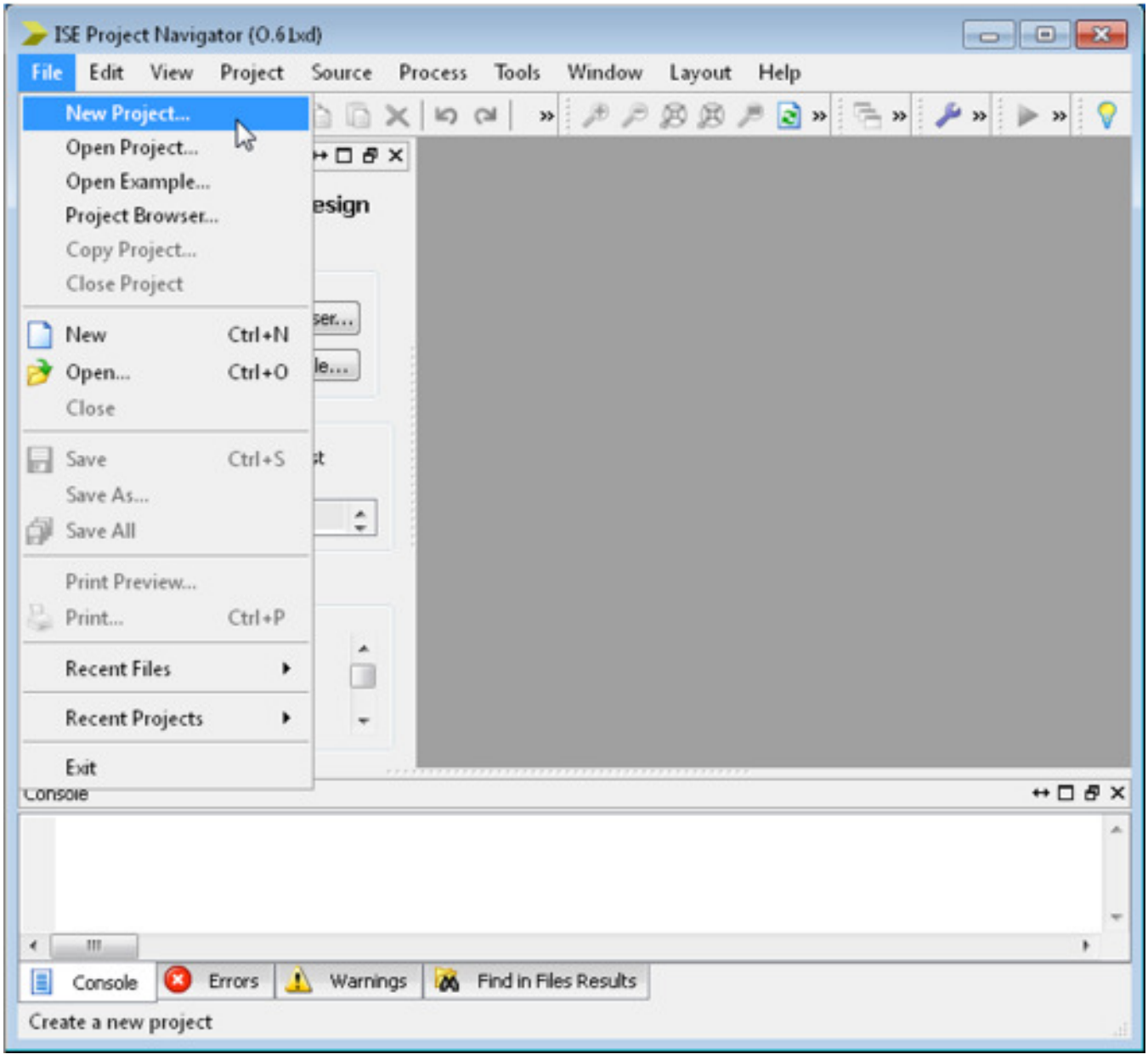
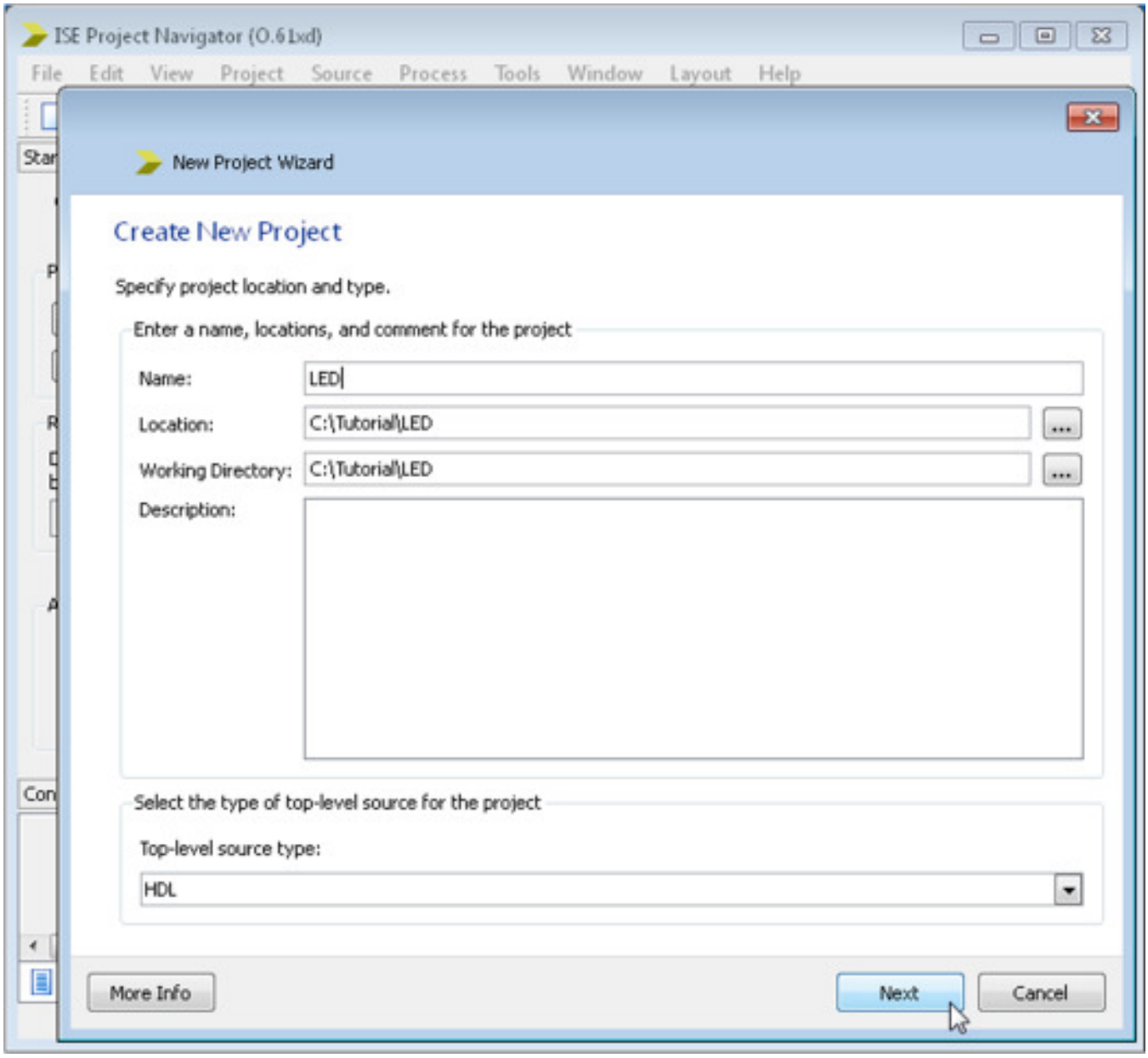
Eine blinkende LED mit Xilinx ISE 13: das “Hello World!“ der Hardware.

Das hier ist eine Schritt-für-Schritt Anleitung, in der gezeigt wird, wie mit Xilinx ISE ein Projekt angelegt, simuliert und anschließend das FPGA konfiguriert wird.

