Einführung in Xilinx Webpack ISE 10.1

Diese Version beschreibt sowohl die Benutzung
des Spartan2 als auch des Spartan3

Version Oktober 2010

Urs Graf

1 Installation

Laden Sie **ISE WebPACK - Single File Download** von der Xilinx-Homepage herunter (Registration erforderlich):

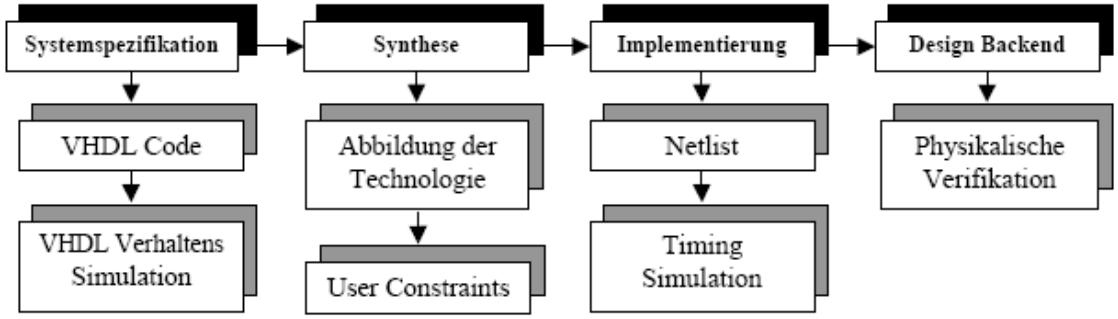
- ***http://www.xilinx.com/ise/logic_design_prod/webpack.htm***

ACHTUNG: Die Installationspfade dürfen keine Leerzeichen enthalten!

Installieren Sie den WebPack indem Sie den Installationsanweisungen folgen.

2 Was ist das Webpack?

Das Softwarepaket Webpack setzt sich aus mehreren Komponenten zusammen. Dies ist zunächst das Webpack Design Entry (***Projekt Navigator***). Als zweite wichtige Komponente gibt es das Simulationstool ***ModelSim XE Starter*** und dazu kommen dann noch viele zusätzliche (kleine) Komponenten, wie z.B. der ***HDL Bencher*** und der ***Chipviewer***. Mit dem ***HDL Bencher*** kann auf einfache Weise eine Testbench erstellt werden und der ***Chipviewer*** visualisiert das PinOut und die benutzten Macrozellen des Bausteins. Mit dem Webpack Projekt Navigator in Zusammenarbeit mit ModelSim XE ist es möglich einen kompletten Workflow für die Erstellung z.B. eines FPGA oder eines CPLD Bausteins zu durchlaufen.



2.1 Welche Hardware wird vom Webpack unterstützt?

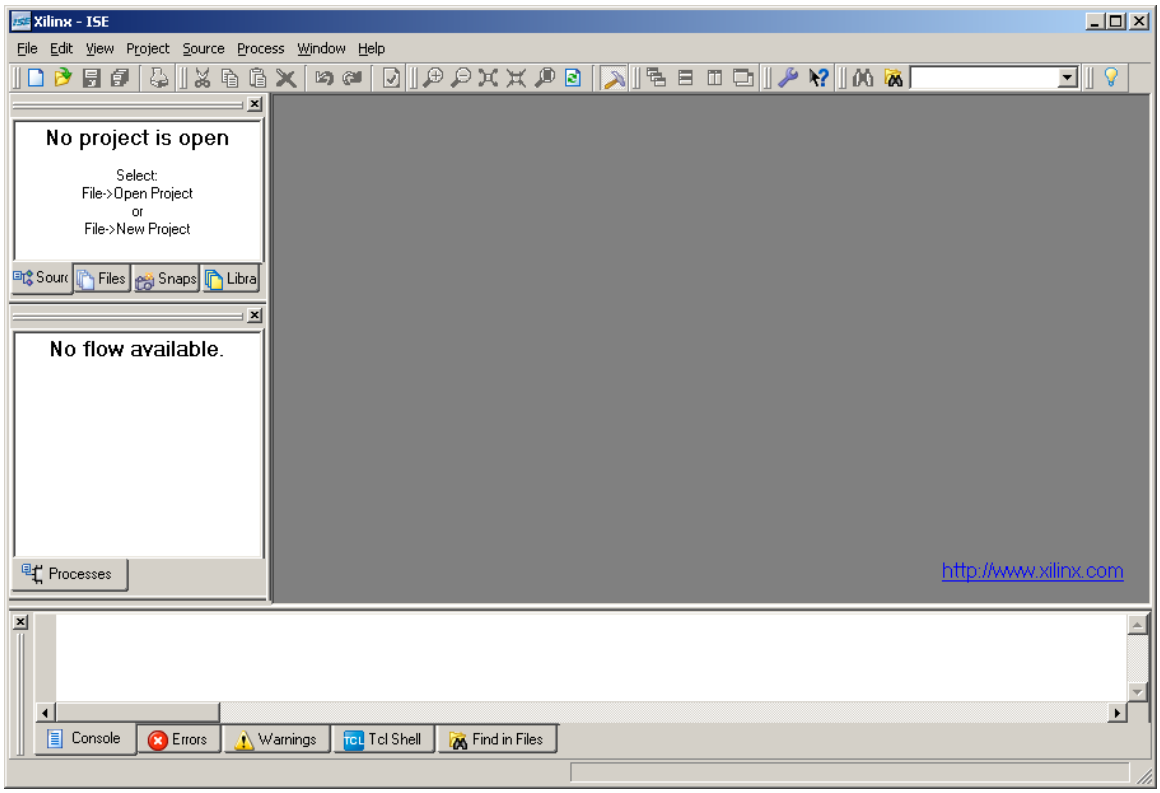
Beim aktuellen Release des ***Xilinx Webpacks***, welches auch in diesem Dokument benutzt wird, werden folgende Baustein unterstützt:

- Virtex-4
- Virtex-E
- Virtex™
- Spartan-3
- Spartan-II
- XC9500
- CoolRunner

Eine komplette Liste finden Sie unter diesem Link.

3 Xilinx Project Navigator

Starten Sie den Project Navigator über das Startmenü. Die nächste Abbildung zeigt den gestarteten ***Project Navigator***. Bei wiederholtem Öffnen, wird im ***Project Navigator*** das Projekt angezeigt, welches zuletzt bearbeitet wurde.



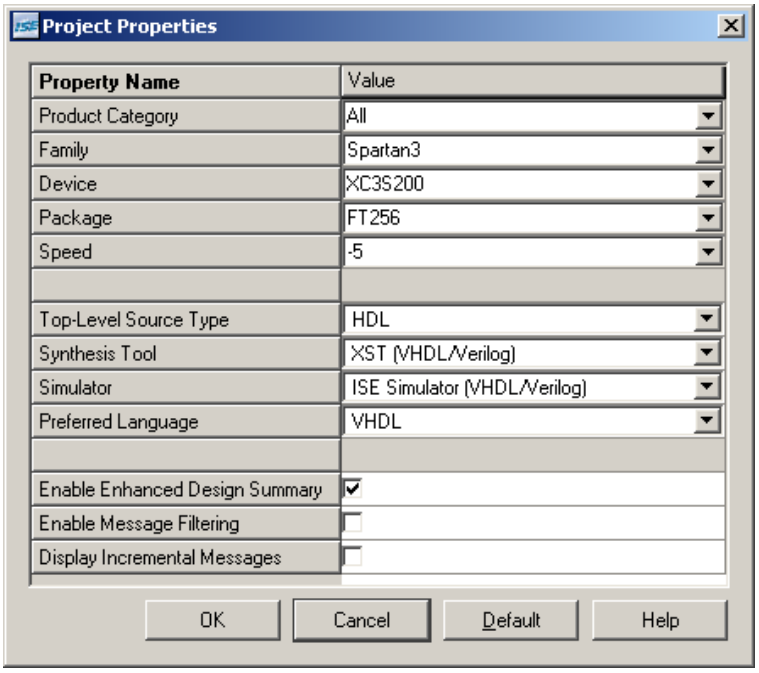
3.1 Erzeugen eines neuen Projektes

Zu Beginn muss ein neues Projekt erstellt werden, dazu den Menupunkt ***File → New Project*** ausführen. Geben Sie Ihrem Projekt einen sinnvollen Namen (z.B. S4BitCounter) und wählen Sie einen Speicherort. Speichern Sie ihre Projekte in Ordnern, die keine Leerzeichen haben, ab. Der ***Xilinx Projekt Navigator*** hat Mühe mit Ordnernamen welche Leerzeichen beinhalten. Als ***"Top-Level Source Type"*** wählen Sie ***"Schematic"*** aus, falls Sie eine Schaltplaneingabe wünschen. Die Verwendung einer Hardware-Beschreibungssprache für bestimmte Module Ihres Designs ist bei dieser Option trotzdem möglich. Die Option ***"HDL"*** ist sinnvoll, wenn Sie Ihr Design komplett in Verilog oder VHDL erstellen wollen und ganz auf eine Schaltplaneingabe verzichten möchten. In unserem Fall wählen Sie die Option ***„HDL“***.

