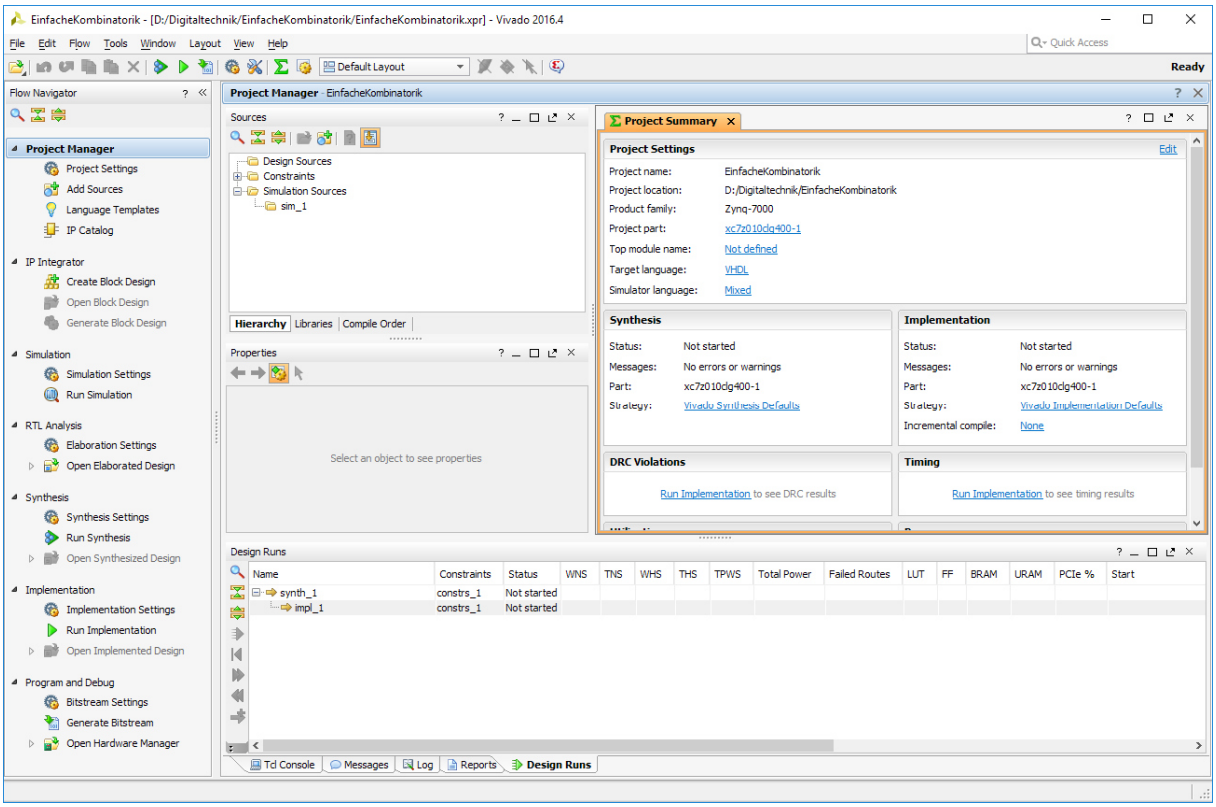
Simulation und Synthese mit Vivado

Nachdem Sie ein Vivado-Projekt (nach der ebenfalls hier zur Verfügung gestellten Anleitung) neu angelegt haben, enthält es noch keine Quelldateien.



Bevor wir Dateien zu dem Projekt hinzufügen, zunächst ein Blick auf die linke Seite des Vivado-Fensters: Dort finden Sie den Flow-Navigator, in dem die einzelnen Schritte des Designflows übersichtlich dargestellt sind. Dies ist sozusagen Ihr Cockpit, aus dem heraus Sie alle wichtigen Schritte von der Designeingabe über Simulation und Synthese bis hin zum Laden Ihres Designs auf ein FPGA-Board durchführen können.

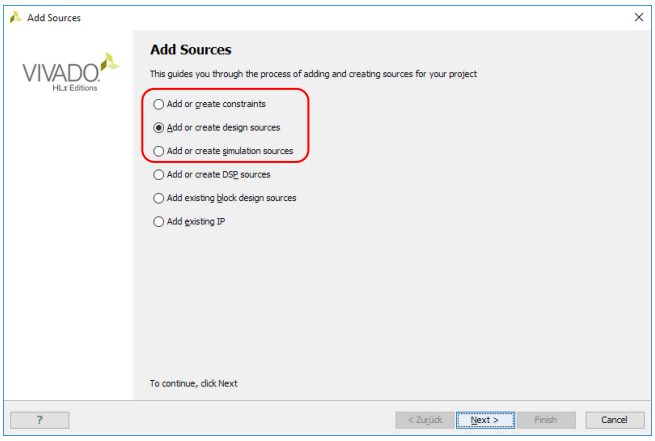
Hier ein kurzer Überblick über die einzelnen Einträge im Flow Navigator:

- **Project Manager**
  Im Bereich Project Manager können Sie Projekteinstellungen wie zum Beispiel die verwendete Sprache (VHDL oder Verilog) oder den verwendeten FPGA-Baustein auswählen. Darüber hinaus können mit *Add Sources* bestehende Dateien zu dem Projekt hinzugefügt werden oder neue Dateien angelegt werden. Unter Language Templates finden Sie zahlreiche Vorlagen, die Sie in eigene VHDL-Beschreibungen übernehmen können.
- **IP Integrator**
  Dieser Bereich bietet Ihnen die Möglichkeit, ein Design mit Hilfe einer grafischen Eingabe anzulegen. Hierbei werden bestehende Blöcke (Intellectual Property, IP) verwendet. Die Möglichkeit zur grafischen Designeingabe nutzen wir in diesem Dokument nicht.
- **Simulation**
  Hier finden Sie Einstellungen zur Simulation eines VHDL-Entwurfs. Mit *Run Simulation* kann der in Vivado integrierte Simulator gestartet werden.

- **RTL Analysis, Synthesis, Implementation, Program and Debug**
  Diese Bereiche umfassen die einzelnen Schritte von der VHDL-Beschreibung bis zur Erzeugung einer binären Datei, die auf das FPGA geladen werden kann. Sie müssen keine Einstellungen verändern, da die von Vivado vorgewählten Einstellungen für viele Designs völlig ausreichend sind.
  Wenn Sie möchten, können Sie die Entwurfsschritte einzeln durchlaufen. Einfacher ist es aber, wenn Sie einfach auf *Generate Bitstream* klicken. Vivado wird dann die erforderlichen Schritte (z.B. Synthese, Place und Route) eigenständig durchführen und anschließend die Binärdatei für das FPGA (bei Vivado wird diese Datei als „Bitstream“ bezeichnet) erzeugen.

Hinzufügen von Quelldateien

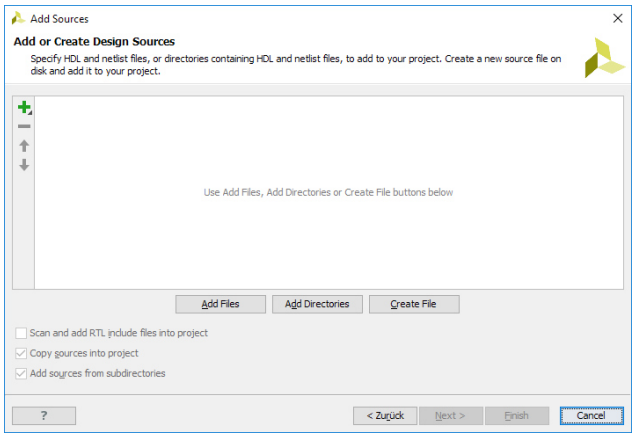
Wählen Sie im Flow Navigator den Project Manager aus und klicken Sie auf Add Sources.



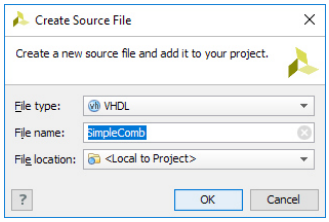
Sie werden nun gefragt, welche Art von Quelldateien Sie hinzufügen möchten:

- **Design Sources**
  Hiermit werden die VHDL-Dateien bezeichnet, die synthetisiert und auf das FPGA transferiert werden sollen.
- **Simulation Sources**
  Diese Dateien werden nur in der Simulation verwendet und nicht synthetisiert. Es handelt sich in der Regel um die Testbench.
- **Constraints**
  Mit Contraint-Dateien werden sozusagen die Randbedingungen für die Generierung der binären FPGA-Datei festgelegt. Constraints enthalten zum Beispiel die Information an welchem Anschluss des FPGAs ein in VHDL beschriebener Eingang oder Ausgang liegen soll und welchen Eigenschaften der Anschluss hat (z.B. welche Logikpegel verwendet werden sollen). Wird ein Taktsignal verwendet, wird mit Hilfe der Constraints angegeben, welche Eigenschaften (z.B. Frequenz) das Taktsignal besitzt.

Wählen Sie ***`Design Sources`*** aus und klicken Sie auf *`Next`*.