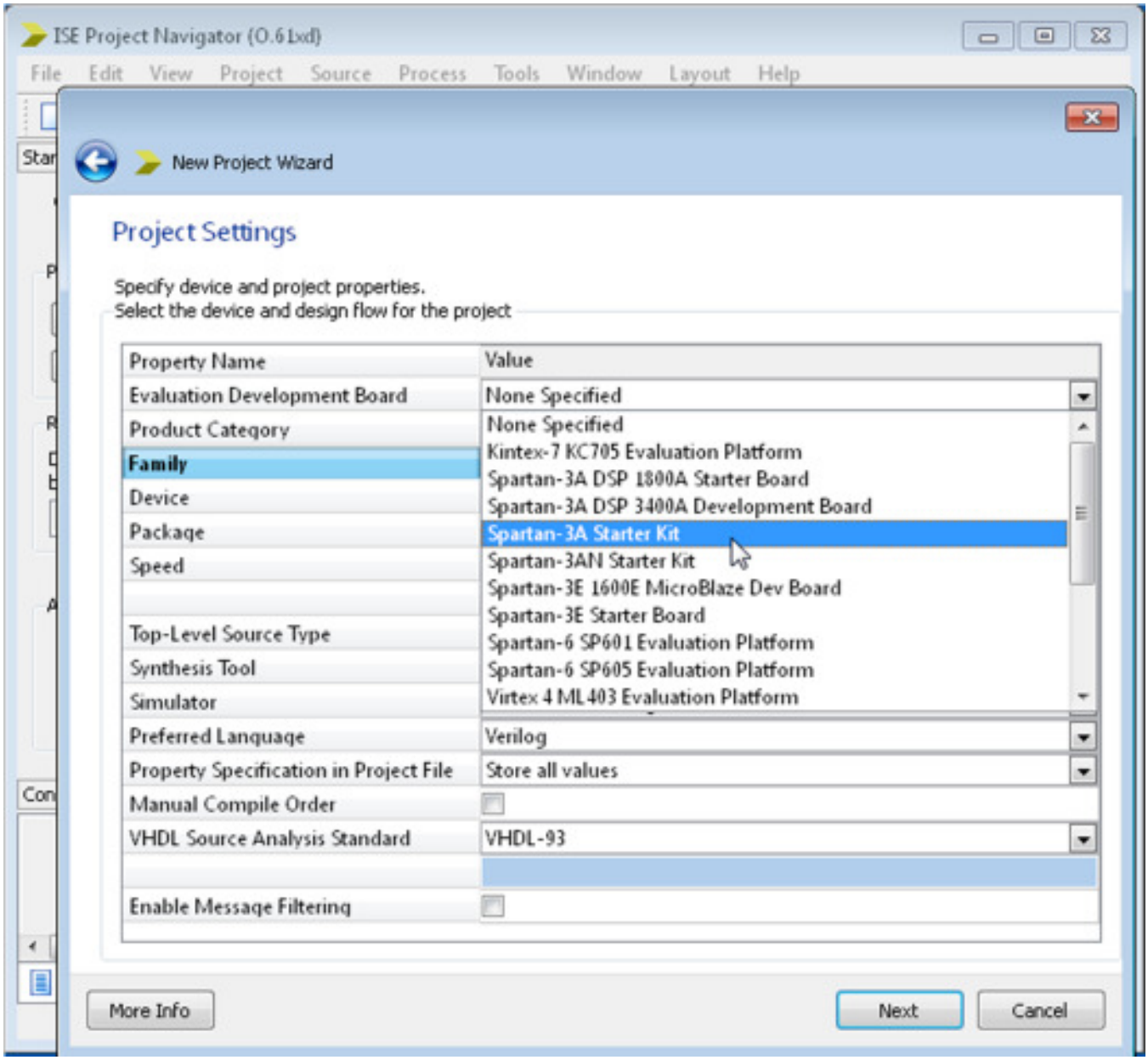
Ich habe hierfür das Spartan3A Board von Xilinx/Digilent verwendet. Weil die Pinzuordnung aber manuell vorgenommen wird, kann jedes andere Board ebenfalls eingesetzt werden.