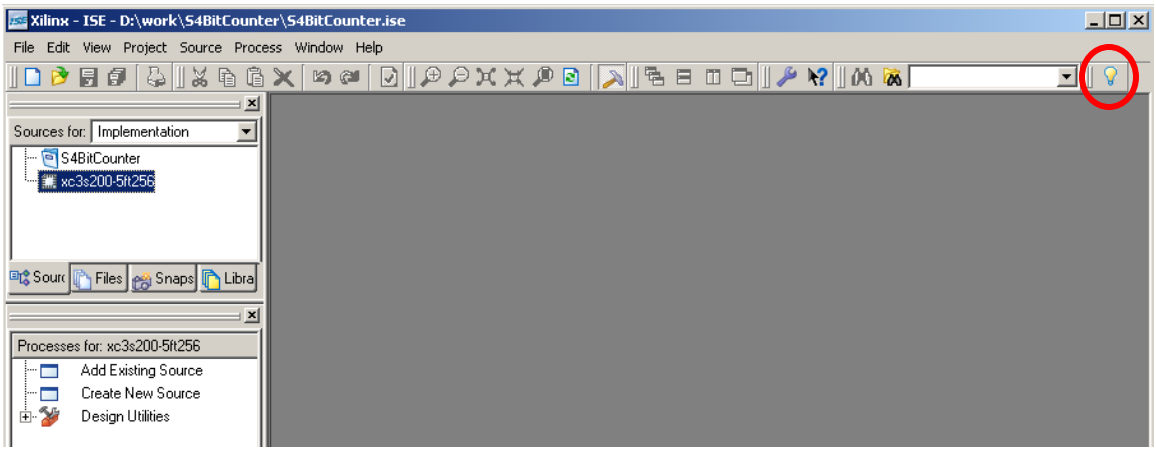
Als Simulator wählen Sie den eingebauten ISE Simulator mit ***"VHDL"*** als ***"Preferred Language"*.**

Im nun folgenden Dialog spezifizieren Sie das Ziel ihres Entwurfsprozesses, d.h. den FPGA-Baustein, der die gewünschte Funktion implementieren soll. Wählen Sie in unserem Falle die folgenden Einstellungen:

	Spartan 2	**Spartan 3**
Product Category	All	All
Family	Spartan2	Spartan3
Device	XC2S100	XC3S200
Package	TQ144	FT256
Speed	-6	-5



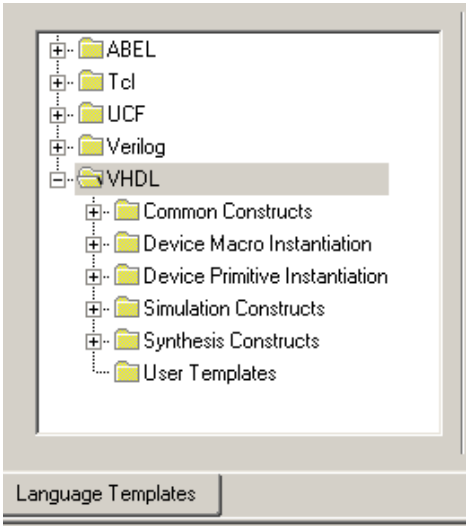
In den nächsten Dialogen können Sie Quelldateien hinzufügen. Wir erledigen dies später. Nach Bestätigen des ***„Finish“***-Buttons erscheint das nachfolgende Fenster. Der Projektname und sämtliche vorher gemachten Einstellungen können durch Doppelklick auf die Objekte im ***„Sources for:“*** Fenster geändert werden.



3.2 Language Templates

Xilinx enthält für verschiedene Design Sprachen (VHDL, Verilog…) vorgefertigte Codeschnipsel (sei es für Sprachkonstrukte oder für bestimmte Entitäten wie zum Beispiel Counter oder Decoder…). Wir verwenden in dieser Dokumentation einen asynchronen 4 Bit Counter. Den ***process*** Block für den Counter können Sie aus dem ***Language Templates*** herausholen. Dazu müssen Sie wie folgt vorgehen:

Klicken Sie das ***„Glühbirnchen“*** (siehe vorherige Abbildung) an um das ***Language Templates*** –Fenster zu öffnen:

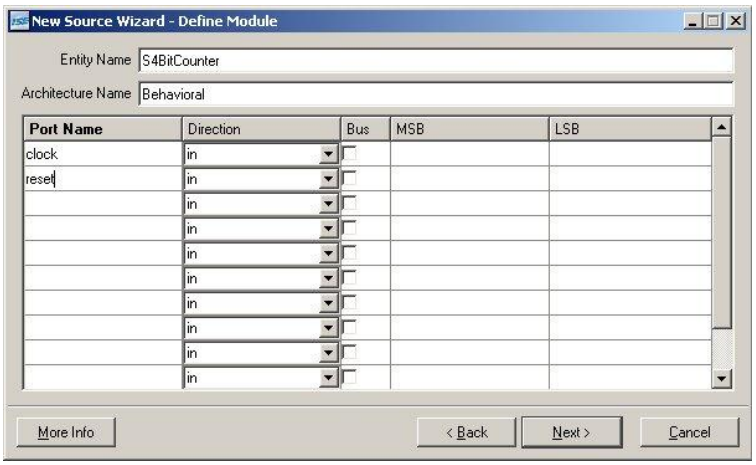


Sie finden den process-Teil zum Counter im Ordner ***VHDL → Synthesis Constructs → Coding Examples → Counters → Binary → Up/Down Counters → /w CE and Async Active High Reset***. Achtung: Sie müssen die Signale (<clock> und <reset> durch in Ihrer Architektur definierte Signalnamen ersetzen.

Sie finden auch Beispiele für bestimmte Sprachkonstrukte. Zum Beispiel das If-Statement in ***VHDL → Synthesis Constructs → Conditional → If / Else If / Else Statement***.

3.3 Neuen Sourcecode im Projekt erstellen

Dazu im Fenster ***Sources for: Implementation*** mit der rechten Maustaste auf ***xc3s200-5tq256 → New Source*** klicken. Im erscheinenden Fenster können Sie den Typ der neuen Datei festlegen und benennen. Wählen Sie ***VHDL Module*** und geben der Datei den Namen ***S4BitCounter***. Anschliessend können Sie den ***Architecture Name*** spezifizieren und die Ports definieren. Wir verwenden einen **clock** (in), **reset** (in) und 4 Datenleitungen. Die Datenleitungen werden wir aber erst später definieren. Somit sehen die Eingaben wie folgt aus:

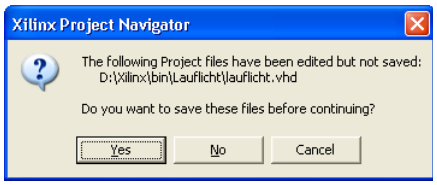


Nachdem sie auf Next und anschliessend au Finish geklickt haben, erscheint ein neues Fenster indem Sie Ihren VHDL-Code eingeben können. Falls das **Design Summary** erscheint, wechseln Sie auf die Quelle, welche unten im Reiter aufgeführt ist.

Die vorhin definierten Ports sind bereits im Code eingetragen. Sie können nun den ***process***-Block des Counters mit „copy-paste“ aus den Templates kopieren. Die **if <clock_enable>** -Anweisung und das dazugehörende **end if** können Sie löschen. Der Sourcecode ist im Kapitel 3.5 **Quellcode** aufgeführt. Achtung: die Signalnamen im Template müssen Sie durch gültige Signalnamen aus Ihrer Portdeklaration ersetzen. Also z.B. statt ‚<reset>’ schreiben Sie ‚reset’.

Das neu erstellte VHDL-File ist nun im Projektfenster sichtbar und befindet sich in der Hierarchie eine Stufe unter dem Hardwaresymbol des Projekts.

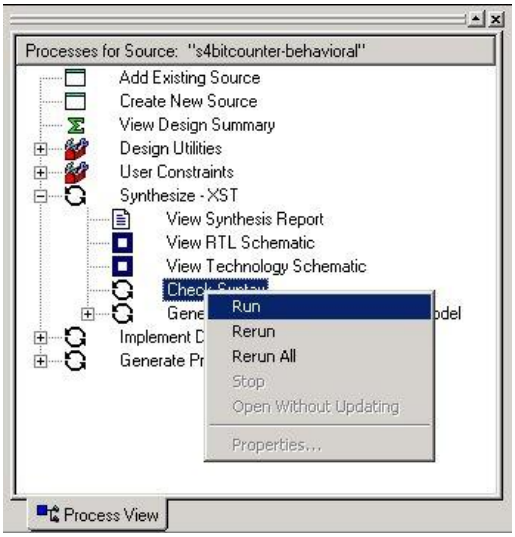
Wichtig ist, dass nach jeder Änderung das VHDL Designs der Sourcecode gespeichert wird. Bei der aktuellen Version des Projekts Navigator werden Sie gefragt ob Sie die Änderung übernehmen wollen, falls Sie davor nicht gespeichert haben.