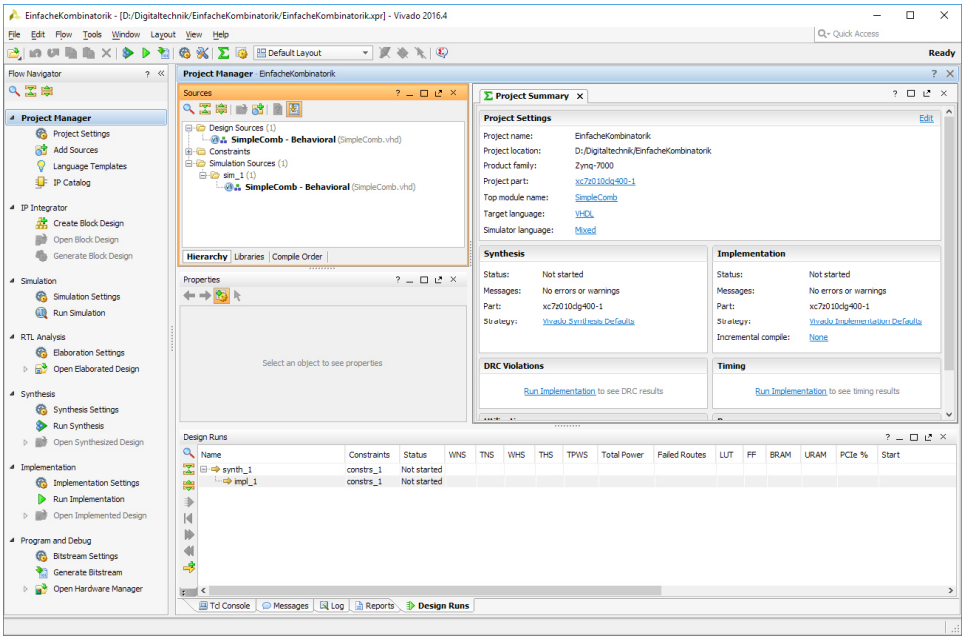
Möchten Sie existierende Dateien, die bereits auf Ihrem Rechner liegen, hinzufügen oder neue Dateien erzeugen? Wir entscheiden uns für die Erzeugung einer neuen Datei und klicken auf Create File. Als Dateinamen wählen wir SimpleComb. Die Dateiendung .vhd fügt Vivado übrigens eigenständig an.



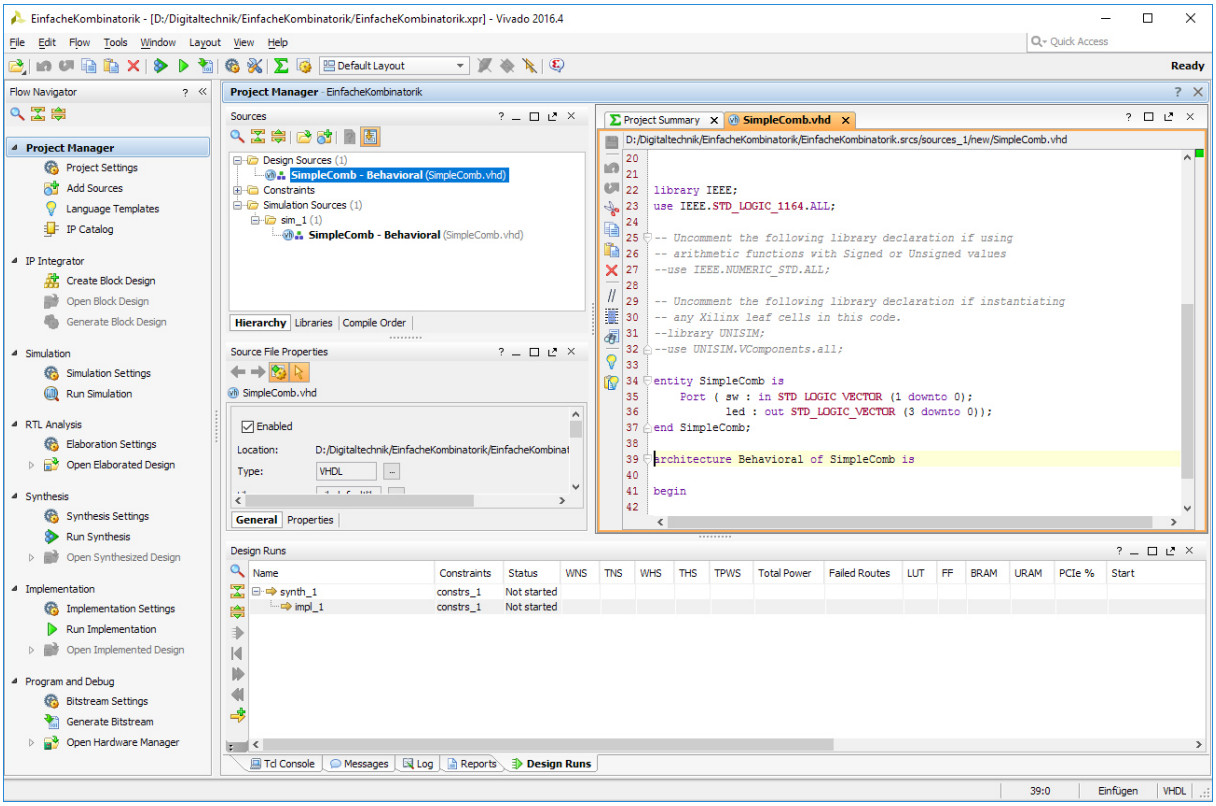
Da wir nur eine Design-Datei hinzufügen wollen, kann nun Finish ausgewählt werden.

