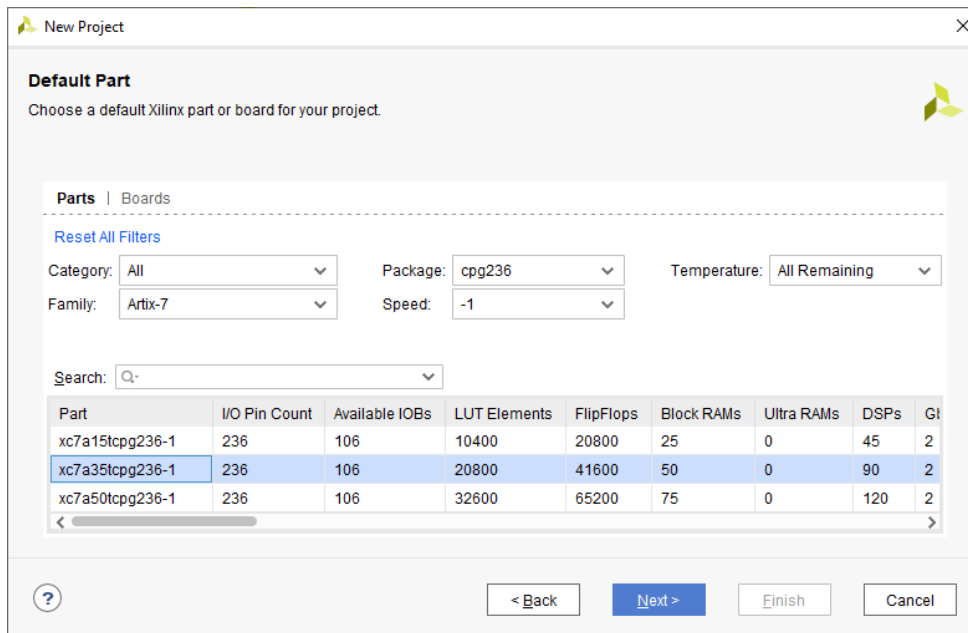
☐ Scan and add RTL include files into project  
☐ Copy sources into project  
☒ Add sources from subdirectories

Target language: **VHDL** Simulator language: **Mixed**

- Klicken Sie im Dialog *Add Constraints* auf **Next**, ohne Änderungen an den Einstellungen vorzunehmen.
- Im Dialog *Default Part* erfolgt die Auswahl des Bausteins, der auf dem *Basys3*-Board des Labors verbaut ist, am schnellsten durch das Setzen der folgenden Einstellungen:

Family: **Artix-7**  
Package: **cpg236**  
Speed grade: **-1**

Wählen Sie aus der Liste den Baustein **xc7a35tcpg236-1**.



**Default Part**

Choose a default Xilinx part or board for your project.

Parts | Boards

[Reset All Filters](#)

Category: **All** Package: **cpg236** Temperature: **All Remaining**

Family: **Artix-7** Speed: **-1**

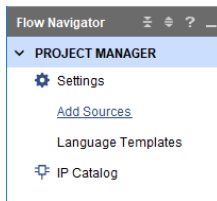
Search:

Part	I/O Pin Count	Available IOBs	LUT Elements	FlipFlops	Block RAMs	Ultra RAMs	DSPs	Gt
xc7a15tcpg236-1	236	106	10400	20800	25	0	45	2
<b>xc7a35tcpg236-1</b>	<b>236</b>	<b>106</b>	<b>20800</b>	<b>41600</b>	<b>50</b>	<b>0</b>	<b>90</b>	<b>2</b>
xc7a50tcpg236-1	236	106	32600	65200	75	0	120	2

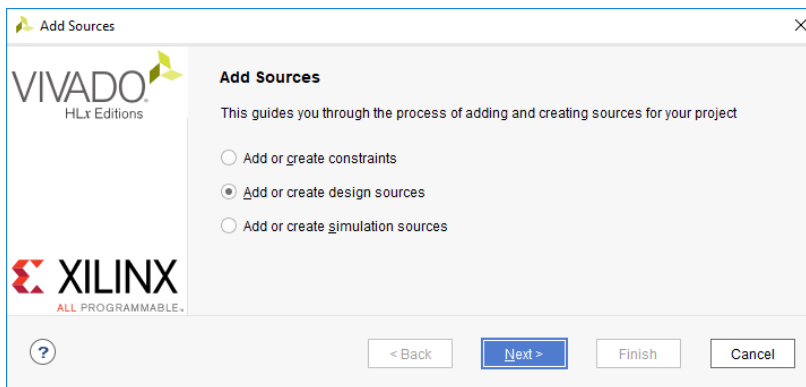
- Schließen Sie die Projekterstellung mit **Next**, bzw. **Finish** im folgenden Dialog ab.

## 2 Hinzufügen einer VHDL-Datei zum Projekt

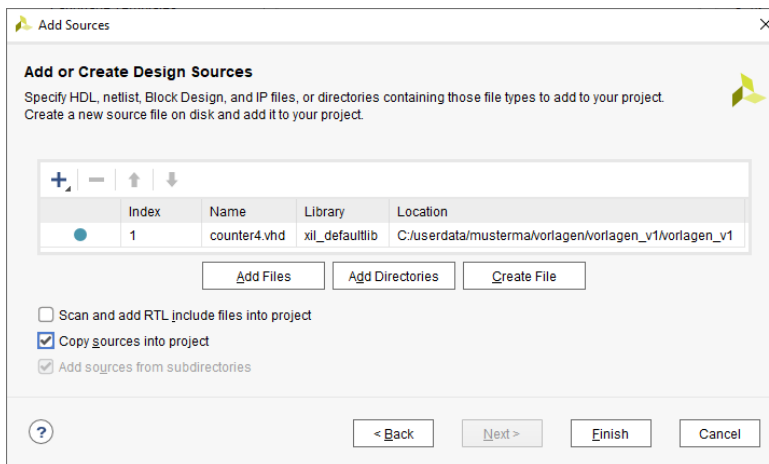
- Klicken Sie im *Flow Navigator* unter *Project Manager* auf **Add Sources**.



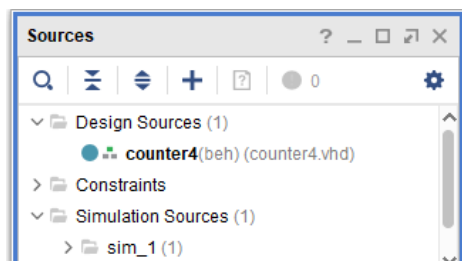
- Wählen Sie im folgenden Dialog **Add or create design sources** und klicken Sie auf **Next**.



- Klicken Sie auf **Add Files...**, navigieren Sie in Ihr Vorlagenverzeichnis, dort in das Unterverzeichnis **vorlagen\_v1** und wählen Sie die Datei **counter4.vhd**. Aktivieren Sie die Option **Copy sources into project** und schließen Sie mit **Finish** ab.

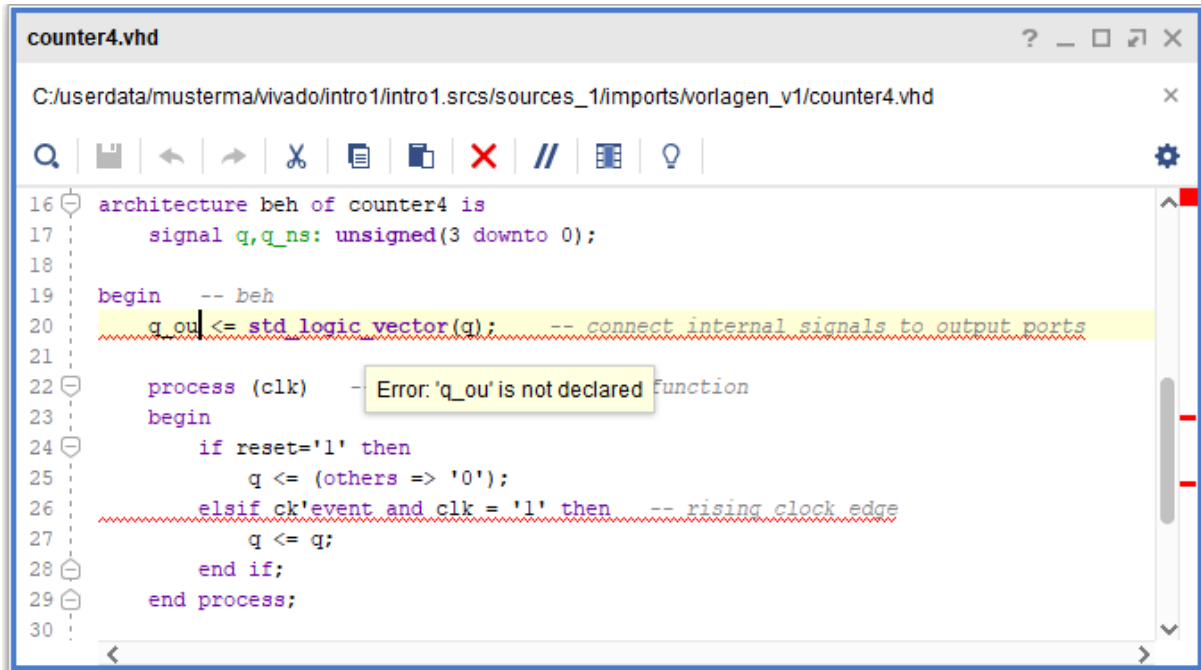


Die Datei `counter4.vhd` liegt jetzt im Panel *Sources* unter *Design Sources*.