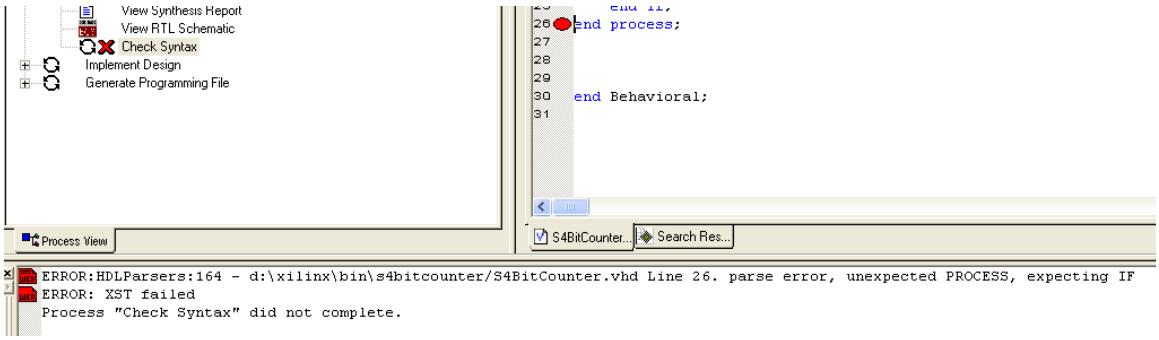
Im Fenster ***Processes*** sehen Sie die Hauptmenüpunkte, welche normalerweise auch die Hauptschritte bei einer Hardwareentwicklung bilden. Wenn Sie nun z.B. bei der ***Synthezise-XST*** mit der Maus auf das **„+“** klicken, öffnen sich unter dem Entwurfsprozess, die Teilschritte wie in diesem Fall u. a. ***Check Syntax***.

3.4 Syntaxcheck

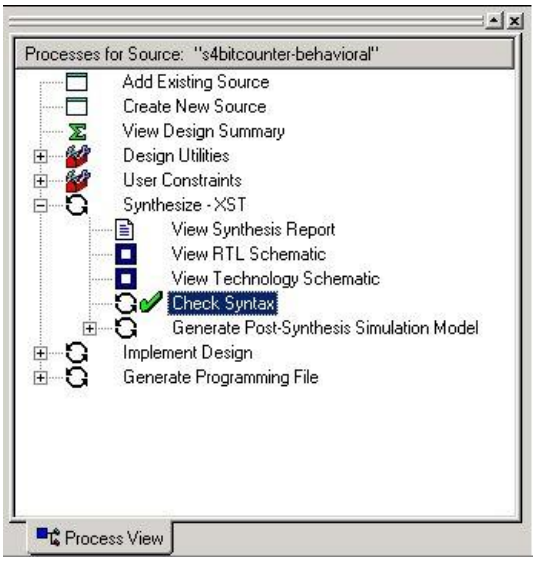
Um den eingegebenen VHDL Code erstmalig zu prüfen wird der Prozess ***Check Syntax*** benutzt. Dazu wird das Kontextmenü des Prozesses mit der rechten Maustaste geöffnet und der Befehl ***Run*** ausgeführt.



In der folgenden Abbildung wird vor dem ***Syntax Check*** ein rotes Kreuz dargestellt, welches auf einen Fehler in der Syntax hinweist. Um den Fehler zu lokalisieren, kann man im unteren Teil des Projekt Navigators, der ***Console***, den Grund des Fehlers ausfindig machen. Dazu muss ggf. der Text ein wenig hochgescrollt werden.



In der ***Console*** sind die Fehler beschrieben. Es ist möglich direkt aus der Konsole, wo der Fehler beschrieben wird, in den VHDL Code zur entsprechenden Fehlerzeile zu wechseln. Dazu auf das rote Quadrat am linken Bildrand neben der jeweiligen Fehlermeldung doppelklicken. Die Fehlerbeschreibung sagt aus: In der Linie 26 wurde ein Semikolon vergessen. Nach Beheben des Fehlers muss das Design neu abgespeichert werden, und der Syntax Check noch einmal durchgeführt werden. An dem grünen Häkchen sehen Sie, dass dieses Mal der Syntax Check erfolgreich war.



3.5 Quellcode

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

entity S4BitCounter is
        Port ( clock : in std_logic;
               reset : in std_logic;
               count_direction : in std_logic;
               count : inout unsigned (3 downto 0));
end S4BitCounter;

architecture Behavioral of S4BitCounter is
begin

    CountProcess: process (clock, reset)
    begin
        if reset='1' then
            count <= (others => '0');
        elsif clock='1' and clock'event then
            if count_direction='1' then
                count <= count + 1;
            else
                count <= count - 1;
            end if;
        end if;
    end process;