Vivado bietet nun die Möglichkeit, die Ports des VHDL-Moduls SimpleComb anzulegen. Hiervon machen wir Gebrauch und legen den Eingang *sw* mit einer Wortbreite von 2 Bit und den Ausgang *Led* mit 4 bit an. Wichtig: VHDL unterscheidet nicht Groß- und Kleinschreibung - die Contraints-Dateien dagegen schon. Bitte verwenden Sie daher die hier angegebene Kleinschreibung der Portnamen.

Die VHDL-Datei *SimpleComb.vhd* wird angelegt und erscheint im Projektmanager sowohl unter Design Sources als auch unter Simulation Sources. (Dies ist ja auch logisch, denn eine Datei, die synthetisiert werden soll, sollte auf jeden Fall auch in der Simulation getestet werden)



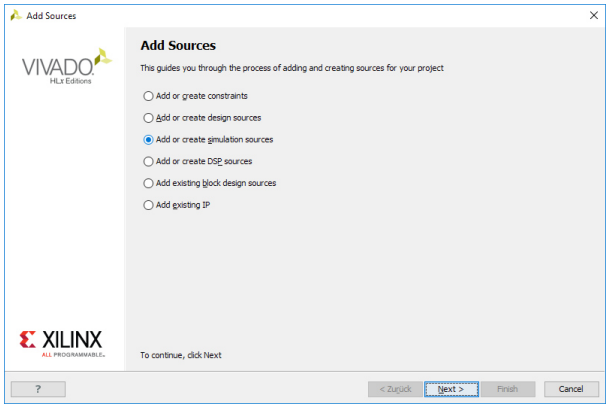
Ein Doppelklick auf die im Project Manager öffnet den Editor. Bis auf die Architecture ist in der Datei bereits alles vorbereitet. Wenn Sie wollen, können Sie die Kommentare in der Quelldatei aufräumen und so die Datei kompakter machen.



In der Architecture fügen wir einen einfachen kombinatorischen Prozess hinzu:

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3
4  entity SimpleComb is
5      port ( sw  : in  STD_LOGIC_VECTOR (1 downto 0);
6             led : out STD_LOGIC_VECTOR (3 downto 0));
7  end;
8
9  architecture Behavioral of SimpleComb is
10
11 begin
12
13     process (sw) -- kombinatorischer Prozess, der auf die Eingänge sw reagiert
14     begin
15         led (0) <= sw(0);
16         led (1) <= sw(1);
17         led (2) <= sw(0) or  sw(1);
18         led (3) <= sw(0) and sw(1);
19     end process;
20
21 end;
```

Auf die gleiche Weise legen wir nun eine Testbench-Datei mit dem Namen *SimpleCombTB* an. Achten Sie hierbei darauf, dass Sie „*Add or create simulation sources*“ auswählen:



Da eine Testbench keine Ports besitzt, beenden Sie den „*Define Module*“ Dialog einfach mit *Ok* und geben keine Ports an.

Die editierte Testbench-Datei könnte wie folgt aussehen:

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3
4  entity SimpleCompTB is
5  end SimpleCompTB;
6
7  architecture Behavioral of SimpleCompTB is
8
9  signal sw_sig  : std_logic_vector (1 downto 0);
10 signal led_sig : std_logic_vector (3 downto 0);
11
12 begin
13
14     -- Instanz des zu testenden Moduls
15     my_comb: entity work.SimpleComb
16     port map (
17         sw  => sw_sig,
18         led => led_sig
19     );
20
21     -- Ein Prozess, der die Eingänge "stimuliert"
22     process
23     begin
24         sw_sig <= "00";         -- Wert an sw zuweisen
25         wait for 100 ns;        -- einige Zeit warten
26         sw_sig <= "01";         -- Nächster Wert...
27         wait for 100 ns;        -- usw.
28         sw_sig <= "10";
29         wait for 100 ns;
30         sw_sig <= "11";
31         wait for 100 ns;
32     end process;
33
34 end Behavioral;
35
```