### 3 Syntaxprüfung

Öffnen Sie die Datei `counter4.vhd` mit einem Doppelklick und studieren Sie den VHDL-Code. Sie werden feststellen, dass einige Zeilen rot unterstrichen dargestellt sind.



```

counter4.vhd
C:/userdata/musterna/vivado/intro1/intro1.srcs/sources_1/imports/vorlagen_v1/counter4.vhd

16 architecture beh of counter4 is
17     signal q,q_ns: unsigned(3 downto 0);
18
19 begin -- beh
20     q_out <= std_logic_vector(q); -- connect internal signals to output ports
21
22     process (clk) -- Error: 'q_out' is not declared function
23     begin
24         if reset='1' then
25             q <= (others => '0');
26         elsif ck'event and clk = '1' then -- rising clock edge
27             q <= q;
28         end if;
29     end process;
30

```

Bewegen Sie die Maus über die betreffenden Zeilen, lesen Sie die angezeigten Meldungen, korrigieren Sie alle vorhandenen Syntaxfehler und speichern Sie zum Schluss die Datei.

Der Code enthält außerdem einige semantische Fehler, die Sie vorerst bitte nicht korrigieren. Sie werden in den nächsten Kapiteln einige Werkzeuge kennenlernen, mit denen man solche Fehler aufspüren kann.

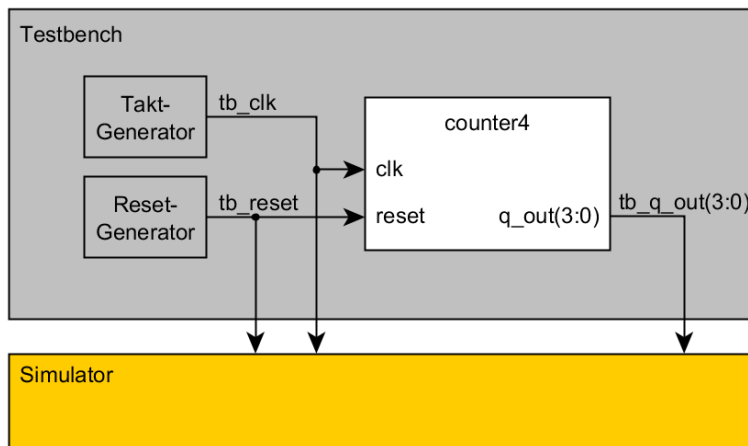
### 4 Erstellen einer Testbench für die Simulation

Das VHDL-Modell des Zählers `counter4` besitzt drei Schnittstellen zur Außenwelt (siehe Entity von `counter4`):

<code>clk</code>	Takteingang
<code>reset</code>	Reset-Eingang
<code>q_out(3:0)</code>	Zähler-Ausgang

Zum Testen dieses Zählers wird eine sog. Testbench verwendet – ein weiteres VHDL-Modell, das alle notwendigen Eingangssignale erzeugt. In unserem Fall handelt es sich dabei um ein Takt- und ein Reset-Signal (`clk` und `reset`). Die Darstellung des zeitlichen Verlaufs der Ein- und Ausgangssignale erfolgt später in einem Simulator (siehe nächster Abschnitt).

In der Testbench werden das Takt- und das Reset-Signal mit Hilfe zweier Prozesse generiert. Das zu testende Modell, die UUT (Unit Under Test) – also der Zähler – ist als hierarchische Komponente eingebunden, d.h. im VHDL-Code der Testbench ist die nur Schnittstelle des Zählers deklariert (`component`) und über eine `port map`-Anweisung werden die Anschlüsse einer Instanz des Zählers mit den Signalen der Testbench verbunden.



Üblicherweise muss man Testbenches in Vivado von Hand schreiben. Der Funktionsumfang von Vivado kann jedoch durch TCL-Skripte erweitert werden. Mit Hilfe eines solchen Skripts lässt sich automatisch eine Vorlage für die Testbench generieren:

- Gehen Sie zum Ausführen des TCL-Skripts in Vivado auf **Tools > Run Tcl Script...** und wählen Sie die Datei **testbench\_generator/generate\_testbench.tcl** aus dem Vorlagenverzeichnis. Eine Vorlage für die Testbench braucht im Allgemeinen nur einmal erzeugt zu werden. Falls im Design bereits eine Testbench gleichen Namens existiert, wird diese überschrieben!
- Vor dem eigentlichen Erstellen der Testbench wird Ihr VHDL-Entwurf elaboriert (Details s. Kapitel 7) und dabei u.a. auf syntaktische Korrektheit geprüft. Einen Fehler erkennen Sie z.B. anhand folgender Ausgabe in der *Tcl Console*:  

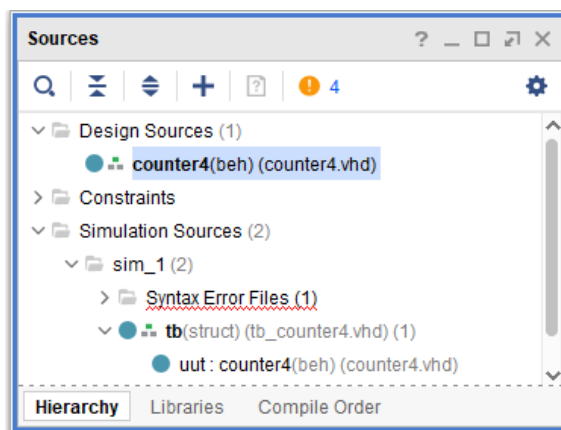
```
ERROR: [Vivado_Tcl 4-5] Elaboration failed - please see the console for details
```

 Scrollen Sie in diesem Fall in der *Tcl Console* ein paar Zeilen nach oben, dort finden Sie eine detaillierte Fehlermeldung. Korrigieren Sie den Fehler und führen Sie das TCL-Skript erneut aus.
- Wurde die Testbench erfolgreich erstellt, erscheint in der *Tcl Console* die Meldung  

```
INFO: Testbench
```

```
C:/userdata/musterna/vivado/intro1/tb_counter4.vhd generated and added to simulation sources.
```

 Die Testbench wird automatisch zum Projekt hinzugefügt. Sie finden die erstellte Datei im Panel *Sources > Simulation Sources > sim\_1*.



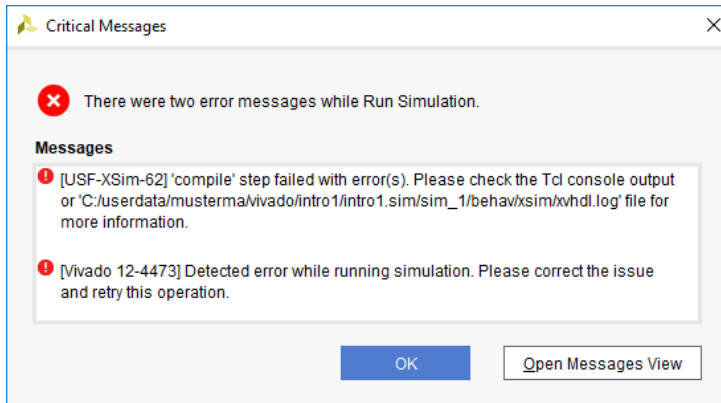
Beachten Sie, dass das Modul *counter4* als Komponente in der Testbench instanziiert wurde und deshalb eine Hierarchieebene unter der Testbench liegt.

- Die Testbench-Vorlage enthält Codeabschnitte, die erst noch von Hand bearbeitet werden müssen (daher auch die Warnmeldung und der Hinweis auf *Syntax Error Files* im Panel *Sources*). Öffnen Sie die Datei `tb_counter4.vhd` mit einem Doppelklick. Sie sehen eine leere Entity, sowie eine Architecture, in der der Zähler `counter4` als Komponente deklariert und instanziiert ist, außerdem Signal- und Konstantendeklarationen und schließlich zwei Prozesse `clk_process` und `stimulus_process`, die später Takt und Reset für den Zähler erzeugen sollen. Im Code sind einige Stellen als fehlerhaft markiert. Das ist im Moment so in Ordnung. Ändern Sie bitte noch nichts, diese Fehler werden Sie später korrigieren.
- Bevor man mit dem Simulieren und Testen startet, sollte man sich darüber Gedanken machen, was und in welchem Umfang getestet werden muss und welche Randbedingungen dabei zu berücksichtigen sind. Studieren Sie daher die VHDL-Quellcodes des Zählers und der Testbench und klären Sie einige Fragen:
  - Wann und wie reagiert der Zähler `counter4` auf Änderungen von Takt und Reset?
  - Wie heißen die Signale in der Testbench (nicht die Ports der Komponente `counter4`!), die an den Takt- und Reseteingang des Zählers angeschlossen sind?
  - In welchem der Testbench-Prozesse wird das Takt-, in welchem das Resetsignal generiert? Beschreiben Sie die Arbeitsweise der beiden Prozesse!
  - Das FPGA auf dem Laborsystem kann extern mit einer Frequenz von 100 MHz getaktet werden (siehe Dokumentation zum BASYS3-Board). Welche Periodendauer des Taktes ergibt sich daraus? Wo könnten Sie diese Periodendauer im Quellcode der Testbench einstellen?
  - Welchen Signalpegel hat das von der Testbench generierte Resetsignal in seiner aktiven Phase? Passt der Pegel des Reset-Impulses zu den Anforderungen der VHDL-Komponente, die Sie testen möchten? Wo und wie können Sie ggf. den Signalpegel des generierten Resetsignals anpassen?
- Ändern Sie nun den Quellcode der Testbench folgendermaßen ab:
  - Die Signale `<clk>` und `<reset>` in den Prozessen `clk_process` und `stimulus_process` sind Platzhalter für die realen Namen der Signale, die an Takt und Reset der Zählerkomponente angeschlossen sind. Ändern Sie die Platzhalter entsprechend ab. Aller Syntaxfehler sollten nun beseitigt sein.
  - Der generierte Takt muss in dieser Simulation nicht unbedingt dem realen Takt entsprechen. In unserem Fall soll er eine Frequenz von 40 MHz und am Beginn der Taktperiode den Wert '0' besitzen. Das Impuls-Pausen-Verhältnis des Takts soll 1:1 betragen.
  - Konfigurieren Sie das Resetsignal so, dass der Zähler ab dem Zeitpunkt 0 s für eine Dauer von zwei Taktperioden zurückgesetzt wird und danach zu zählen beginnt.
- Der Prozess `stimulus_process` ist in Ihrer Testbench nur für die Erzeugung des Reset-Signals zuständig. Das muss aber nicht immer so sein. Wenn Ihre UUT neben Takt und Reset noch weitere Eingänge hätte, könnten Sie in diesem oder weiteren Prozessen die Signalverläufe generieren, die Sie an den Eingängen Ihrer Komponente für den Test benötigen.