end Behavioral;
```

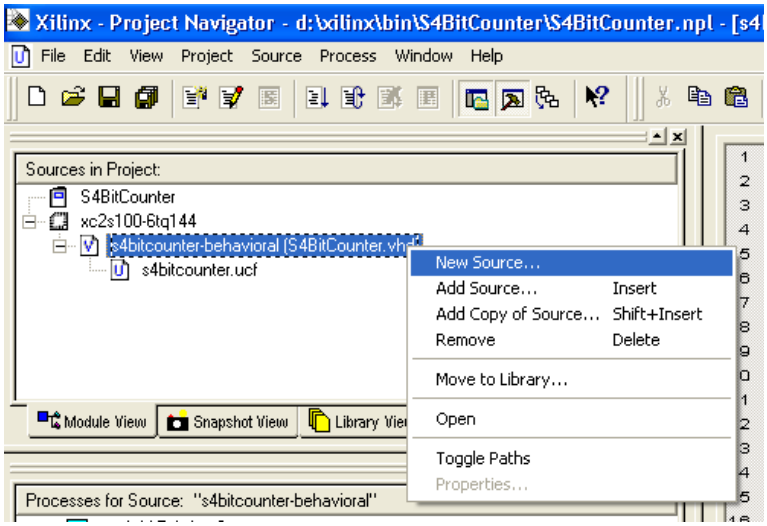
Achtung: Benutzen Sie den Typ „std_logic“ anstelle des Typs „bit“. Grund: Beim Erstellen der Testbench-Datei für die spätere Simulation wird der Typ „bit“ nicht unterstützt. Ebenso sollten Sie den Modus „buffer“ nicht benützen. Verwenden Sie „inout“. Hier liegt der Grund ebenfalls in der späteren Simulation: der Generator für das Post-Fitting Simulationsmodell unterstützt „buffer“ nicht.

4 Simulation mit ISE Simulator

Nach erfolgreicher Syntaxprüfung kann eine Simulation gestartet werden. Am Einfachsten wird dazu eine Testbench erstellt. Alternativ können einzelne Signale auch auf Kommandozeilenebene modifiziert werden.

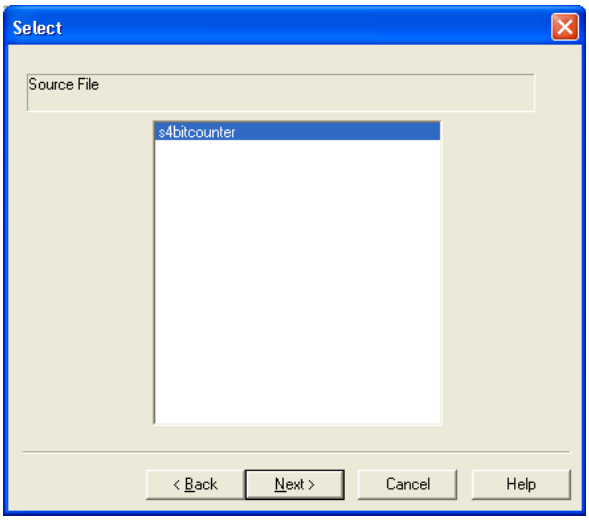
4.1 Testbench erzeugen

Um eine neue Testbench zu erzeugen, können Sie mit der rechten Maustaste auf Ihre ***VHDL-Datei*** klicken und wählen ***New Source…*** .



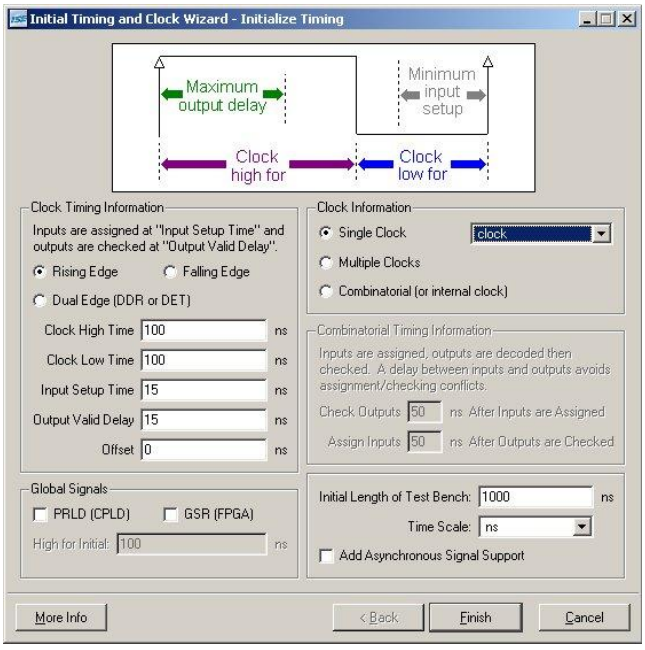
Im neuen Fenster wählen Sie ***Test Bench Waveform*** und geben diesem den Namen **bench1**. Achtung: Die Testbench darf auf keinen Fall den gleichen Namen wie ein VHDL-File in Ihrem Projektverzeichnis haben. Wenn also ein VHDL-File ThreeBitCounter.vhd heisst, könnte die Testbench dazu ThreeBitCounter_tbw.tbw heissen aber nicht ThreeBitCounter.tbw. Der Grund liegt darin, dass für die Testbench ebenfalls eine Entity-Deklaration erstellt wird und diese nicht gleich wie eine zweite im gleichen Projekt heissen darf. Im Fehlerfall werden beim Starten des Simulators die Files aus dem Projekt entfernt!

Im nächsten Fenster sollte Ihre ***VHDL-Datei*** erscheinen welche zur Testbench eingebunden werden soll.

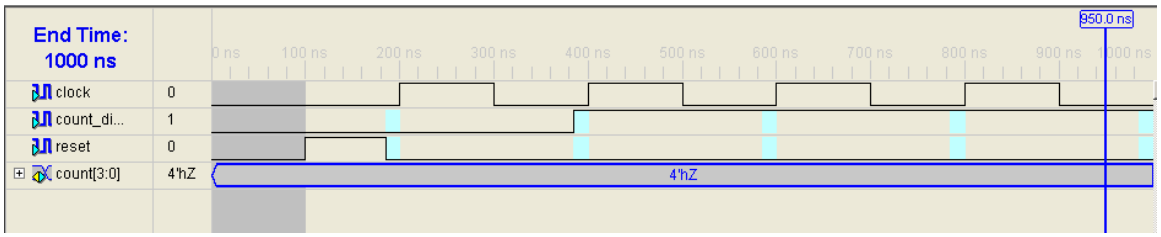


Nun können Sie die Testbench-Datei ***bench1.tbw*** mit Finish erzeugen lassen.

Im nächsten Fenster können Sie die Grundeinstellungen für das Timing vornehmen. Hier können Sie in der Regel alles so belassen wie es eingestellt ist. In **Clock Information** müssen Sie noch das Clock-Signal (hier: ***clock***) auswählen.



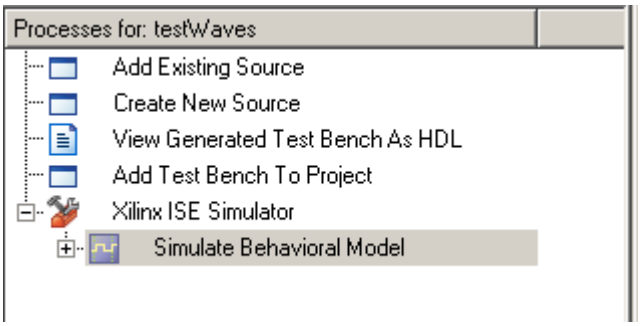
Durch Drücken des ***Finish***-Buttons wird ein grafischer Editor mit den Signalen geöffnet. Mit dem Editor können Sie ganz einfach mit der Maus die zeitlichen Signalpegeländerungen Ihrer Signale definieren. Falls sie die Simulationsendzeit verändern möchten, können Sie auf den rot markierten Bereich doppelklicken und die End-Zeit vergrössern.



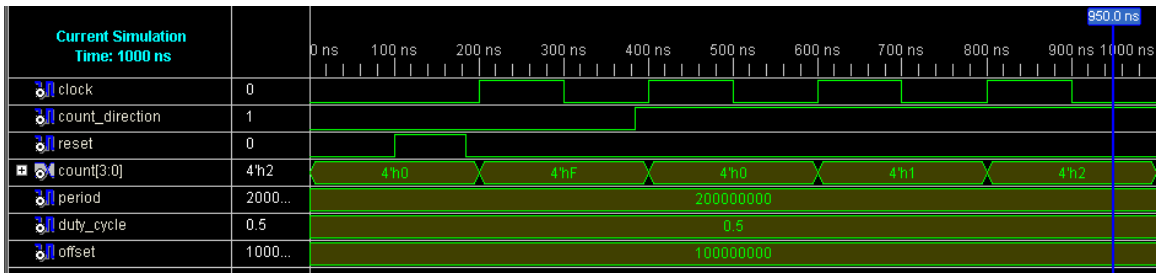
Abschliessend müssen Sie die Einstellungen mit **Ctrl + S** speichern. Wechseln Sie im Fenster ***Sources for:*** die Einstellung auf ***Behavioral Simulation***. Hier erscheint auch die vorhin definierte Test Bench Waveform. Stellen Sie sicher, dass im Fenster darunter die ***Processes*** – Ansicht aktiviert und im oberen Fenster ***bench1*** markiert ist.

Die Simulation wird durch Doppelklicken von ***Simulate Behavioral Model*** gestartet. Das ***Simulate Behavioral Model*** simuliert ohne die zeitlichen Verzögerungen zu berücksichtigen. Die zweite Variante ist ***Simulate Post-Place & Route VHDL Model***, welche diese Verzögerungen berücksichtigt. Für diese Simulation muss in ***Sources for:*** auf ***Post-Route Simulation*** gewechselt werden.

Da wir das Design noch nicht implementiert haben, ist nur ***Simulate Behavioral Model*** an dieser Stelle sinnvoll.

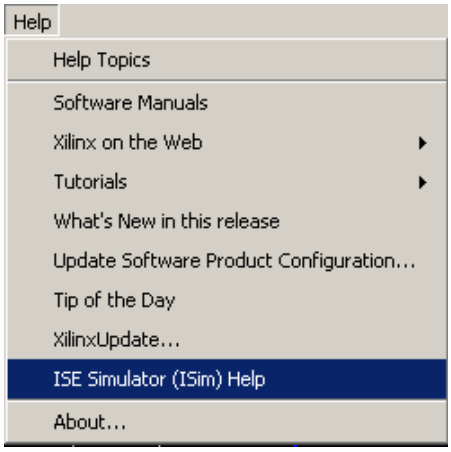


Der Simulator startet und die Ergebnisse können im ***Simulation*** –Fenster betrachtet werden. Um die ganze Simulationsdauer auf einen Blick zu betrachten, können Sie den Menupunkt ***View → Zoom → To Full View*** ausführen.