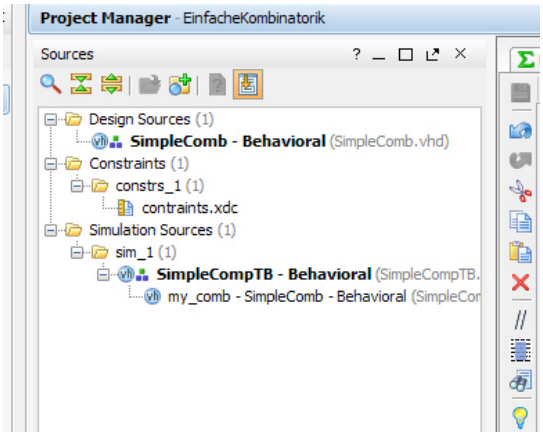
Nun könnten wir bereits simulieren. Da wir aber das Design auch auf einem FPGA live sehen möchten, fügen wir noch eine Constraints-Datei hinzu. Für das **ZyBo-Board** muss die Datei den folgenden Inhalt haben. Sollten Sie ein anderes Board verwenden, müssen Sie die korrekten Anschlüsse für Schalter und LEDs des Boards den Board-Unterlagen (z.B. Schaltplan) entnehmen.

Project Summary | SimpleCompTB.vhd | **contraints.xdc**

D:/Digitaltechnik/EinfacheKombinatorik/EinfacheKombinatorik.srcs/constrs_1/new/contraints.xdc

```
1  #Schalter des ZyBo-Boards
2
3  # welche Logikpegel werden verwendet
4  set_property IOSTANDARD LVCMOS33 [get_ports {sw[*]}]
5
6  # an welchem FPGA-Anschluss liegen die VHDL-Ein-/Ausgänge? z.B. sw(0) => FPGA-Anschluss G15
7  set_property PACKAGE_PIN G15 [get_ports {sw[0]}]
8  set_property PACKAGE_PIN P15 [get_ports {sw[1]}]
9
10 #LEDs des ZyBo-Boards
11 set_property IOSTANDARD LVCMOS33 [get_ports {led[*]}]
12 set_property PACKAGE_PIN M14 [get_ports {led[0]}]
13 set_property PACKAGE_PIN M15 [get_ports {led[1]}]
14 set_property PACKAGE_PIN G14 [get_ports {led[2]}]
15 set_property PACKAGE_PIN D18 [get_ports {led[3]}]
16
```

Im Project Manager sollten Sie die folgenden Einträge sehen:

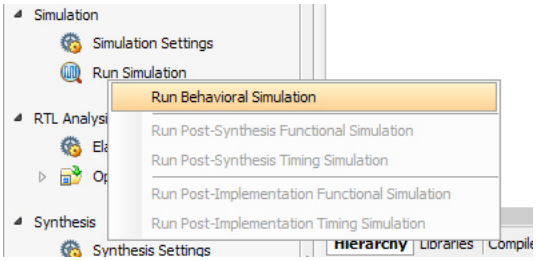


Unter *Simulation Sources* finden Sie übrigens auch die Hierarchie Ihres Designs: Die Testbench instanziiert das Modul *SimpleComb*. Diese Hierarchie wird grafisch angezeigt.

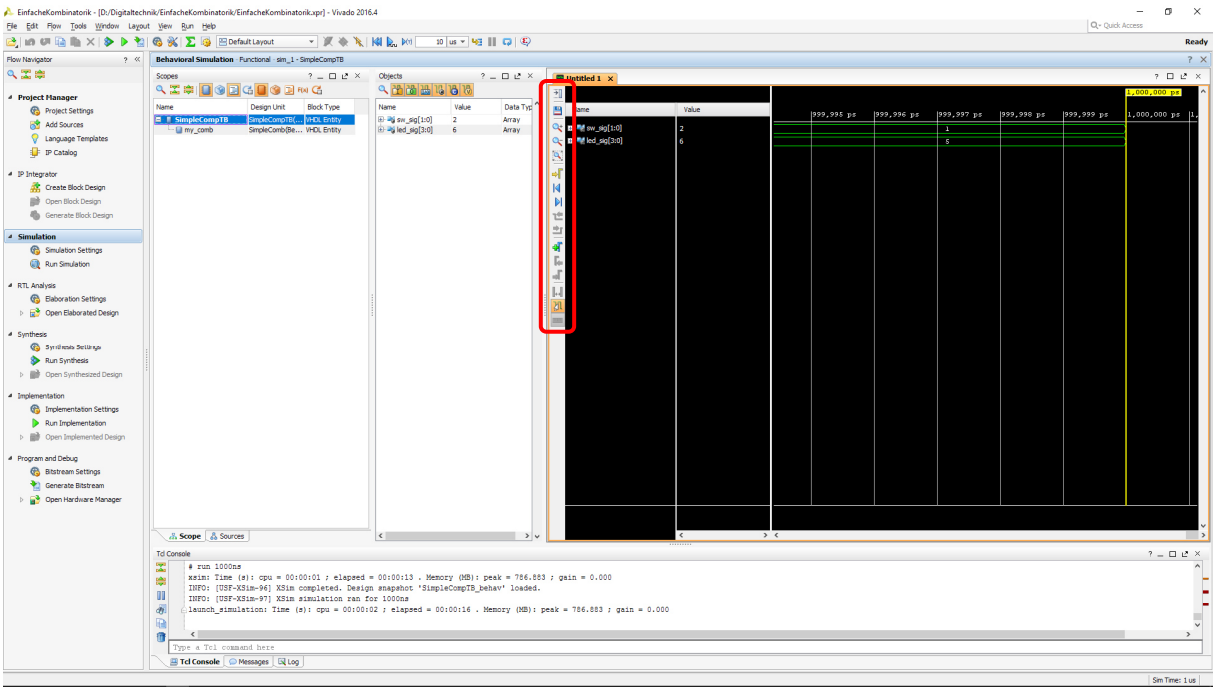
Unter *Design Sources* sehen Sie keine Hierarchie? Na klar: Das was synthetisiert werden soll, besteht - anders als die Simulation - nur aus einem einzelnen VHDL-Modul.