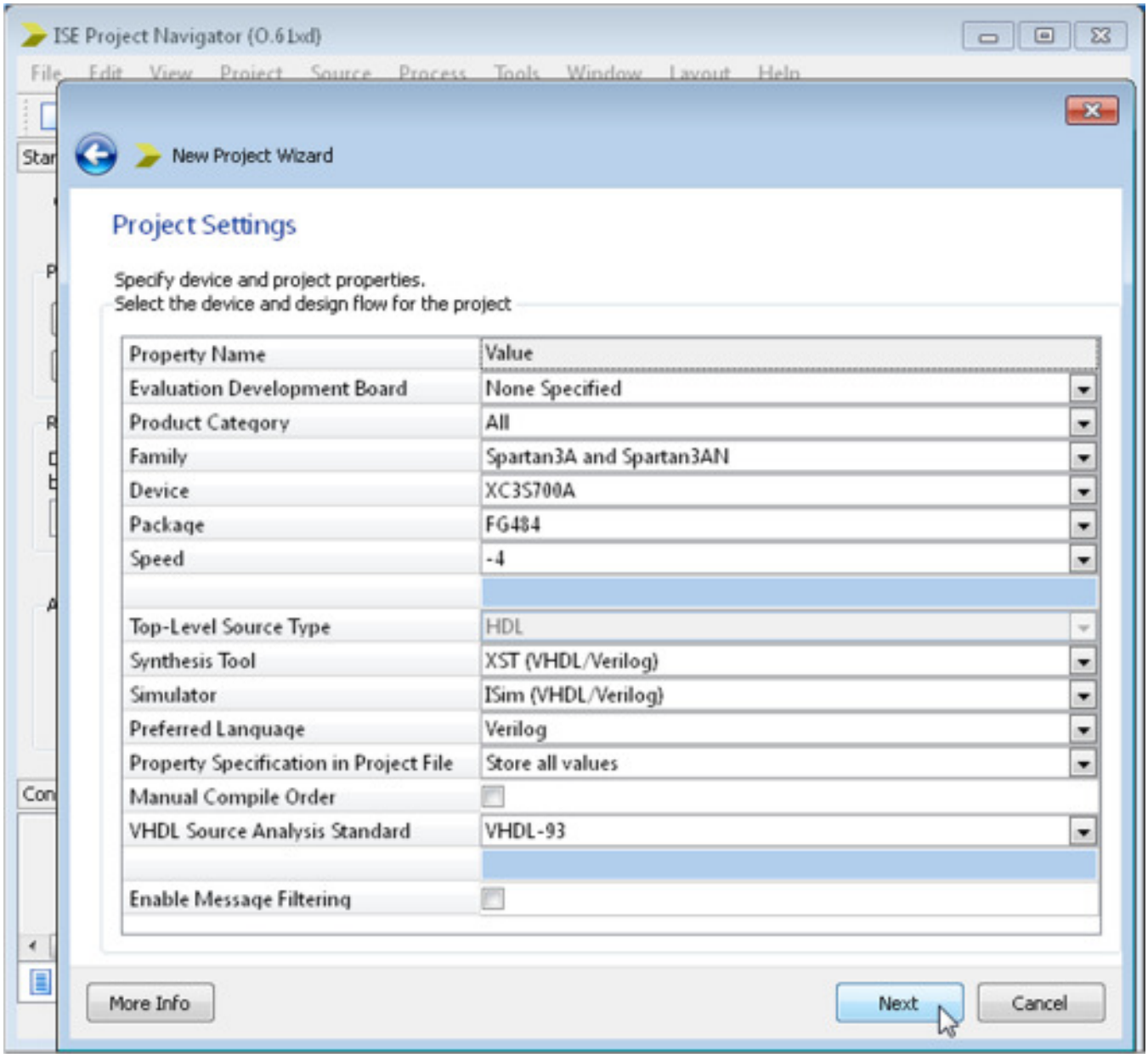
Nach dem Start von ISE wird mit [New Project…] der Wizard aufgerufen.