4.2 Simulator

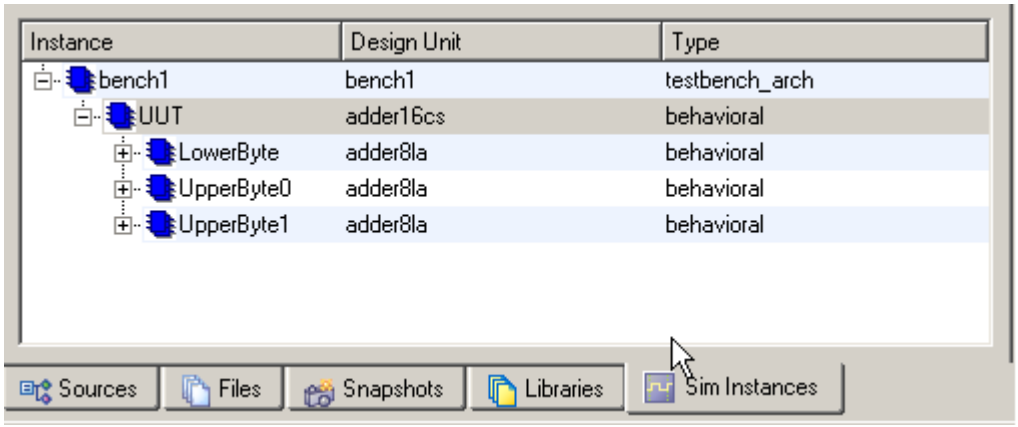
Der in der Xilinx Entwicklungsumgebung integrierte Simulator ist ausführlich in der online-Hilfe beschrieben.



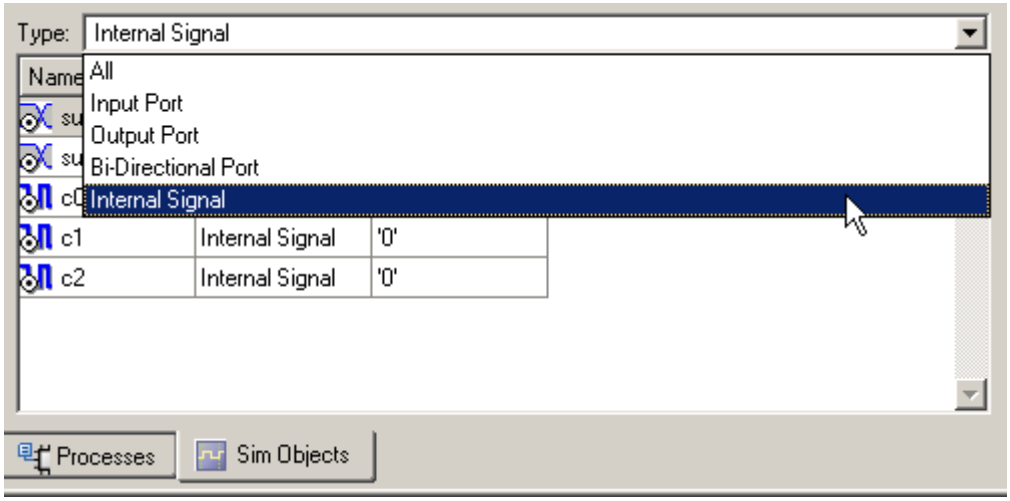
Achten Sie darauf, dass in den Projekteigenschaften auch wirklich der integrierte Simulator ausgewählt wurde (und nicht der Modelsim-Simulator). In der Beschreibung zum Simulator ist beschrieben, wie Sie mit dem Simulator umgehen können, wie Sie mit Do-Dateien simulieren können und wie komplexe Testbenches erstellt werden. Besonders hilfreich sind auch die Angaben darüber, wie Sie das zu simulierende VHDL-File abändern können (neue Signale, andere Signalnamen usw.) und die Änderungen auch in die Simulation übernehmen können.

4.3 Interne Signale darstellen

Per Default werden im Wave-Fenster nur die Signale angezeigt, die sich in der Schnittstelle eines VHDL-Moduls befinden. Oft interessieren aber natürlich auch interne Signale innerhalb einer Architektur. Für die Darstellung solcher interner Signale gehen Sie wie folgt vor.



Klappen Sie im Reiter “Sim Instances” die entsprechende Testbench auf. UUT steht für „Unit Unter Test“. In der Spalte „Design Unit“ ist das zugehörige VHDL-Modul aufgelistet. Selektieren Sie das Modul, in dem das gewünschte interne Signal definiert wurde. Anschliessend zeigen Sie sich im Reiter „Sim Objects“ die internen Signale an.



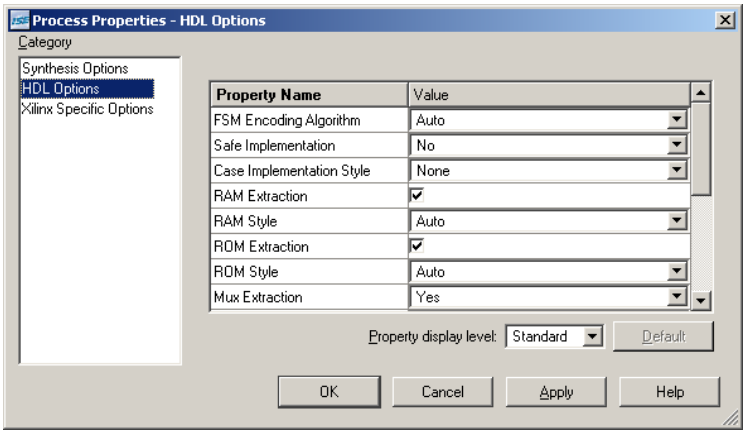
Mit der rechten auf dem gewünschten Signalnamen können Sie das Signal ins Wave-Fenster hinzufügen. Zum Schluss müssen Sie eine erneute Simulation durchführen, damit der Verlauf des neuen Signals auch wirklich dargestellt wird.

5 Synthetisieren

Die Synthese beinhaltet zunächst ein Syntax Check aller VHDL Dateien eines Projektes.

5.1 Syntheseeigenschaften

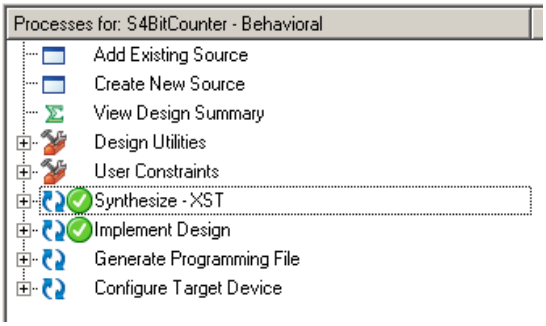
Um die Synthese-Eigenschaften zu ändern, muss ***Sources for:*** auf ***Implementation*** stehen. Drücken Sie im ***Processes*** Fenster die rechte Maustaste auf ***Synthesise XST*** und wählen Sie den Menüpunkt ***Properties***.



Im Dialog ***Process Properties*** haben Sie die Möglichkeit Optionen für die Synthese anzupassen. z.B. bei ***HDL Options*** kann angegeben werden wie die Zustände der FSM´s codiert werden sollen, oder welche FlipFlop Typen in der Synthese verwendet werden sollen. Für das Beispiel 4 Bit Counter, dass in diesem Dokument behandelt wird, sind keine Änderungen notwendig.

5.2 Synthese durchführen

Um die Synthese durchzuführen, kann über das Kontextmenü im Prozessfenster gegangen werden. Drücken Sie dazu die rechte Maustaste auf dem Item ***Synthese*** und wählen Sie den Menüpunkt ***Run***.



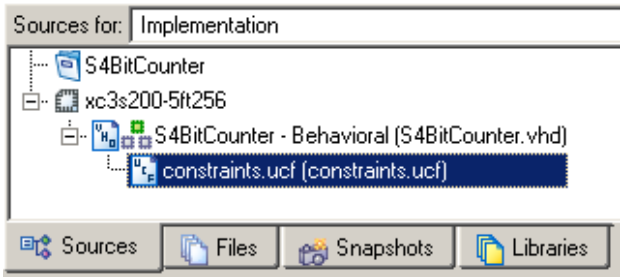
6 Constraints-Datei erzeugen

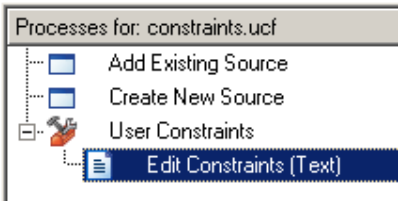
Als weiteres Tool ist der Constraints Editor (PACE) im Webpack vorhanden, mit dem vor der Implementierung die Ein- und Ausgangsports speziellen Pins am Baustein zugewiesen und spezielle Vorgaben für das Zeitverhalten definiert werden können. Diese Einstellungen werden in einer ***Constraints-Datei*** gespeichert.

6.1 Textuelle Eingabe

Klicken Sie im Fenster ***Sources for: Implementation*** mit der rechten Maustaste auf ***xc3s200-5tq256 → New Source*** klicken. Im erscheinenden Fenster wählen Sie ***Implementation Constraints File*** und geben der Datei den Namen ***constraints***. Das erzeugte File wird ***constraints.ucf*** heissen.

Um die ***Constraints-Datei*** zu editieren, wählen Sie die Datei aus und drücken auf ***Edit Constraints (Text)*** im ***Sources***-Fenster.





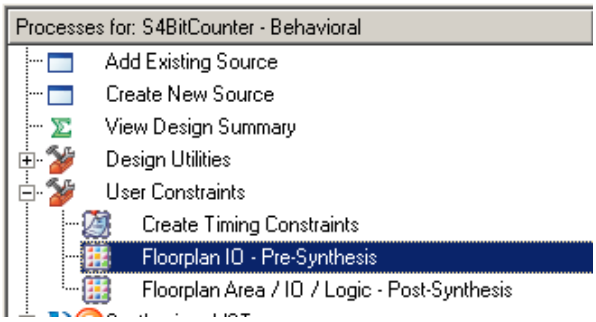