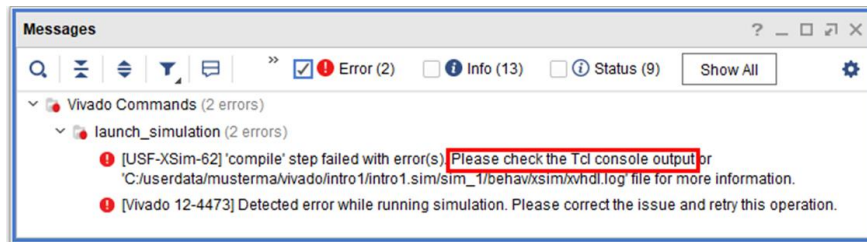


## 5 Simulation

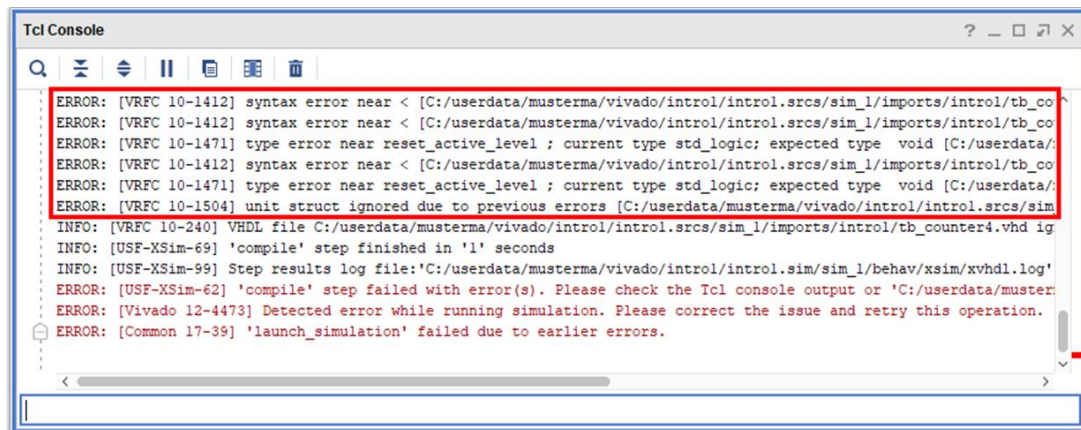
- Klicken Sie zum Starten des Simulators im *Flow Navigator* unter *Simulation* auf **Run Simulation > Run Behavioral Simulation**.
- Sollte Ihr VHDL-Code Fehler enthalten, wird eine entsprechende Meldung angezeigt:



Klicken Sie in diesem Fall auf **Open Messages View**.

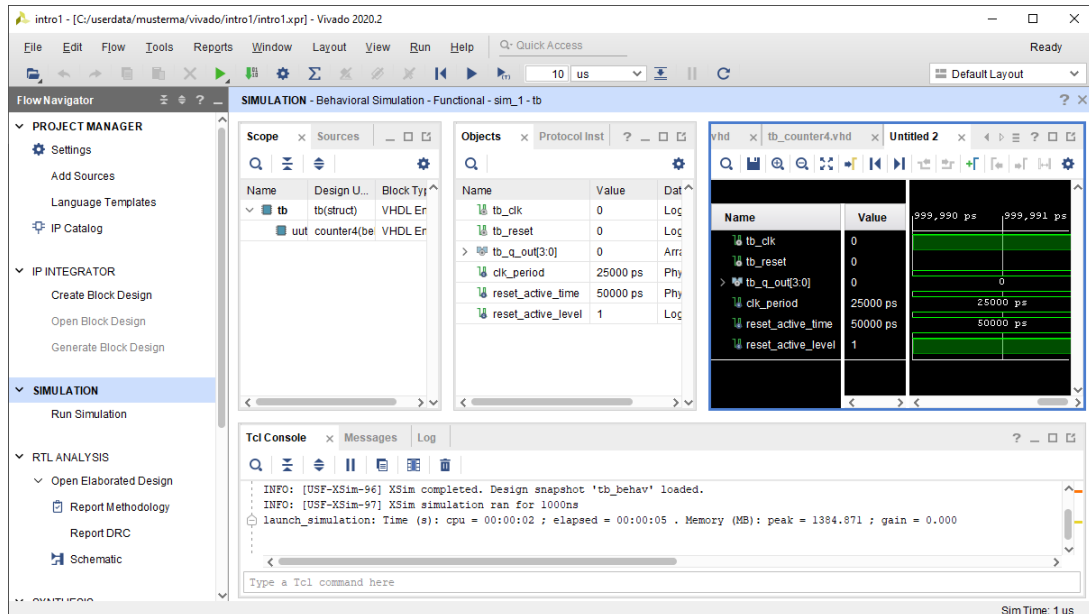


Die dort dargestellte Fehlermeldung verweist Sie auf die Ausgabe der **Tcl Console** (siehe nächste Abbildung).



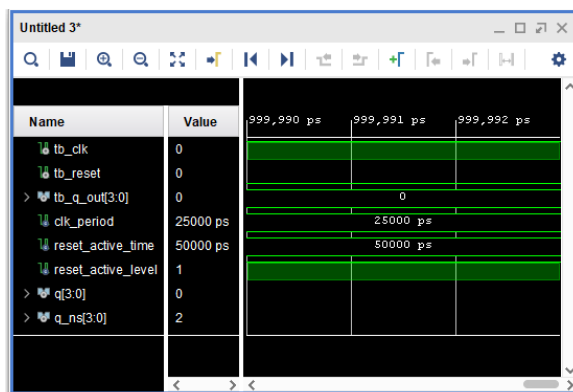
Nach der Korrektur des Fehlers können Sie die Simulation erneut starten.

- Nach dem Start des Simulators wird folgendes Fenster angezeigt:



Unter **Scope** sehen Sie den hierarchischen Aufbau Ihres Entwurfs. Auf der obersten Ebene liegt die Testbench **tb**, darunter der Zähler **counter4** (Instanzname **uut**, siehe Instanziierung der Komponente **counter4** in der Testbench). Für das aktuell unter **Scope** ausgewählte Modul werden unter **Objects** die jeweils definierten Ports, Signale und Konstanten aufgelistet; diese wiederum können per Drag & Drop in die Darstellung der Signalverläufe übernommen werden. Standardmäßig werden die Signalverläufe aller Objekte der obersten Hierarchieebene dargestellt, in diesem Fall also diejenigen der Testbench.

- Um sich die Signale im Inneren des Zählers anschauen zu können, wählen Sie unter **Scope** das Modul **uut** aus und ziehen anschließend die Signale **q(3:0)** und **q\_ns(3:0)** in die Darstellung der Signalverläufe. Für die beiden Signale liegen noch keine Simulationsergebnisse vor, deshalb wird kein Signalverlauf angezeigt.



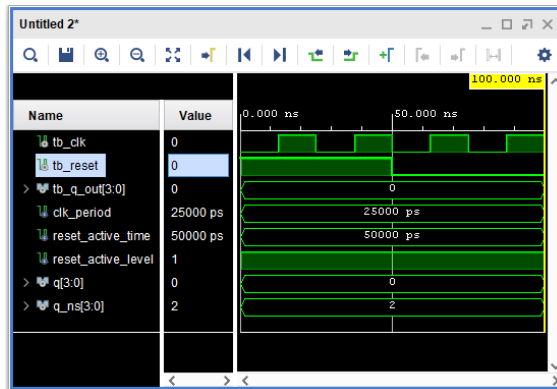
- Setzen Sie den Simulationszeitpunkt auf 0 zurück, indem Sie in der **Tcl Console** das Kommando **restart** eingeben.
- Geben Sie den Befehl **run 100 ns** ein. Der Simulationszeitpunkt stand vorher auf 0, nach diesem Befehl auf 100 ns. Durch eine erneute Eingabe des gleichen Befehls würde die Simulationszeit wieder um 100 ns vorwärtsschreiten, also zum Zeitpunkt 200 ns, 300 ns, usw.

- Oberhalb der Signale finden Sie einige Werkzeuge, mit denen Sie die Darstellung anpassen können:



Wenn Sie die Maus über die Symbole bewegen, erhalten Sie eine kurze Funktionsbeschreibung.

- Klicken Sie mit der Maus auf „Zoom Fit“ und stellen Sie so den kompletten Signalverlauf dar.