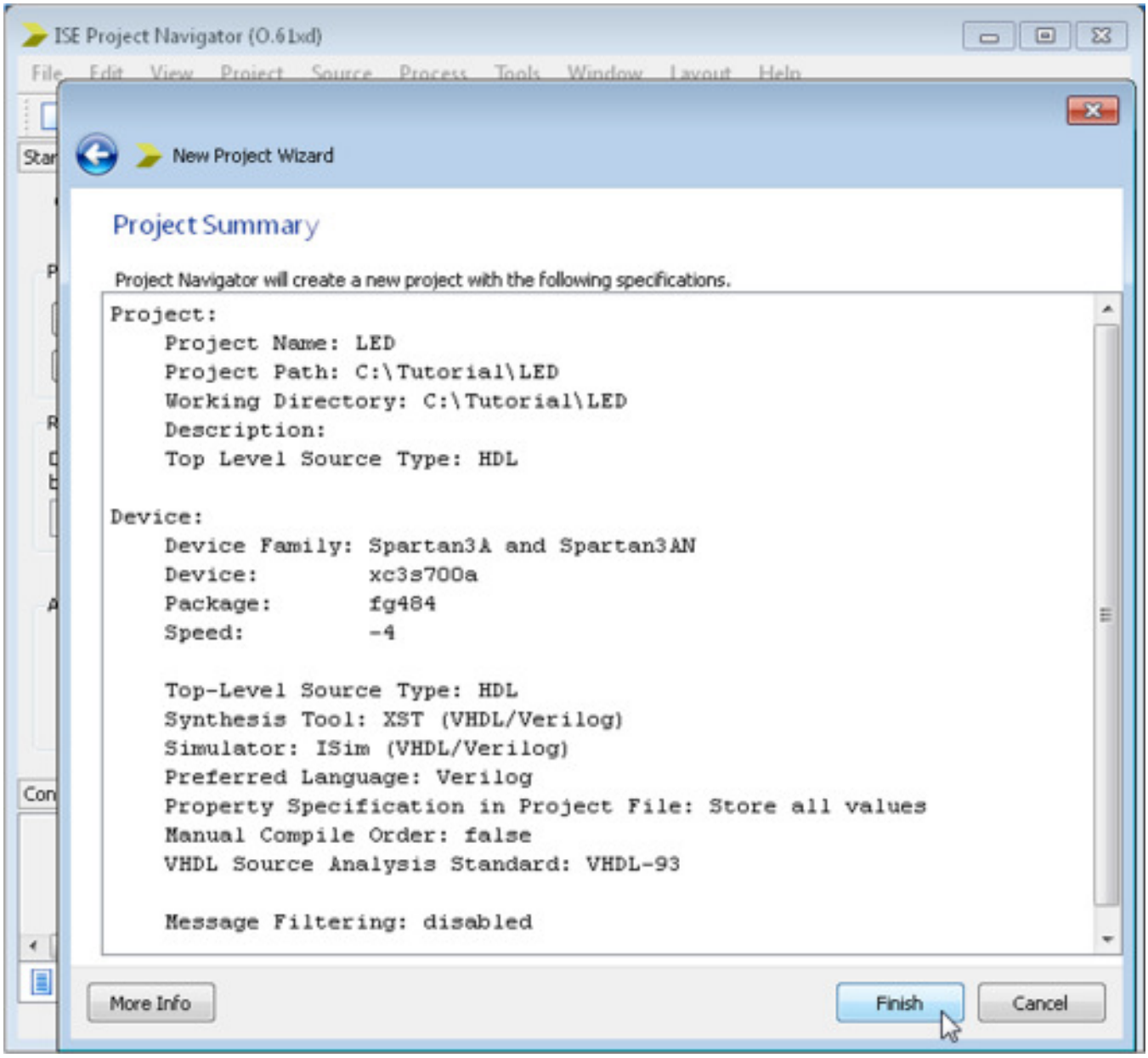
Im Wizard wird jetzt ein Projektname „LED“ angegeben und der Projektpfad ausgewählt.