```
 2  #PINLOCK_BEGIN
 3
 4  #Mon Jul 18 13:38:48 2005
 5
 6  NET "count<0>"        LOC =  "P138";
 7  NET "count<1>"        LOC =  "P5";
 8  NET "count<2>"        LOC =  "P13";
 9  NET "count<3>"        LOC =  "P26";
10  NET "clock"           LOC =  "P15";
11  NET "reset"           LOC =  "P42";
12  NET "count_direction"  LOC =  "P54";
13  #PINLOCK END
```

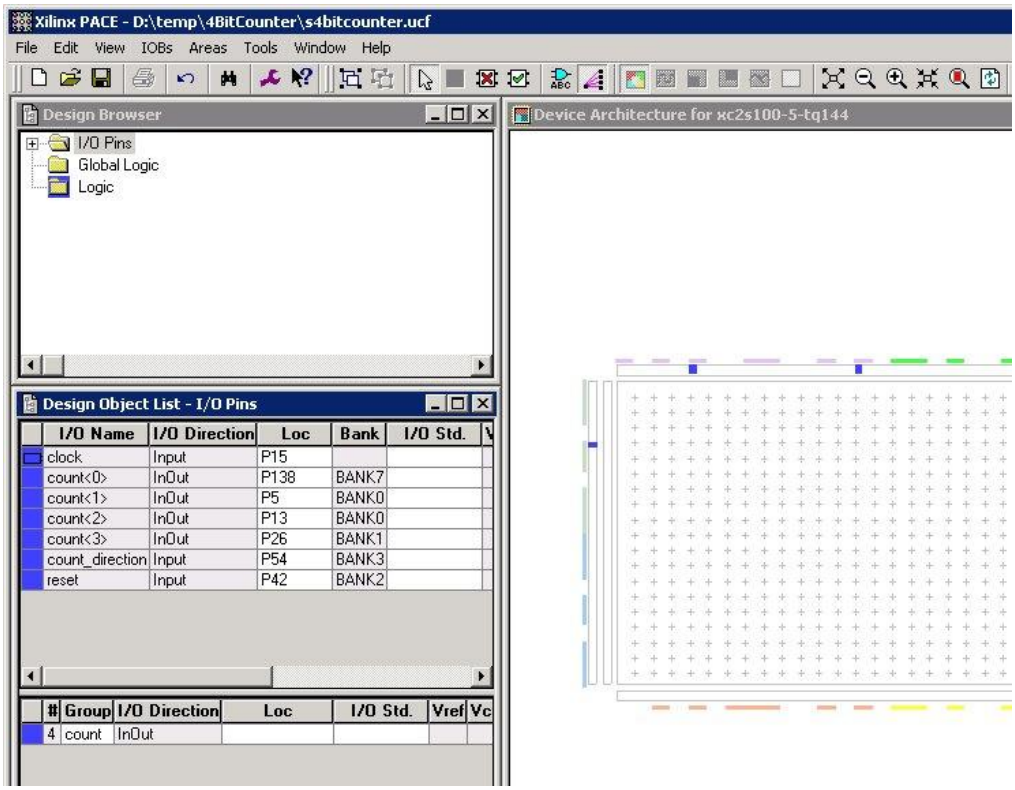
NET "clock" LOC = "P15" → P15 bedeutet, dass der Pin I/O(15) auf dem Testboard verwendet wird.

Dieses Vorgehen loht sich insbesondere dann, wenn Sie bereits eine Constraints-Datei haben und nur kleine Änderungen vornehmen müssen.

6.2 Constraints Editor (PACE)

Der grafische Editor für die Zuweisung der einzelnen Pins kann über das ***Process***-Fenster mit einem Doppelklick auf ***Floorplan IO – Pre-Synthesis*** gestartet werden.





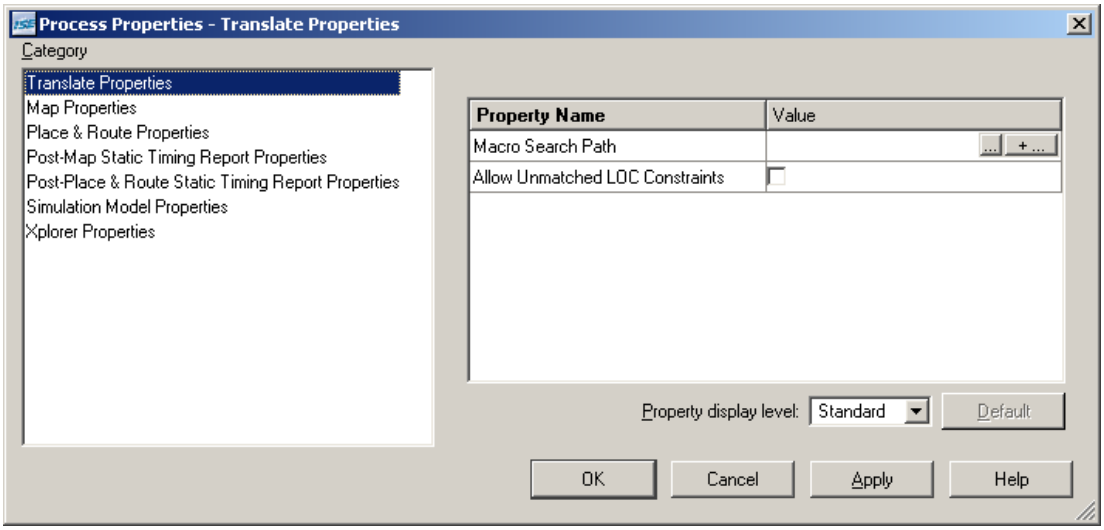
Im Fenster ***Design Object List – I/O Pins*** in der Spalte ***Location*** können Sie nun die Hardware-Pins für Ihr Design eintragen. Auf den Experimentierboards sind die Pins mit ***I/O(x)*** beschriftet, was hier im PACE mit ***Px*** angegeben wird. Nach der Zuweisung der einzelnen Signale auf die Pins können Sie Ihre ****.ucf***-Datei speichern. Es genügt dazu auf das Disketten-Symbol zu drücken. Sie können nun den PACE schliessen.

7 Design Implementierung

Nachdem die Synthese abgeschlossen ist, kann mit der Funktion ***Implement Design*** fortgefahren werden. In dieser Funktion des ***Projekt Navigators*** wird die während der Synthese erzeugte symbolische Gatternetzliste auf die zur Verfügung stehenden Ressourcen (Produktterme, Eingangs- und Ausgangszellen) abgebildet. Darüber hinaus wird eine Datei erzeugt, mit der eine Timing Simulation durchgeführt werden kann, welche alle Verzögerungen der Hardware mit berücksichtigt.

7.1 Implementierungseigenschaften

Um die Eigenschaften für die Implementierung zu ändern, kann im Prozessfenster mit der rechten Maustaste auf ***Implement Design*** über das Kontextmenü der Befehl ***Properties*** ausgewählt werden.



Im Allgemeinen sind Änderungen gegenüber den Voreinstellungen in unserem Falle nicht nötig. Es können aber Zusätze eingetragen werden. Um kontextbedingte Hilfe für die einzelnen Einstellungen zu erhalten, können Sie den ***Help***-Button drücken. Danach erscheinen im Browser die Erklärungen zu den ***Name / Value*** – Paaren im entsprechenden Tab.

7.2 Implementierung durchführen

Um die Implementierung durchzuführen, kann über das Kontextmenü im Prozessfenster gegangen werden. Drücken Sie dazu die rechte Maustaste auf dem Item ***Implement Design*** und wählen Sie den Menüpunkt ***Run***.

8 Post-Fitting Simulation

Post-Fitting Simulation bezeichnet eine Simulation, welche zusätzlich zur funktionalen Simulation auch die zeitlichen Verzögerungen des realen Bausteins berücksichtigt. Die folgende Abbildung zeigt den Vorgang zur Durchführung einer Post-Fitting Simulation.

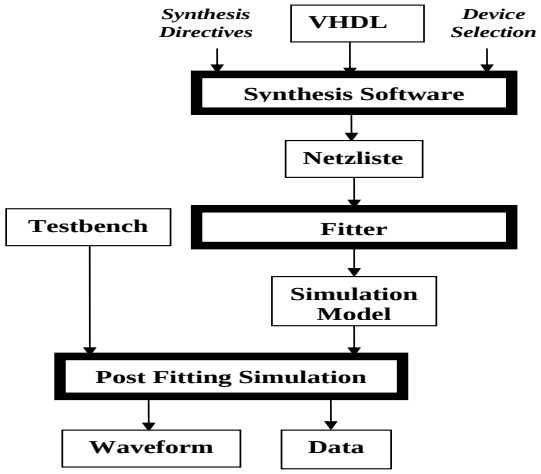
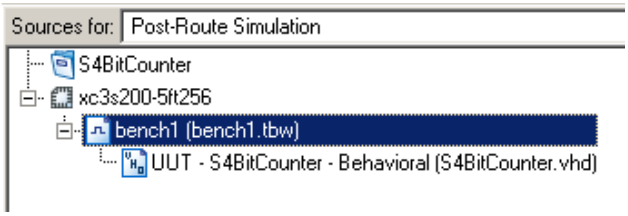
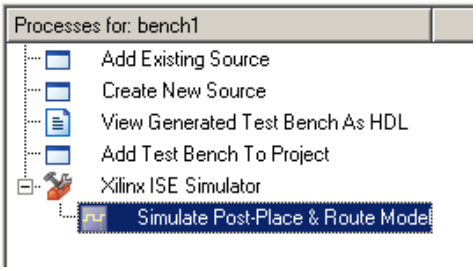


Abbildung 1: Post-Fitting Simulation

Für eine Post-Fitting Simulation benötigen Sie ebenfalls eine ***Testbench***-Datei. Diese können Sie wie im Unterkapitel 4.1 beschrieben erstellen. Nach Erstellen der ***Testbench***-Datei, ***Sources for: Post-Route Simulation*** auswählen und dann ***Simulate Post-Place& Route VHDL Model*** ausführen. Achtung: Sie können die ***Post-Route Simulation*** nur auswählen, wenn Sie die Implementation bereits durchgeführt haben und dazu müssen Sie ein gültiges Device unter den Projekteigenschaften ausgewählt haben.