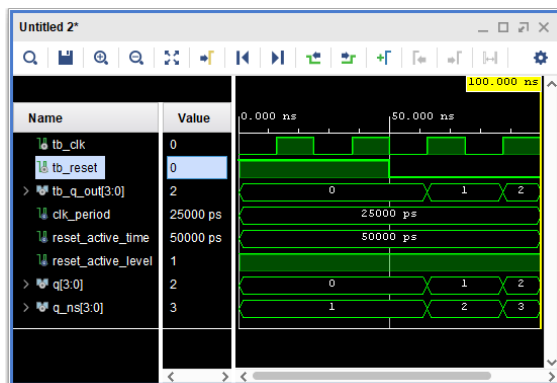


Takt und Reset besitzen die gewünschten Werte, jedoch scheint der Zähler nicht zu funktionieren. Das Ausgangssignal  $q$  des Zählerregisters steht dauerhaft auf 0, das Ausgangssignal  $q\_ns$  des Überführungsschaltnetzes auf 2. Im VHDL-Code des Zählers gibt es also mindestens zwei semantische Fehler.

- Klicken Sie auf den Reiter **Sources** und öffnen Sie die Datei **counter4.vhd**.
  - In welchem Prozess ist das Zählerregister modelliert, in welchem das Überführungsschaltnetz?
  - Wo wird im Registerprozess der vom Überführungsschaltnetz berechnete Wert übernommen?
  - Wo wird der nächste Zählerstand berechnet?

Korrigieren Sie die Fehler und speichern Sie die Datei.

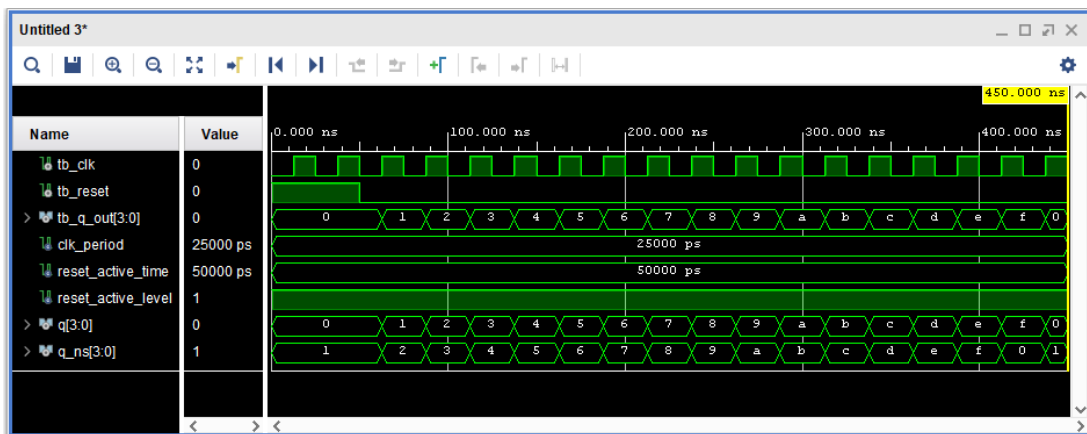
- Da der Quellcode geändert wurde, reicht es nicht, die Simulation mit *restart* auf den Zeitpunkt 0 zurückzusetzen. Sie müssen stattdessen die gesamte Simulation neu starten, indem Sie im Menü **Run** die Option **Relaunch Simulation** wählen. Geben Sie anschließend wieder **restart** und **run 100 ns** an der Kommandozeile der **Tcl Console** ein.



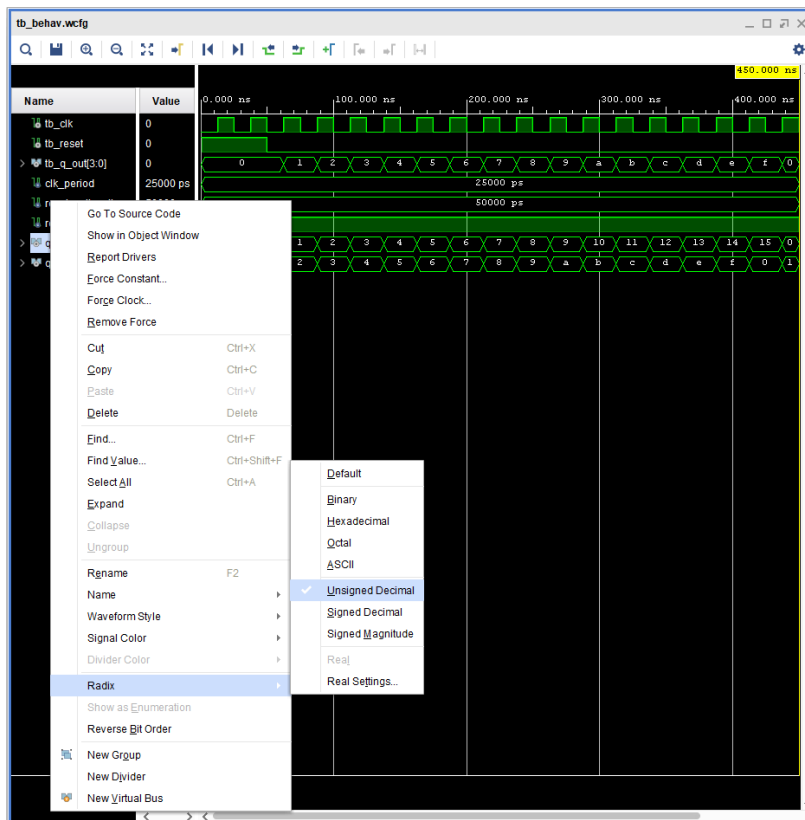
Der Zähler scheint jetzt richtig zu funktionieren. Allerdings reicht eine Simulationszeit von 100 ns nicht aus, um alle Zählschritte des Zählers sichtbar zu machen. Bevor Sie jetzt einfach mehrere Male `run 100 ns` eingeben, denken Sie bitte über folgende Fragen nach:

- Wie viele Bits hat der Zähler? Was ist demnach der größtmögliche Zählerstand?
- Zu welchem Zeitpunkt wird also ein Überlauf des Zählers erfolgen, wenn Sie die Dauer des Resetimpulses mit einkalkulieren?

Geben Sie nun genau einmal den Befehl `run t ns` ein, und wählen Sie `t` dabei so, dass die Simulation ungefähr in der Mitte der Taktperiode nach dem Überlauf des Zählers stehen bleibt.



- Standardmäßig werden alle Signalwerte unter *Value* im Hexadezimalformat angezeigt. Wenn Sie den Zählerstand als Dezimalzahl darstellen möchten, wählen Sie das Signal `q_out[3:0]` mit der rechten Maustaste aus, klicken auf den Eintrag **Radix** und wählen Sie das gewünschte Format **Unsigned Decimal**.



- Der Sinn einer Simulation besteht darin, vor der Implementierung in der Hardware zu prüfen, ob die Schaltung korrekt arbeitet. Überlegen Sie also, ob Ihre Simulation alle relevanten Fälle abdeckt und kontrollieren Sie anhand der Signalverläufe und des RTL-Blockschaltbildes auf der ersten Seite dieser Anleitung, ob Ihr VHDL-Entwurf sich den Anforderungen entsprechend verhält:
  - ☐ Ändert sich der Zählerstand mit jeder steigenden Flanke des Eingangstakts?
  - ☐ Arbeitet der Zähler korrekt (aufsteigend, Eilerschritte)?
  - ☐ Welcher Zählerstand wird einen Takt nach dem Erreichen des höchsten Zählerstands angezeigt (Verhalten bei Überlauf)?
  - ☐ Auf welchen Wert wird der Zähler durch das Reset-Signal zurückgesetzt?
  - ☐ Ist der Reset-Eingang des Zählers 0- oder 1-aktiv (oder anders formuliert: Durch welchen Pegel des Reset-Signals wird der Zähler auf 0 zurückgesetzt)?
- Wie bereits erwähnt, brauchen Sie die Simulation nicht zu beenden, wenn Sie eine VHDL-Datei ändern müssen. Wählen Sie die betreffende Datei im Reiter **Sources** mit einem Doppelklick aus, diese öffnet sich dann in einem integrierten Editor. Über den Menüpunkt **Run > Relaunch Simulation** wird Ihr VHDL-Code erneut übersetzt und die Simulation neu gestartet.
- Beenden Sie den Simulator (nicht Vivado!) nach erfolgreicher Simulation durch einen Klick auf das Kreuz rechts oben im Titelfeld **SIMULATION**. Vivado fragt Sie vor dem Beenden des Simulators, ob Sie die Konfiguration der Signaldarstellung speichern und zum Projekt hinzufügen möchten. Wenn Sie diese Rückfragen mit

OK/Save/Yes bestätigen, werden die Einstellungen beim nächsten Start des Simulators automatisch geladen und angewendet.

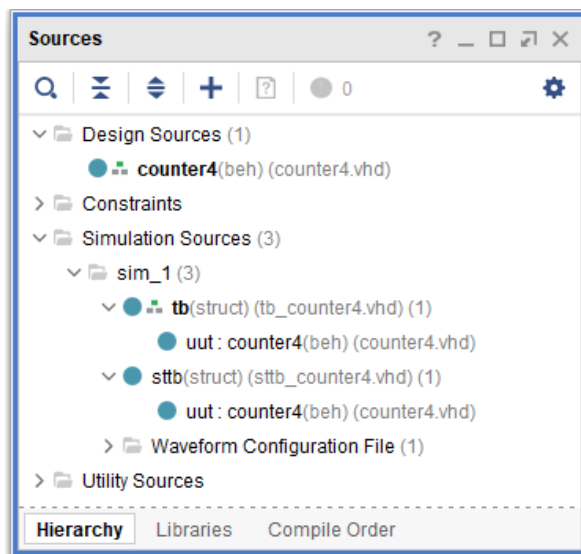
## 6 Verifikation des Zählers mit einer selbsttestenden Testbench

Eine weitere Möglichkeit zur Simulation eines VHDL-Entwurfs ist eine selbsttestende Testbench. Sie legt Stimuli an eine „Unit Under Test“ (UUT) an und überprüft gleichzeitig, ob zu bestimmten Zeitpunkten an den Ausgängen der UUT die erwarteten Werte anliegen. Anstelle der in Kapitel 5 beschriebenen Prüfung von grafisch dargestellten Signalverläufen muss man sich beim Erstellen einer solchen Testbench vor der Simulation einige Gedanken machen:

- Mit welchen Stimuli lassen sich alle relevanten Testfälle abdecken?
- Welche Werte der Ausgangssignale werden zu welchem Zeitpunkt erwartet?