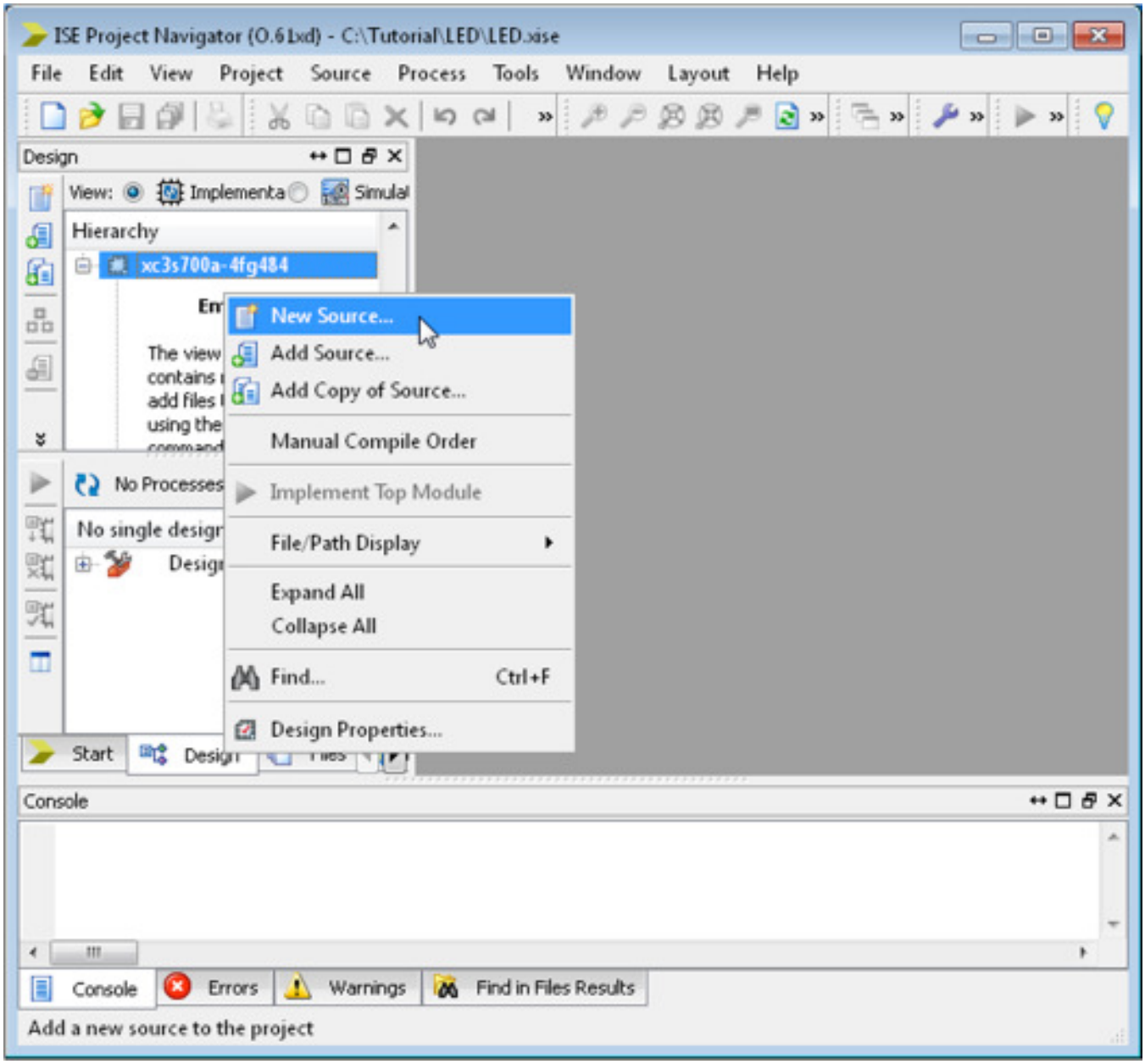
Nun werden die hardwarespezifischen Einstellungen vorgenommen.
Hier könnte man bei [Evaluation Development Board] auch gleich das Spartan-3A Starter Kit ausgewählt werden, dann muss wird automatisch das richtige FPGA eingestellt.
Wir machen das aber manuell und wählen hier [None Specified].