9 Programm auf Hardware laufen lassen

Nach der ***Synthese*** und ***Design Implementation*** kann nun das Programm auf die Hardware herunter geladen werden.

Dazu muss noch der letzte Schritt im ***Process***-Fenster (***Sources for: Implementation***) durchgeführt werden. Einfach dazu mit der rechten Maustaste auf ***Generate Programming File*** und ***Run*** ausführen.

9.1 FPGA- Board

Die folgende Beschreibung ist stets doppelt gehalten. Wir haben aktuell an der NTB für den Unterricht den Spartan2 auf einer Eigenbauplattform sowie den Spartan3 auf einem kommerziellen EVM im Einsatz. Bitte behandeln Sie die Hardware mit der notwendigen Vorsicht: Generell ist es wichtig, dass Sie vor jedem Umschalten der Dip-Schalter, Jumper oder Kabels die Speisung ausschalten!

Spartan 2:

Das Board weist zwei 50-polige Stiftleisten auf. Alle verfügbaren I/O's sind zugänglich. Eine Schaltung für die Konfiguration (***PROM***) ist bereits vorhanden.

Das Board wird extern mit 5V gespiesen. Intern läuft das Board mit 3.3V für alle I/O's und 2.4V im Kern. Aus diesem Grund ist es äusserst wichtig, dass die I/O's des Spartan immer so konfiguriert werden, dass sie 5V tolerant sind (default). Man muss sich auch bewusst sein, dass ein Ausgang nie mehr als 3.3V liefern kann.

Eine Interface-Schaltung ***LPT-2-JTAG*** ist auf dem Experimentierprint vorhanden (Unterseite). Wahlweise kann über ***JTAG*** oder ***Slave-Serial*** programmiert werden. Wir benützen ***JTAG*** und stecken das lose kleine Kabel in den Sockel ***"JTAG"***.

Achtung: Wenn das ***PROM*** nicht bestückt ist, muss der Jumper ***"Bypass"*** gesetzt sein (wir haben ein ***PROM***). Ohne diesen Jumper und ohne PROM ist die JTAG-Chain nicht geschlossen.

Der zweite Jumper muss auf ***"PROM"*** stehen, ansonsten wird die Konfiguration für's ***FPGA*** nach einem Power-up über ***Slave-Serial*** geholt und nicht aus dem ***PROM***.

Auf dem Board befinden sich drei Dip-Schalter(M0, M1, M2). Mit diesen legen Sie fest, wie das FPGA konfiguriert wird.

	M0	M1	M2
FPGA	OFF	OFF	OFF
JTAG	OFF	ON	OFF
PROM	ON	ON	ON

Die Einstellung *FPGA* bedeutet, dass die Konfiguration zu Beginn über ***Slave-Serial*** geholt wird. Diese Einstellung benötigen wir nicht. Mit *JTAG* können Sie vom Host aus entweder das FPGA oder das PROM oder beides programmieren. Schlussendlich be-

nutzen Sie die Einstellung *PROM*, damit das FPGA im stand-alone Betrieb seine Konfiguration aus dem PROM holt.

Spartan 3:

Diese Beschreibung gilt für das Spartan-3 EVM von Digilent.

Damit die Konfiguration des FPGA's nach einem Power Up über das Flash geholt wird, muss Jumper 1 auf ***Flash*** oder ***Flash Read*** stehen. Bei ***Flash Read*** besteht im Gegensatz zu ***Flash*** nach dem Bootvorgang die Möglichkeit, zusätzliche Daten aus dem ***PROM*** auszulesen.

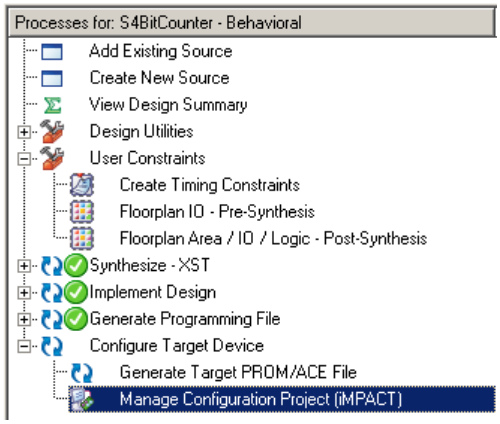
Über die Jumper M0,M1,M2 wird festgelegt wie das FPGA konfiguriert wird.

	M0	M1	M2
PROM	G	G	G
Slave Serial	O	O	O
Master Parallel	G	G	O
Slave Parallel	G	O	O
JTAG	O	G	O

Bei den Einstellungen *Slave Serial*, *Master Parallel* und *Slave Parallel* wird die Konfiguration des FPGA's über die entsprechende Schnittstelle geholt. Mit JTAG können Sie vom Host aus entweder das FPGA, das PROM oder beides programmieren. Schlussendlich benutzen Sie die Einstellung PROM, damit das FPGA aus dem PROM konfiguriert wird. Das heisst, dass die Stellung JTAG gewählt werden muss, für das Programmieren des FPGA's und die Stellung PROM, wenn sich das FPGA die Konfiguration aus dem PROM holen soll.