Simulation

Nun wird's aber Zeit für die Simulation. Klicken Sie im *Flow Navigator* auf *Run Simulation* und wählen Sie dann *Run Behavoiural Simulation* aus.



Der Inhalt des rechten Teils des Vivado-Fensters ändert sich: Sie sehen nun die Ausgabe des Simulators. (Falls Sie zurück zur vorigen Ansicht (dem Project Manager) wechseln möchten, klicken Sie links im Flow Navigator einfach auf Project Manager. Anschließend können Sie mit einem Klick auf Simulation wieder in die Simulator-Ansicht wechseln.)



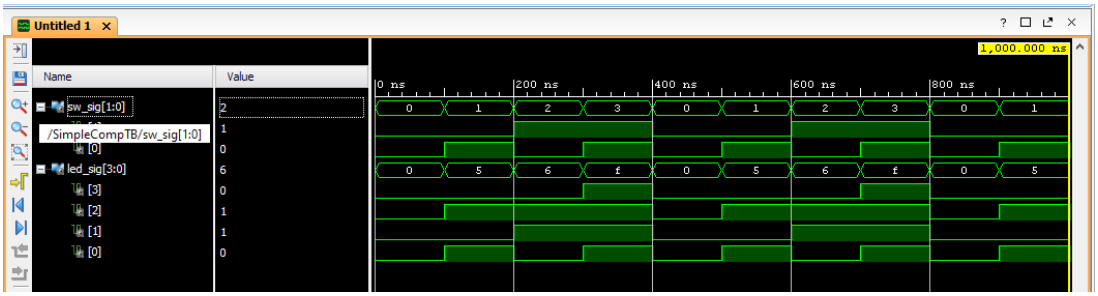
Die Simulation wird automatisch für 1 us gestartet. Daher sehen Sie bereits simulierte Signalverläufe.

Im linken Teil des Simulationsfensters sehen Sie einige hilfreiche Buttons. Durch Anfahren mit der Maus erhalten Sie kurze Hilfetexte zu den einzelnen Buttons. Sie können zum Beispiel Zoomen oder nach Signalwechseln suchen. Probieren Sie die Buttons einfach aus...

Klicken Sie zunächst „*Zoom Fit*" Button. So sehen Sie die gesamten Simulationsergebnisse.

Mit einem Klick auf das kleine + bei den Signalnamen können Sie die Signale „ausklappen" und so die einzelnen Bits betrachten.

Sie sollten etwa Folgendes sehen:



Das Signal *sw_sig* hat den erwarteten Verlauf. Nach dem der Wert 3 (binär: 11) erreicht wurde, wird wieder der Wert 0 an *sw_sig* zugewiesen. Warum? Nun, in der Testbench haben wir einen Prozess verwendet, der nicht beendet (zum Beispiel mit *wait;*) wird. Also wird dieser Prozess immer wieder durchlaufen (vgl. Kapitel 3 im Buch).

Da wir keine selbst-checkende Testbench geschrieben haben, müssen wir die Signale manuell inspizieren. Dies ist bei diesem einfachen Design nicht sonderlich schwer.