Eine vollständige selbsttestende Testbench enthält die komplette Spezifikation der Schaltung. Mit einer solchen Testbench lassen sich auch komplexe Designs sehr schnell und effizient testen.

- Bei den Vorlagen finden Sie in der Datei `sttb_counter4.vhd` eine Testbench, mit der der Zähler vollständig getestet werden kann. Fügen Sie diese Datei mit **Project Manager > Add Sources > Add or create simulation sources > Add Files** zum Projekt hinzu. Achten Sie dabei darauf, dass die Option **Copy sources into project** aktiviert ist.
- Im Panel *Sources* taucht die neue Datei unter *Simulation Sources* auf:



- Es gibt nun zwei Testbenches, `tb` und `sttb`. Die Testbench `tb` ist fett markiert – sie wird ausgeführt, wenn Sie jetzt eine Simulation starten würden.
- Um `sttb` als Standard-Testbench zu verwenden, klicken Sie mit der rechten Maustaste auf `sttb` und wählen **Set as Top**. Nach dieser Aktion wird `sttb` fett dargestellt und steht in der Auflistung der Testbenches an erster Stelle.
- Öffnen Sie die Datei `sttb_counter4.vhd`, studieren Sie deren Inhalt, finden Sie Unterschiede und Gemeinsamkeiten zwischen den beiden Testbenches, und versuchen Sie zu verstehen, wie die selbsttestende Testbench funktioniert.
- Starten Sie die Simulation nun mit **Simulation > Run Simulation > Run Behavioral Simulation**. Sollte dabei ein Fehler auftreten, informieren Sie sich anhand der

Meldungen in der *Tcl Console* über die Ursache. Liegt der Fehler in Ihrem Entwurf, müssen Sie ihn korrigieren. Gibt es ein Problem mit der Testbench, könnte das ebenfalls an Ihrem Entwurf liegen.

- Setzen Sie nach dem Starten des Simulators die Simulationszeit mit **restart** auf 0 zurück und geben Sie anschließend das Kommando **run -all** ein. Die Simulation läuft nun so lange, bis sie von der selbsttestenden Testbench beendet wird.
- Kontrollieren Sie die Ausgaben in der *Tcl Console*. Die Meldung

```
Note: Test finished. No errors.
```

informiert Sie darüber, dass alle Tests erfolgreich verlaufen sind. Es wird außerdem der Zeitpunkt angezeigt, am dem die Simulation angehalten wurde. Die Meldung

```
Failure: Wrong q_out value.
```

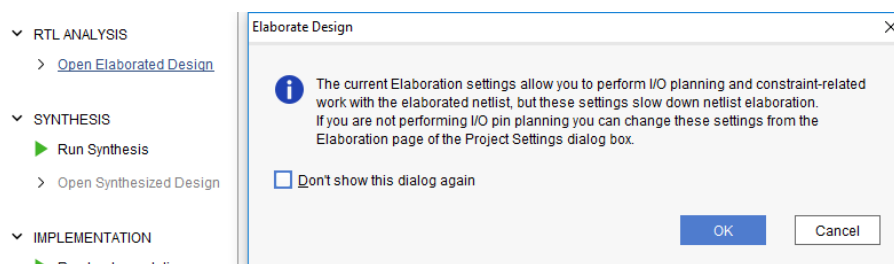
zeigt dagegen an, dass ein Ausgangswert des Zählers zu einem bestimmten Zeitpunkt nicht den erwarteten Wert hatte. Um in diesem Fall weitere Informationen über die Fehlerursache zu erhalten, öffnen Sie die Datei `sttb_counter4.vhd`. Ziehen Sie außerdem diejenigen Signale in die grafische Darstellung der Signalverläufe, deren Verlauf Sie für das Verständnis Ihrer Schaltung benötigen. Geben Sie dann in der *Tcl Console* nacheinander die Kommandos `restart` und `run -all` ein. Im Quellcode der Testbench wird die Zeile farblich hervorgehoben, in der der Fehler entdeckt wurde. Die Darstellung der Signalverläufe endet ebenfalls an der fehlerhaften Stelle. Korrigieren Sie den Fehler und starten Sie die Simulation mit `Run > Relaunch` neu.

- Beenden Sie den Simulator (nicht Vivado!) nach erfolgreicher Simulation durch einen Klick auf das Kreuz rechts oben im Titelfeld *SIMULATION*.

## 7 Elaboration / RTL-Analyse

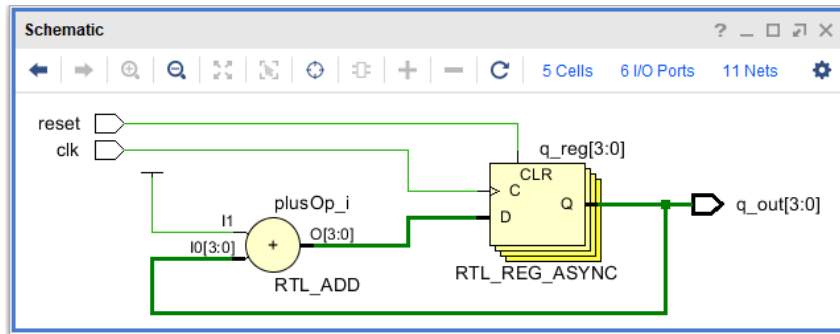
An das Erstellen des Projekts, das Hinzufügen bzw. Editieren von VHDL-Code und die Simulation schließt sich die Elaboration an – die Extraktion der RTL-Struktur aus dem VHDL-Code. Der Code wird analysiert, auf syntaktische Korrektheit geprüft, die hierarchische Struktur aufgelöst und das Design als RTL-Blockschaltbild dargestellt. Anhand dieses Schaltbildes überprüfen Sie, ob die Hardware in VHDL korrekt modelliert wurde: Für jeden Prozess im VHDL-Code muss ein entsprechender RTL-Block vorhanden sein und die Verbindungen der RTL-Blöcke untereinander müssen den Signalen im VHDL-Code entsprechen.

- Klicken Sie im *Flow Navigator* unter *RTL Analysis* auf **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 nun aus Ihrem VHDL-Code folgendes RTL-Blockschaltbild:



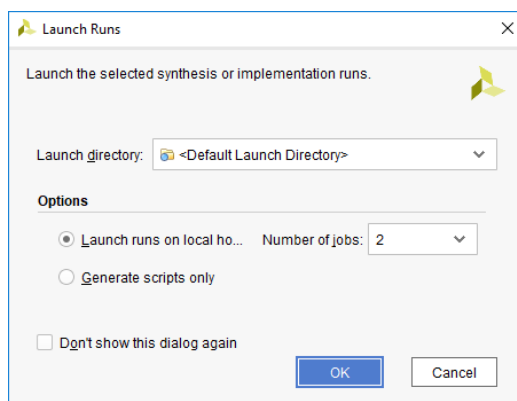


- Überprüfen Sie, ob das generierte Blockschaltbild funktional mit dem RTL-Blockschaltbild auf der ersten Seite dieser Anleitung übereinstimmt:
  - ☐ Welcher der dargestellten Blöcke ist das Register, welcher das Überführungsschaltnetz?
  - ☐ Wie viele Bits hat das Register?
  - ☐ Ist im Überführungsschaltnetz die richtige Funktion implementiert?
  - ☐ Sind die Ein- und Ausgangssignale vollständig und besitzen Sie die richtige Polarität?
- Schließen Sie das elaborierte Design, indem Sie im Titelfalken *ELABORATED DESIGN* rechts oben auf das Kreuz klicken.

## 8 Synthese

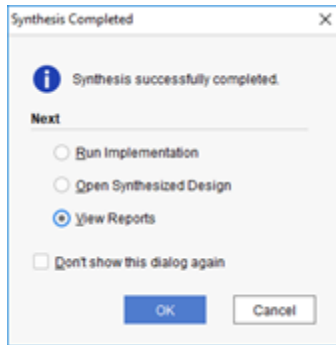
In der Synthese erfolgt die Abbildung des VHDL-Entwurfs auf die Logikelemente, die im FPGA zur Verfügung stehen (Flip-Flops, Look-Up-Tabellen, Multiplexer, RAM). Nebenbei kann die Synthese auch Designfehler aufdecken, die in Syntaxprüfung, Simulation und Elaboration evtl. nicht sichtbar sind (z.B. fehlerhafte Sensitivitätslisten oder unvollständig auskodierte IF-Bedingungen). Es empfiehlt sich also dringend, die Meldungen der Synthese genau zu untersuchen!

- Klicken Sie im *Flow Navigator* unter *Synthesis* auf **Run Synthesis**. Bestätigen Sie den Dialog *Launch Runs* mit **OK**. Während die Synthese läuft, erscheint rechts oben im Fenster die Meldung *Running synth\_design*.