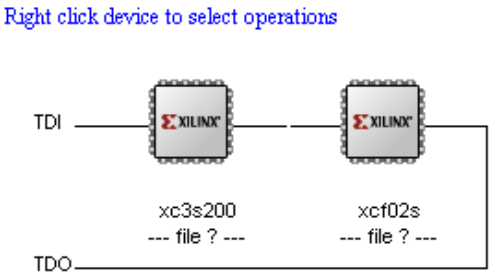
9.2 Vorgehen bei FPGA- Konfiguration

1. Dips auf Stellung ***JTAG*** und Speisung einschalten.
2. ***iMPACT*** starten, indem Sie im ***Processes*** Fenster unter ***Configure Target Device*** den Punkt ***Manage Configuration Project (iMPACT)*** doppelklicken.

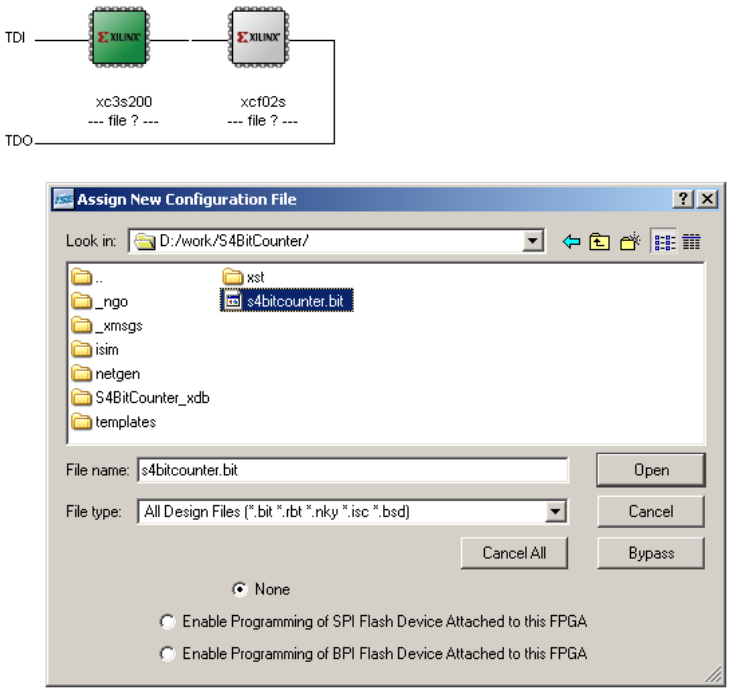


3. Wählen sie in ***Configure devices using Boundary-Scan (JTAG)*** den Punkt ***Automatically connect to a cable and identify Boundary-Scan chain***.
4. **Spartan 2:** In der Chain sollten jetzt 2 Devices angezeigt werden, das ***PROM "xc18v01"*** (falls vorhanden) und das ***FPGA "xc2s100"***:

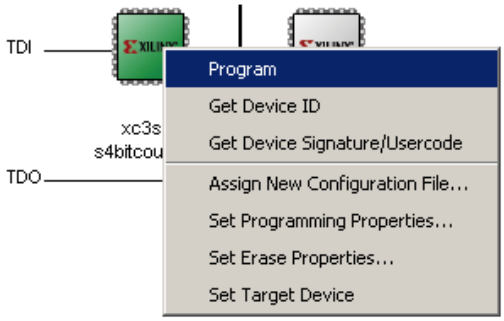
   Spartan 3: Hier sollten jetzt das ***PROM "xcf02s"*** sowie das ***FPGA*** "***sc3s200***" in der Chain angezeigt werden.

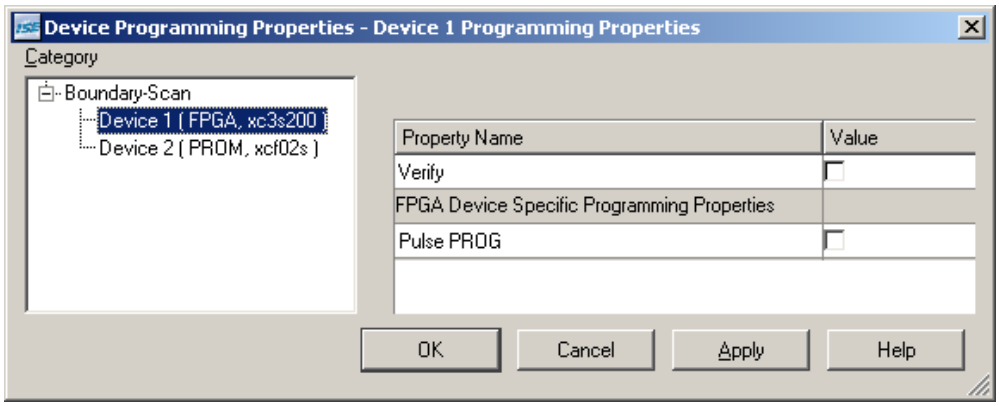


5. Konfigurationsfile zuweisen
   - Für ***PROM*** noch nicht vorhanden, also ***"Cancel".***
   - Für ***FPGA*** richtige ***Bit***-Datei wählen.

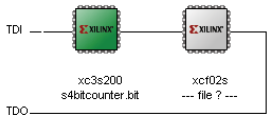


6. Rechte Maustaste auf ***FPGA*** und ***"Program"*** ausführen.
Achtung Haken bei ***"Verify"*** unbedingt entfernen und danach ***OK*** drücken:



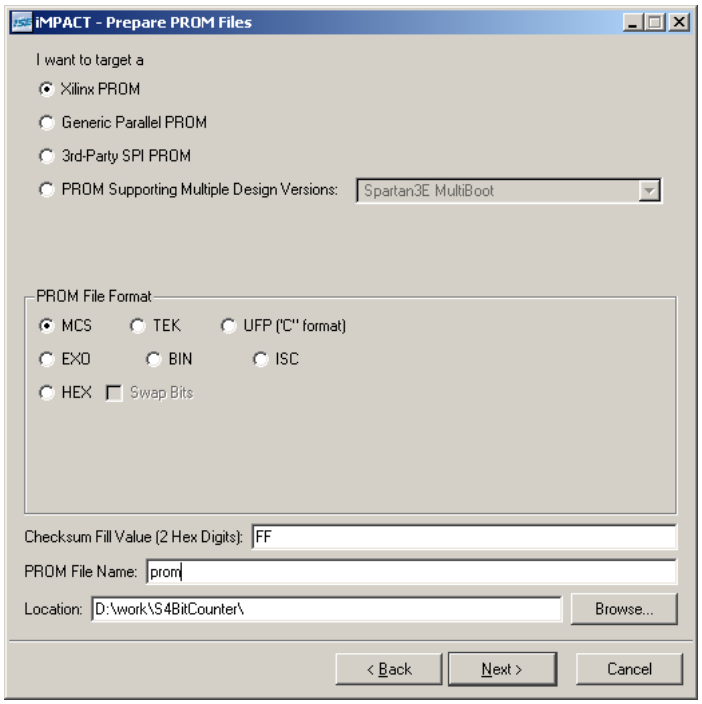


7. ***FPGA*** ist nun konfiguriert und läuft gleich anschliessend mit dieser Konfiguration.



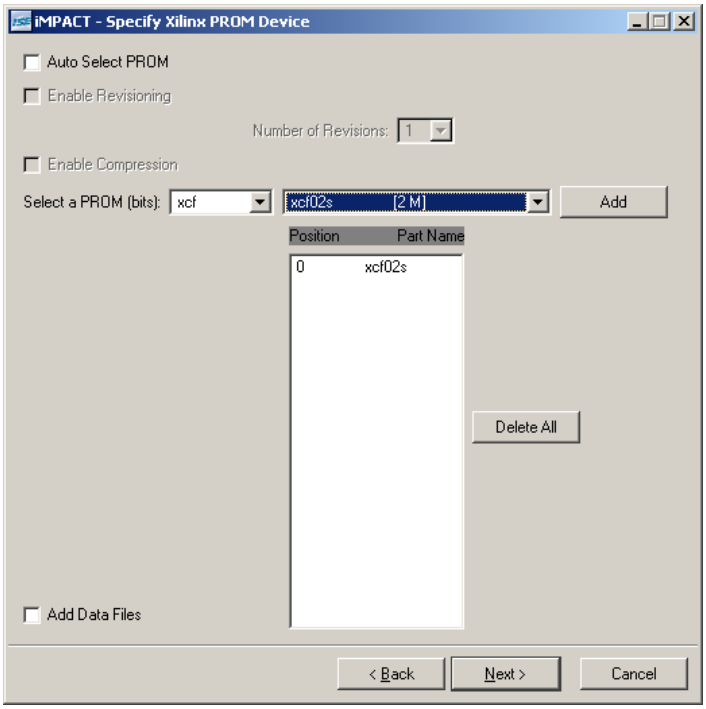
9.3 Vorgehen bei PROM- Konfiguration

1. Dips auf Stellung ***JTAG*** und Speisung einschalten.
2. Den Punkt ***Generate Traget PROM/ACE File*** im ***Processes*** Fenster unter ***Configure Target Device*** anwählen, ***Prepare a PROM File*** selektieren.
3. Wir haben ein ***XILINX Prom*** und File Format ***"MCS"***. Angabe des Speicherortes und Dateiname nicht vergessen:

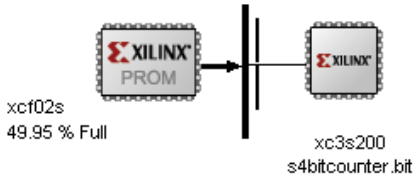


Im folgenden Dialog ***I am using a Xilinx PROM in Serial Mode*** wählen.

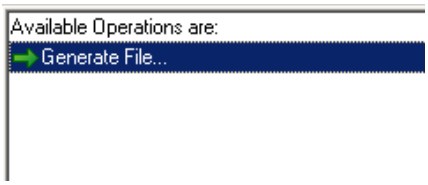
4. **Spartan 2:** ***PROM xc18v01*** wählen und ***"Add"*** drücken und mit ***Next*** weitergehen.
   Spartan 3: ***PROM xcf02s*** wählen**,** "***Add*****"** drücken und mit ***Next*** weitergehen**.**



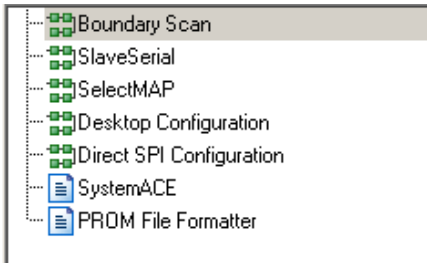
5. Mit ***"Add File"*** das richtige ***Bit***-File zuweisen und mit ***"Finish"*** abschliessen.
6. Jetzt wird das PROM in die JTAG-Kette eingefügt und dem richtigen FPGA zugewiesen. Dabei wählen Sie auch das korrekte Bit-File aus.
7. Das Addieren weiterer Bausteine lehnen Sie ab.



8. Über das Menu ***Configuration Operations*** kann der Punkt ***Generate File*** gewählt werden



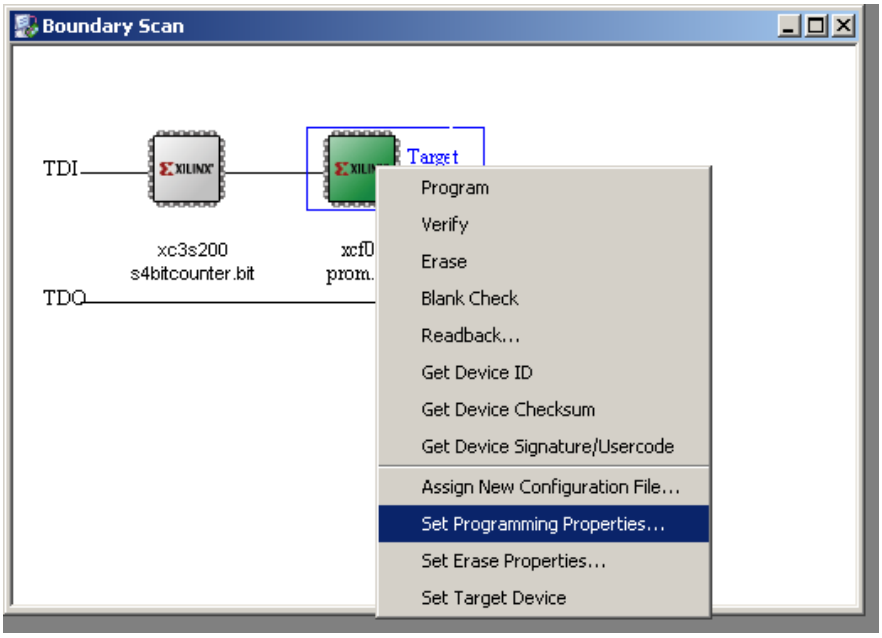
9. Auf Punkt ***Boundary-Scan***. Rechte Maustaste ***"Initialize Chain"***. In der Chain sollten jetzt 2 Devices angezeigt werden, das ***PROM "xcf02s"*** und das ***FPGA "xc3s200"***.



10. Konfigurationsfile zuweisen
    - Für ***PROM*** das vorher erzeugte ***PROM***-File wählen
    - Für ***FPGA*** richtiges ***Bit***-File wählen. Diese Wahl können Sie mit „Cancel" auch weglassen.

Die auftretenden Warnungen können Sie ignorieren. Wenn sie nerven: Setzen Sie unter *Properties* von *Generate Programming File* den Startup-Clock auf JTAG-Clock.

11. Programming Properties setzen: Haken bei ***Verify*** und ***Erase Before Programming*** setzen.



12. Rechte Maustaste auf ***PROM*** und ***Program*** ausführen
13. Das ***PROM*** ist nun gebrannt.
14. Speisung aus und Dips auf Stellung ***PROM*** setzen.
15. Nach Einschalten der Speisung holt das ***FPGA*** die Konfiguration aus dem ***PROM***.