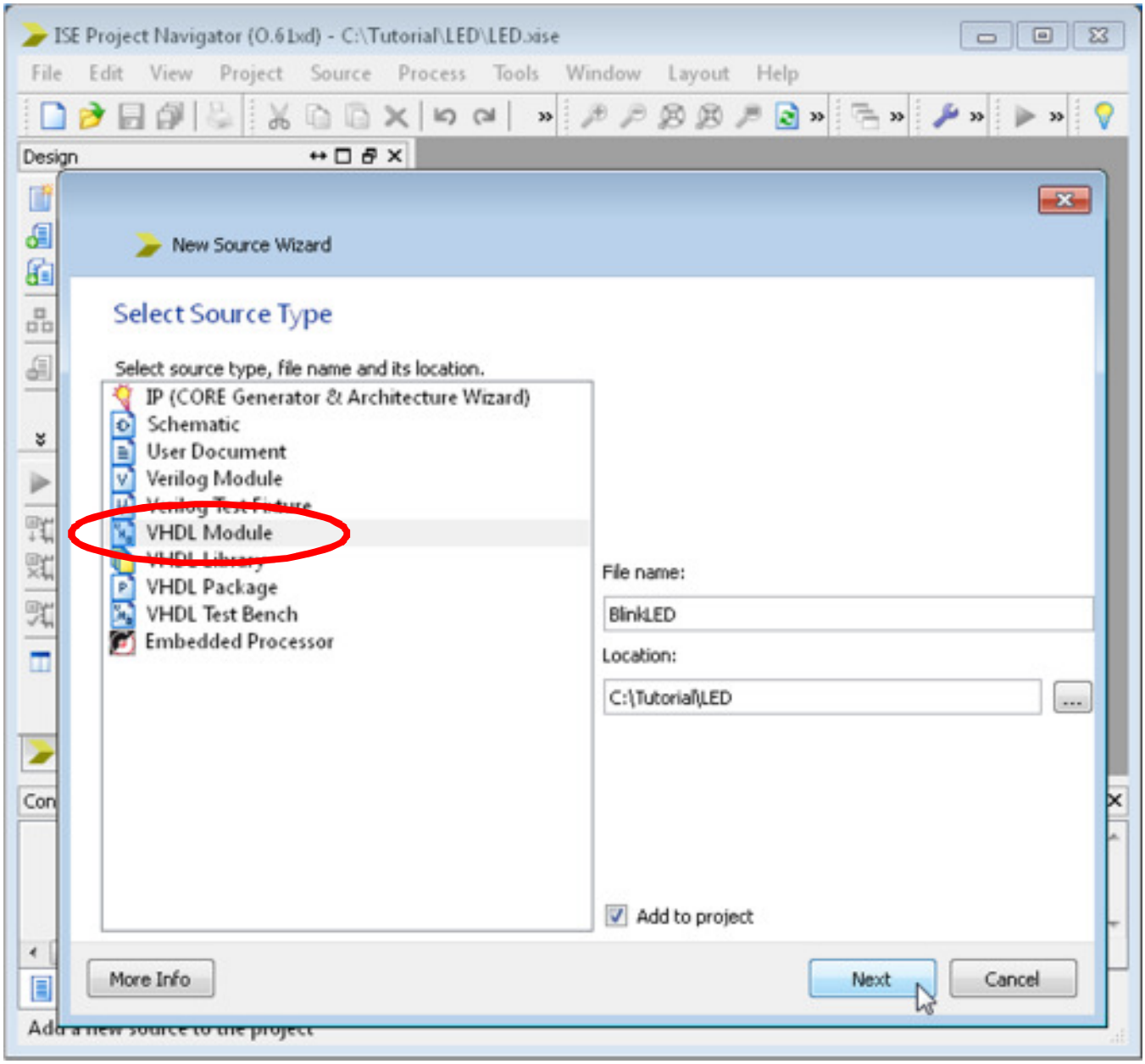
Nach Auswahl der richtigen FPGA-Familie, des Devices, des Gehäuses und des Speedgrades geht es weiter.
Der Speedgrade ist hier nicht so wichtig, es wird noch nicht das Letzte aus dem Baustein herausgekitzelt… ☺



In einer Zusammenfassung wird gezeigt, was alles ausgewählt wurde.
Mit [Finish] wird das neue, noch leere Projekt angelegt.