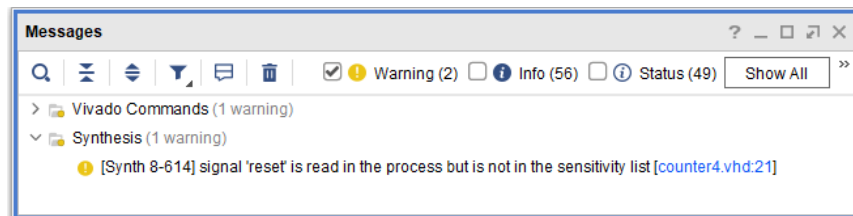
- Sofern der Entwurf noch Fehler enthält, wird die Synthese abgebrochen. Korrigieren Sie in diesem Fall den Fehler, führen Sie erst eine Simulation und eine Elaboration durch und starten Sie dann wieder die Synthese.
- Klicken Sie nach dem Abschluss der Synthese auf **View Reports**.





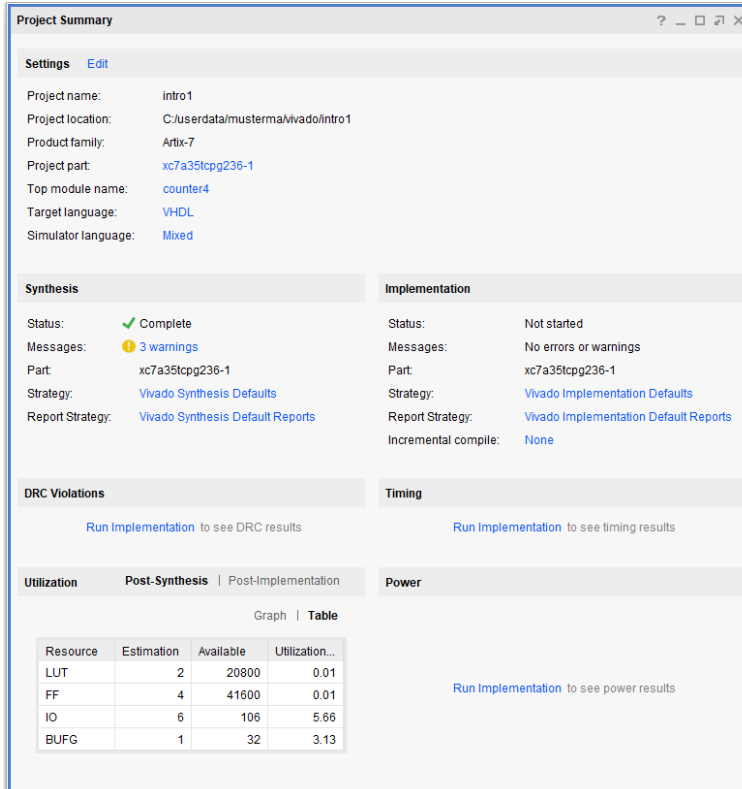
- Überprüfen Sie die Meldungen im Reiter **Messages**. Entfernen Sie das Häkchen bei **Info**, um nur Warnungen anzuzeigen. Klicken Sie außerdem auf das Mülleimer-Symbol, um evtl. veraltete Meldungen aus früheren Syntheseläufen zu löschen.

Es wird eine Warnung angezeigt, die sich auf eine fehlerhafte Sensitivitätsliste in der Datei `counter4.vhd` bezieht:



Die Warnungen und Fehlermeldungen der Synthese sollten Sie keinesfalls ignorieren! Klären Sie folgende Fragen, bevor Sie den Code editieren:

- ☐ Welcher Prozess des Zählers muss sensitiv auf das Reset-Signal sein?
- Korrigieren Sie den Fehler, simulieren und elaborieren Sie Ihren Entwurf, und starten Sie erst dann wieder die Synthese.
- Wählen Sie den Reiter *Project Summary* und klicken Sie unter *Utilization* auf **Table**. Kontrollieren Sie, ob die Anzahl der Flip-Flops, sowie der Ein- und Ausgänge (IO) jeweils der erwarteten Anzahl entsprechen.



**Project Summary**

**Settings** [Edit](#)

Project name: intro1  
Project location: C:/userdata/musterna/vivado/intro1  
Product family: Artix-7  
Project part: xc7a35tcbg236-1  
Top module name: counter4  
Target language: VHDL  
Simulator language: Mixed

**Synthesis**

Status: ✔ Complete  
Messages: ! 3 warnings  
Part: xc7a35tcbg236-1  
Strategy: Vivado Synthesis Defaults  
Report Strategy: Vivado Synthesis Default Reports

**Implementation**

Status: Not started  
Messages: No errors or warnings  
Part: xc7a35tcbg236-1  
Strategy: Vivado Implementation Defaults  
Report Strategy: Vivado Implementation Default Reports  
Incremental compile: None

**DRC Violations**

[Run Implementation](#) to see DRC results

**Timing**

[Run Implementation](#) to see timing results

**Utilization** **Post-Synthesis** | Post-Implementation

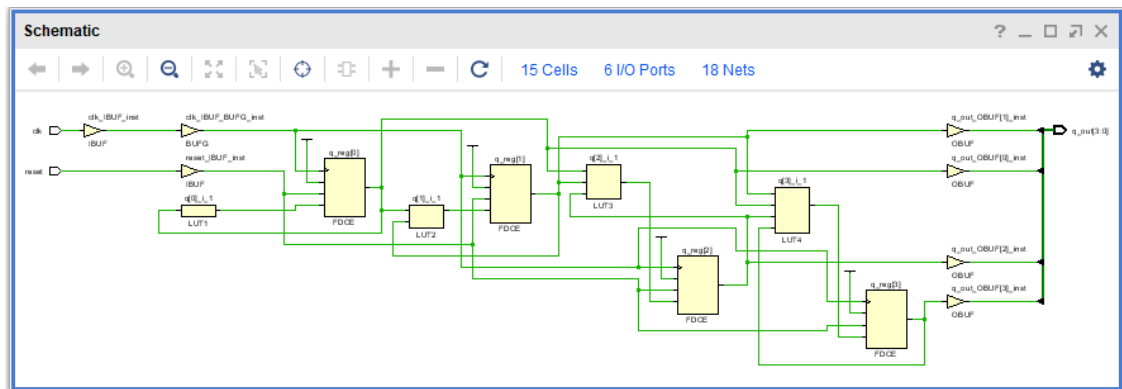
[Graph](#) | [Table](#)

Resource	Estimation	Available	Utilization...
LUT	2	20800	0.01
FF	4	41600	0.01
IO	6	106	5.66
BUFG	1	32	3.13

[Run Implementation](#) to see power results

In diesem Fall werden genau die vier Flip-Flops erzeugt, die der 4-Bit-Zähler benötigt. Auch die Anzahl der Ein- und Ausgänge ist korrekt (Takt + Reset + vier Ausgänge).

- Klicken Sie im *Flow Navigator* auf **Synthesis > Open Synthesized Design > Schematic**.



Der Schaltplan zeigt das Syntheseergebnis mit I/O-Buffern, Flip-Flops (FDCE) und Look-Up-Tabellen (LUTx).

- Schließen Sie das Fenster mit den Syntheseergebnissen durch einen Klick auf das Kreuz rechts oben im Titelbalken **Synthesized Design**.

## 9 Definition von Randbedingungen (*Constraints*)

Bis jetzt haben Sie nur einen 4-Bit-Zähler in VHDL beschrieben und in einer Simulation nachgewiesen, dass die Schaltung funktioniert. Außerdem hat die RTL-Analyse gezeigt, dass die im VHDL modellierte Hardware den Erwartungen entspricht und auf die Logikelemente im Inneren des FPGAs abgebildet werden kann. Um nun den Zähler auch

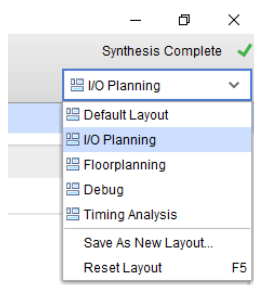
tatsächlich auf der Hardware des Laborsystems testen zu können, müssen Sie die Anschlüsse des Zählers mit realer Hardware verbinden: Den Takteingang mit einem Taktgenerator, den Reseteingang mit einem Taster und die Zählerausgänge mit Leuchtdioden.

Das Verbinden der Zählerschaltung im Inneren des FPGAs mit der Hardware auf dem Laborsystem erfolgt an zwei Stellen:

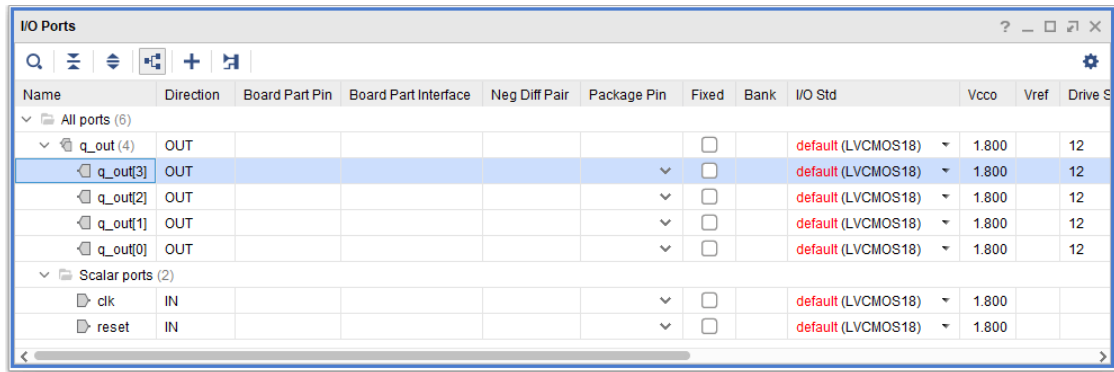
1. Außerhalb des FPGA-Gehäuses verbinden fest verdrahtete Leitungen (z.B. Taster BTNC, Leuchtdioden) oder freie Verdrahtung (z.B. Takt auf der Peripherieplatine) den jeweiligen FPGA-Anschluss (FPGA-Pin) und die Hardwarekomponente.
2. Die FPGA-interne Verdrahtung zwischen den Ein- und Ausgängen (I/O-Ports) der Schaltung und den FPGA-Pins wird über sogenannte Randbedingungen (*Constraints*) festgelegt. Auch die Frequenz des Systemtakts muss der Synthese über Constraints mitgeteilt werden. Theoretisch könnten Constraints an jeder Stelle im Entwurfsablauf definiert werden. Es kann jedoch sein, dass Hardware aufgrund eines Designfehlers von der Synthese teilweise oder sogar komplett wegoptimiert wird. Dies kann u.U. auch die externen Schnittstellen der Schaltung betreffen. Definiert man die Zuordnung von FPGA-Pins erst nach der Synthese, sieht man sofort aufgrund der fehlenden Pins, dass bei der Synthese etwas schiefgegangen ist.