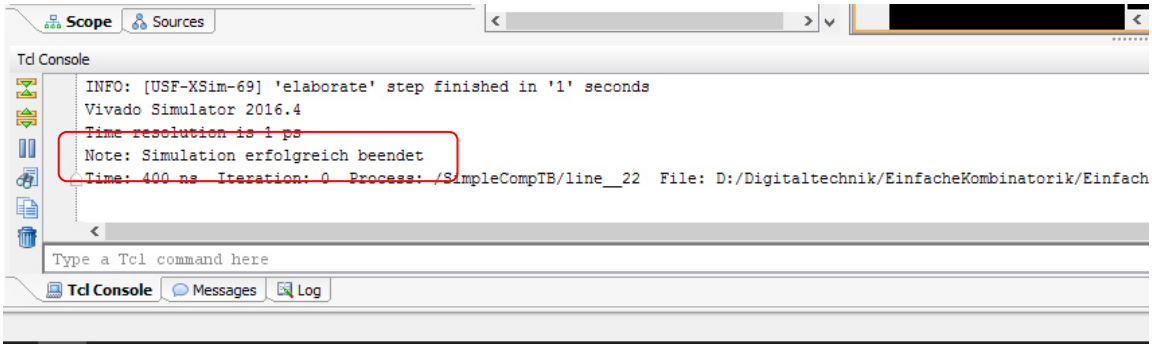
Möchten Sie dagegen eine automatisierte Testbench realisieren, können Sie die Testbench wie folgt mit Assert-Anweisungen erweitern:

```
    -- Ein Prozess, der die Eingänge "stimuliert"
    process
    begin
        sw_sig <= "00";         -- Wert an sw zuweisen
        wait for 100 ns;        -- einige Zeit warten
        assert led_sig = "0000" report "Fehler in der Simulation" severity error;
        sw_sig <= "01";         -- Nächster Wert...
        wait for 100 ns;        -- usw.
        assert led_sig = "0101" report "Fehler in der Simulation" severity error;
        sw_sig <= "10";
        wait for 100 ns;
        assert led_sig = "0110" report "Fehler in der Simulation" severity error;
        sw_sig <= "11";
        wait for 100 ns;
        assert led_sig = "1111" report "Fehler in der Simulation" severity error;
        assert false report "Simulation erfolgreich beendet" severity note;
        wait; -- Ende der Simulation: Prozess wird beendet.
    end process;

nd Behavioral;
```

Da wir den VHDL-Code geändert haben, muss er neu übersetzt werden. Dies geht sehr elegant mit dem *ReLaunch*-Button im oberen Bereich des Fensters:

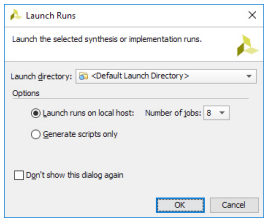
Im unteren Bereich (Tcl Console) erscheinen die Meldungen der Simulation: „Simulation erfolgreich beendet“.



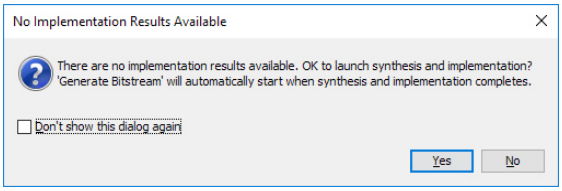
Synthese

Die Erzeugung der binären Datei zur Programmierung des FPGAs erzeugt man am einfachsten durch einen Klick auf *Generate Bitstream* im unteren Bereich des Flow Navigators.

Vivado kann die Synthese durch die Verwendung mehrerer CPUs Ihres PCs beschleunigen („Number of Jobs“)

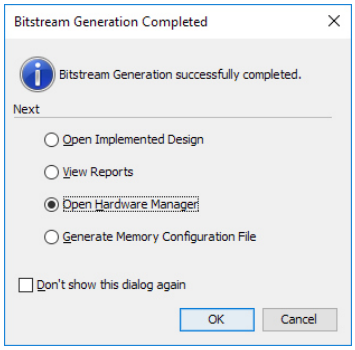


und stellt eigenständig fest, dass die neben der Erzeugung des binären „Bitstreams“ zuvor noch weitere Schritte durchgeführt werden müssen:



Bestätigen Sie den Dialog mit *Yes*.

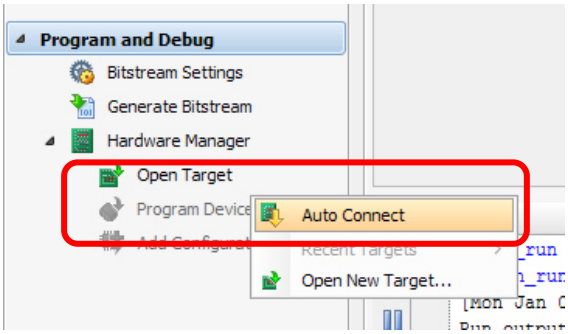
Nach einigen Minuten ist die Erzeugung des Bitstreams abgeschlossen:



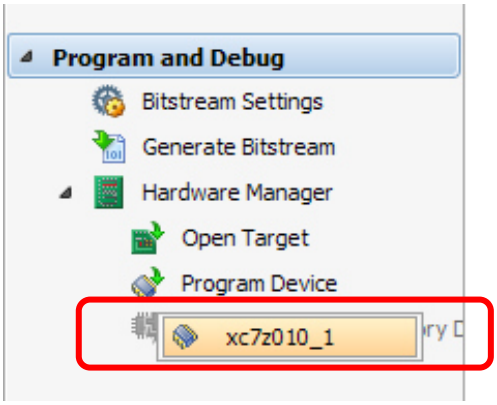
Wählen Sie *Open Hardware Manager* aus, um den Bitstream auf das FPGA zu laden.