Dies ist der ideale Zeitpunkt für einen Blick in das Dokument „**Technischer Anhang**“, Kapitel „Taster und Schiebeschalter“, „Leuchtdioden“ und „Taktgeneratoren“, sowie in das „**Basys 3 FPGA Board Reference Manual**“, Kapitel „Basic I/O“.

- Informieren Sie sich anhand der genannten Dokumente darüber, mit welchen FPGA-Pins die Ein- und Ausgänge verbunden werden müssen.
  - Als Reset-Taster wird der Taster BTNC auf dem BASYS3-Board verwendet.
  - Der Zählertakt kommt vom Taktgenerator auf der Peripherieplatine.
  - Der Zählerstand soll mit Hilfe der Leuchtdioden LD3...LD0 auf dem BASYS3-Board angezeigt werden.
- Sofern das synthetisierte Design (*Synthesized Design*) nicht geöffnet ist, klicken Sie im *Flow Navigator* unter *Synthesis* auf **Open Synthesized Design**.
- Wählen Sie rechts oben in der Werkzeugleiste die Ansicht **I/O Planning**.



- Erweitern Sie im unteren Fensterbereich im Reiter **I/O Ports** die Einträge **q\_out** und **Scalar ports**.

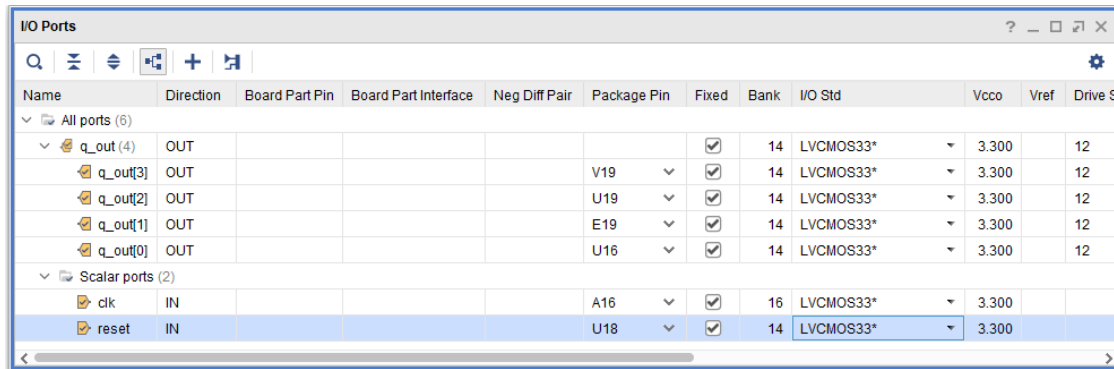


Name	Direction	Board Part Pin	Board Part Interface	Neg Diff Pair	Package Pin	Fixed	Bank	I/O Std	Vcco	Vref	Drive S
q_out (4)	OUT					<input type="checkbox"/>		default (LVCMOS18)	1.800		12
q_out[3]	OUT					<input type="checkbox"/>		default (LVCMOS18)	1.800		12
q_out[2]	OUT					<input type="checkbox"/>		default (LVCMOS18)	1.800		12
q_out[1]	OUT					<input type="checkbox"/>		default (LVCMOS18)	1.800		12
q_out[0]	OUT					<input type="checkbox"/>		default (LVCMOS18)	1.800		12
Scalar ports (2)											
clk	IN					<input type="checkbox"/>		default (LVCMOS18)	1.800		
reset	IN					<input type="checkbox"/>		default (LVCMOS18)	1.800		

- Tragen Sie unter **Package Pin** die Pinnummern ein, an die die jeweiligen Signale angeschlossen werden sollen.

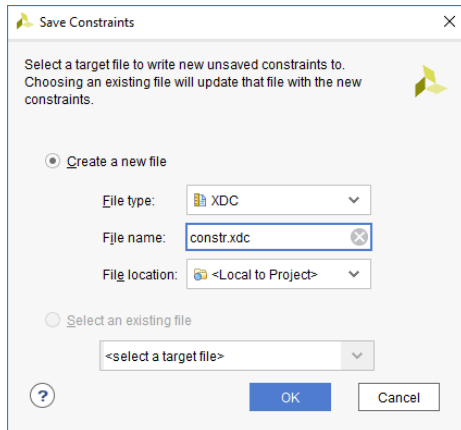
Signalname	Pin-Nummer (Package Pin)	Beschreibung
q_out[3]	V19	Leuchtdiode LD3 auf BASYS3-Board
q_out[2]	U19	Leuchtdiode LD2 auf BASYS3-Board
q_out[1]	E19	Leuchtdiode LD1 auf BASYS3-Board
q_out[0]	U16	Leuchtdiode LD0 auf BASYS3-Board
clk	A16	CCIO-Eingang, angesteuert von Taktgenerator auf Peripherieplatine
reset	U18	Taster BTNC auf BASYS3-Board

- Das Board arbeitet mit einer Spannung von 3,3 Volt. Ändern Sie deshalb bei allen Signalen den Wert in der Spalte **I/O Std** auf **LVCMOS33**.

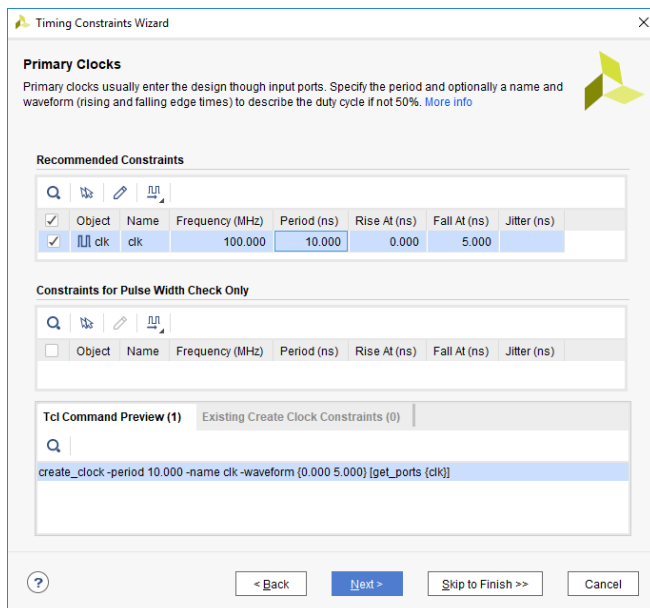


Name	Direction	Board Part Pin	Board Part Interface	Neg Diff Pair	Package Pin	Fixed	Bank	I/O Std	Vcco	Vref	Drive S
q_out (4)	OUT					<input checked="" type="checkbox"/>	14	LVCMOS33*	3.300		12
q_out[3]	OUT				V19	<input checked="" type="checkbox"/>	14	LVCMOS33*	3.300		12
q_out[2]	OUT				U19	<input checked="" type="checkbox"/>	14	LVCMOS33*	3.300		12
q_out[1]	OUT				E19	<input checked="" type="checkbox"/>	14	LVCMOS33*	3.300		12
q_out[0]	OUT				U16	<input checked="" type="checkbox"/>	14	LVCMOS33*	3.300		12
Scalar ports (2)											
clk	IN				A16	<input checked="" type="checkbox"/>	16	LVCMOS33*	3.300		
reset	IN				U18	<input checked="" type="checkbox"/>	14	LVCMOS33*	3.300		

- Speichern Sie die Randbedingungen mit **File > Constraints > Save**. Bestätigen Sie die Meldung, dass die Synthesergebnisse anschließend möglicherweise veraltet sein werden, mit **OK**. Geben Sie der Datei im folgenden Dialog den Namen **constr.xdc** (Name ist frei wählbar, Erweiterung **.xdc** empfehlenswert).



- Zur Definition der Randbedingung für den Systemtakt klicken Sie im *Flow Navigator* unter *Synthesis* auf **Constraints Wizard**.
- Klicken Sie im Startfenster des *Timing Constraints Wizard* einmal auf **Next**. Im folgenden Dialog können Sie die Frequenz des Taktsignals `clk` eingeben. Tragen Sie unter **Recommended Constraints > Frequency (MHz)** den Wert 100 ein. Aufgrund dieser Einstellung wird die Synthese die Schaltung so optimieren, dass sie mit einer Frequenz von 100 MHz betrieben werden kann. Beim Test auf der Hardware schließen Sie später einen Takt mit einer Frequenz von wenigen Hz an, dementsprechend hätte man auch die Randbedingung anders formulieren können. Allerdings gibt es auf dem System auch noch einen 100 MHz-Taktgenerator...



- Überspringen Sie alle weiteren Dialogfenster mit **Skip to Finish >>** und schließen Sie auch den letzten Dialog mit **Finish**.
- Beenden Sie das Definieren von Constraints durch einen Klick auf das Kreuz rechts oben im Titelfeld **SYNTHESIZED DESIGN**.
- Im *Project Manager* sehen Sie unter **Sources > Constraints** die Datei `constr.xdc`. Öffnen Sie die Datei mit einem Doppelklick. Die Datei enthält Kommandos zum Erzeugen der oben definierten Constraints. Als Alternative zum oben beschriebenen Weg über die grafische Definition der Randbedingungen könnte eine solche Datei also auch sehr viel einfacher von Hand editiert und mit *Add Sources > Add or create constraints* zum Projekt hinzugefügt werden.

```

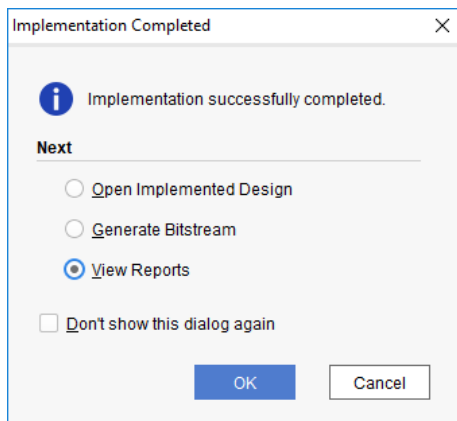
set_property PACKAGE_PIN V19 [get_ports {q_out[3]}]
set_property PACKAGE_PIN U19 [get_ports {q_out[2]}]
set_property PACKAGE_PIN E19 [get_ports {q_out[1]}]
set_property PACKAGE_PIN U16 [get_ports {q_out[0]}]
set_property PACKAGE_PIN A16 [get_ports clk]
set_property PACKAGE_PIN U18 [get_ports reset]
set_property IOSTANDARD LVCMOS33 [get_ports {q_out[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {q_out[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {q_out[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {q_out[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports clk]
set_property IOSTANDARD LVCMOS33 [get_ports reset]
create_clock -period 10.000 -name clk -waveform {0.000 5.000}
[get_ports clk]

```

## 10 Implementierung

Die Implementierungswerkzeuge von Vivado weisen die Look-Up-Tabellen, Flip-Flops usw. an konkrete Elemente im FPGA-Baustein zu und verdrahten sie miteinander.

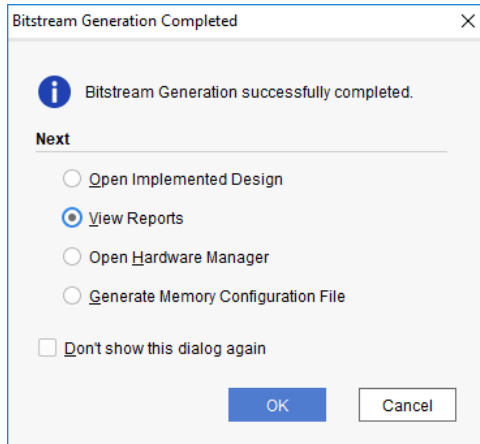
- Klicken Sie im *Flow Navigator* unter *Implementation* auf **Run Implementation**. Die Definition von Constraints hat Einfluss auf das Syntheseresultat. Sie werden deshalb darauf hingewiesen, dass die Syntheseresultate veraltet sind. Bestätigen Sie einen erneuten Syntheselauf mit **Yes**.
- Der Implementierungsvorgang dauert u.U. einige Minuten. Beachten Sie die Statusmeldungen am rechten oberen Rand des Fensters, im Reiter *Design Runs* und in der *Tcl Console*.
- Sofern der Entwurf noch Fehler enthält, wird die Implementierung abgebrochen. Korrigieren Sie in diesem Fall den Fehler, führen Sie erst eine Simulation durch und starten Sie dann wieder den Synthese- und Implementierungslauf.
- Klicken Sie nach dem Abschluss der Implementierung auf **View Reports** und überprüfen Sie die Warn- und Fehlermeldungen im Reiter **Messages**.



## 11 Erzeugen des Bitstreams

Bevor das FPGA konfiguriert werden kann, müssen die Ergebnisse des Implementierungslaufs in einen sogenannten *Bitstream* (eine Konfigurationsdatei für das FPGA) umgewandelt werden.

- Klicken Sie im *Flow Navigator* unter *Program and Debug* auf **Generate Bitstream**.
- Wählen Sie nach dem Generieren des Bitstreams die Option **View Reports**.

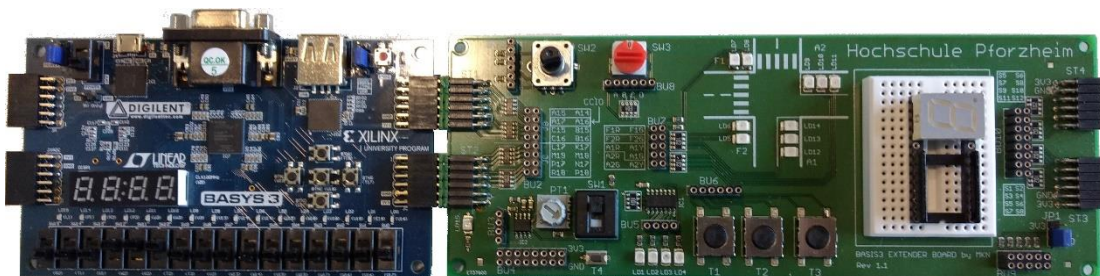


Kontrollieren Sie im Reiter **Messages**, ob der Vorgang erfolgreich abgeschlossen wurde. Ignorieren Sie die Warnmeldung, die Sie darauf hinweist, dass zwei Konfigurationsparameter nicht gesetzt sind (CFGBVS, CONFIG\_VOLTAGE).



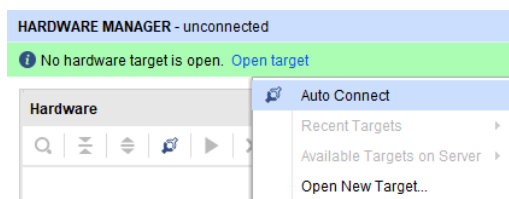
## 12 Konfigurieren des FPGAs

- Schalten Sie das BASYS3-Board noch nicht ein! Die Verdrahtung muss grundsätzlich stromlos erfolgen, um das Risiko von Beschädigungen an den Bauteilen zu minimieren.

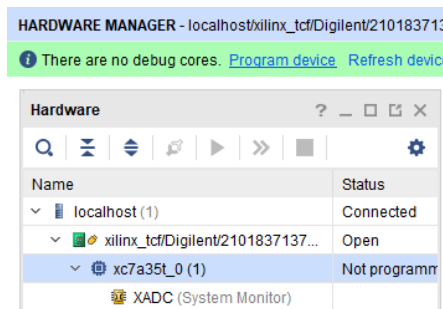




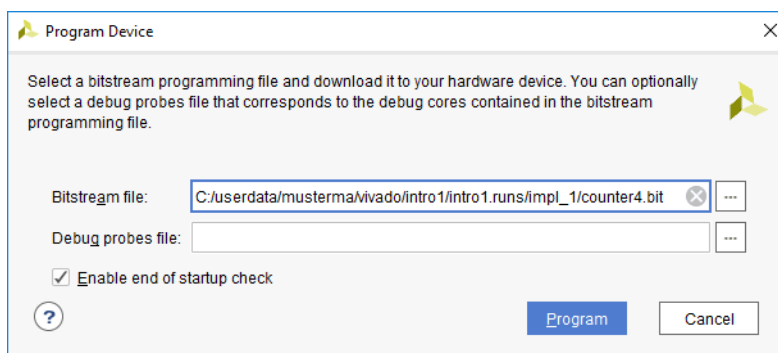
- Die Ansteuerung des Zählers (Signal `clk`) erfolgt durch den Taktgenerator auf dem Peripherieboard, der zwischen monostabilem und astabilem Modus umgeschaltet werden kann (siehe separate Dokument *Technischer Anhang*). Verdrahten Sie deshalb auf dem Peripherieboard den Ausgang des Taktgebers (BU3) auf das Pin A16 der Buchsenleiste BU2/JB. Zeigen Sie das Taktsignal zusätzlich über eine der Leuchtdioden auf dem Peripherieboard an.
- Verbinden Sie das BASYS3-Board über ein USB-Kabel mit Ihrem PC und schalten Sie das BASYS3-Board ein.
- Klicken Sie im *Flow Navigator* unter *Program and Debug* auf **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**.



## 13 Funktionsprüfung auf der Hardware

- Überprüfen Sie mit Hilfe der Komponenten auf der Laborhardware, ob der Zähler korrekt arbeitet. Im Prinzip stellen sich hier die gleichen Fragen wie bei der Simulation:
  - ☐ Ändert sich der Zählerstand mit jeder steigenden Flanke des Eingangstakts?
  - ☐ Arbeitet der Zähler korrekt (aufsteigend, Eilerschritte)?
  - ☐ Welcher Zählerstand wird einen Takt nach dem Erreichen des höchsten Zählerstands angezeigt (Verhalten bei Überlauf)?



- ☐ Auf welchen Wert wird der Zähler durch das Reset-Signal zurückgesetzt?
- ☐ Stimmt die Polarität des Reset-Signals?
- Wenn Sie das BASYS3-Board aus- und wieder einschalten, müssen Sie das FPGA erneut konfigurieren (SRAM-Technologie).