Programmierung des FPGAs

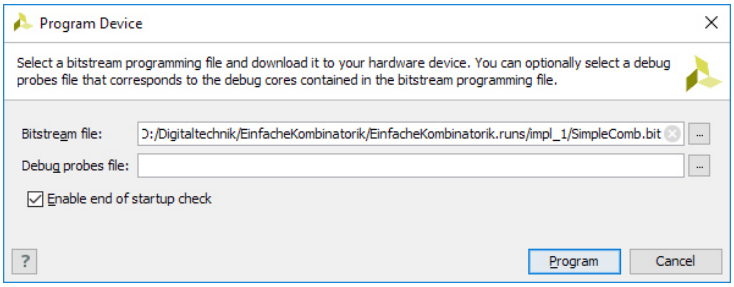
Sie haben das ZyBo-Board über USB mit Ihrem Rechner verbunden und eingeschaltet? Dann können Sie im unteren Bereich des Flow Navigators auf *Open Target* und dann *Auto Connect* klicken.



Vivado verbindet sich nun über USB mit Ihrem Board. Klicken Sie anschließend auf Program Device um das FPGA mit Ihrem Design zu laden. Hierbei wird Ihnen der auf dem Board befindliche Baustein (im Fall des ZyBo-Boards ein xc7z010) zur Auswahl angeboten.



In der nachfolgenden Dialogbox klicken Sie auf *Program* und Ihr Design wird auf das Board übertragen.



Nun können Sie Ihr Design live testen. Ändern Sie die Stellung der Schalter des Boards und kontrollieren Sie, ob sich die LEDs wie erwartet verhalten.

Geschafft! Sie haben Ihr erstes eigenes VHDL-Design simuliert und auf einem FPGA-Board getestet.

Sequentielle Schaltungen

Wenn Sie sequentielle Schaltungen entwerfen möchten, benötigen Sie ein Taktsignal. Das ZyBO-Board stellt Ihnen ein Taktsignal mit einer Frequenz von 125 MHz zur Verfügung.

Um dieses in der Synthese nutzen zu können, müssen Sie der Constraints-Datei die folgenden Zeilen hinzufügen:

```
set_property PACKAGE_PIN L16 [get_ports clk]
set_property IOSTANDARD LVCMOS33 [get_ports clk]
create_clock -add -name sys_clk -period 8.00 -waveform {0 4} [get_ports clk]
```

Für die Simulation fügen Sie die Signaldefinition

```
signal clk : std_logic := '0';
```

in der Testbensch hinzu und erzeugen Sie den Takt mit der VHDL-Zeile:

```
clk <= not clk after 4 ns;
```

Weitere Constraints für das ZyBo-Board

Sie möchten alle Schalter und auch die Taster nutzen? Nachfolgend finden Sie die hierfür benötigten Constraints:

```
# Taster
set_property IOSTANDARD LVCMOS33 [get_ports {btn*}]
set_property PACKAGE_PIN R18 [get_ports {btn[0]}]
set_property PACKAGE_PIN P16 [get_ports {btn[1]}]
set_property PACKAGE_PIN V16 [get_ports {btn[2]}]
set_property PACKAGE_PIN Y16 [get_ports {btn[3]}]

# LEDs
set_property IOSTANDARD LVCMOS33 [get_ports {led[*]}]
set_property PACKAGE_PIN M14 [get_ports {led[0]}]
set_property PACKAGE_PIN M15 [get_ports {led[1]}]
set_property PACKAGE_PIN G14 [get_ports {led[2]}]
set_property PACKAGE_PIN D18 [get_ports {led[3]}]

# Schalter
set_property IOSTANDARD LVCMOS33 [get_ports {sw[*]}]
set_property PACKAGE_PIN G15 [get_ports {sw[0]}]
set_property PACKAGE_PIN P15 [get_ports {sw[1]}]
set_property PACKAGE_PIN W13 [get_ports {sw[2]}]
set_property PACKAGE_PIN T16 [get_ports {sw[3]}]
```

Weitere Constraints finden Sie auf der Seite des Board-Herstellers www.digilentinc.com. Auf der Seite zum Board wird Ihnen eine sogenannte „Master XDC“-Datei zur Verfügung gestellt, die alle Anschlüsse des Boards enthält.