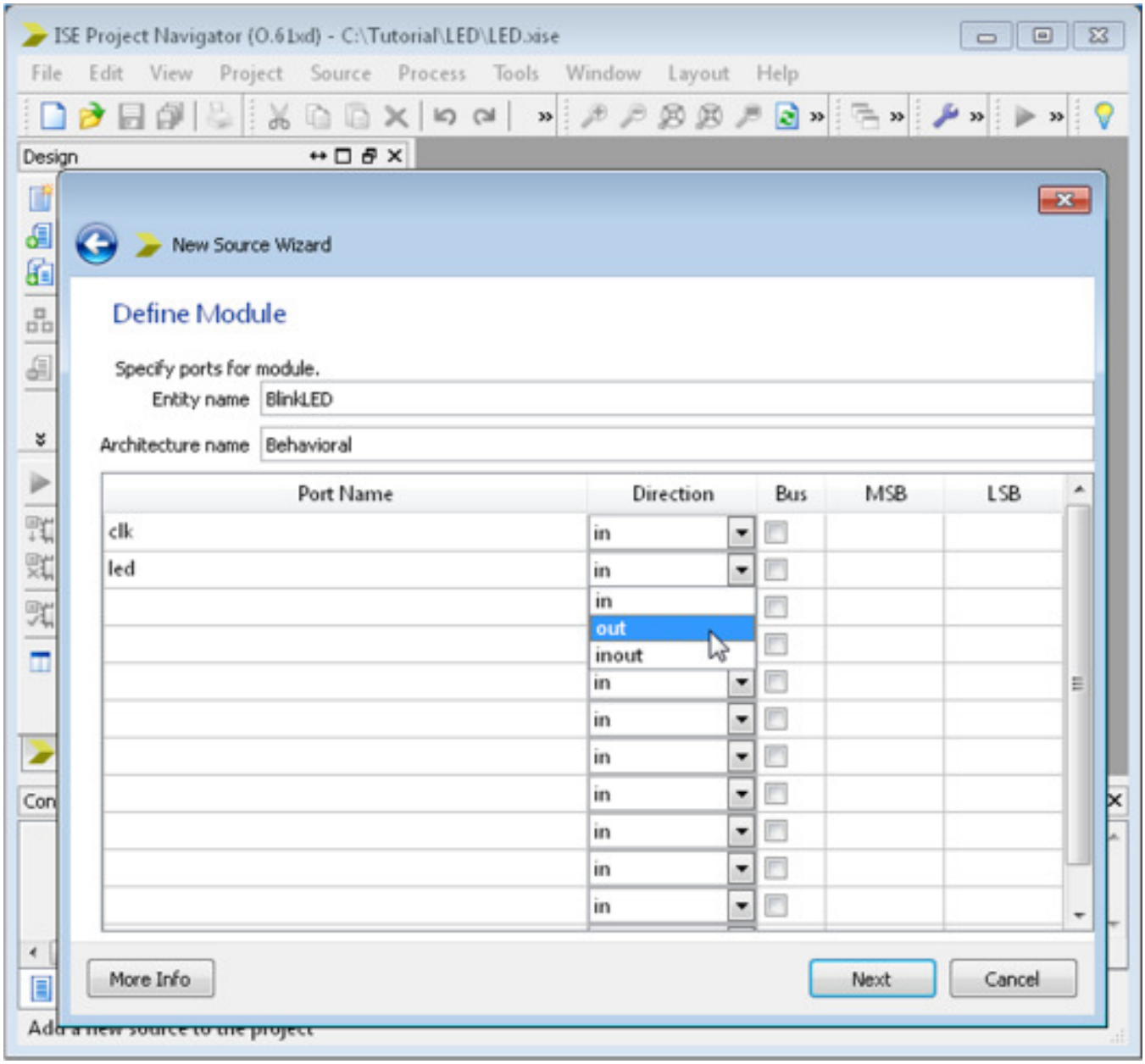


Zum neuen Projekt muss jetzt erst mal eine Beschreibungsdatei hinzugefügt werden.
Wir machen das über Rechtscklick auf das Device und dann [New Source…].
Hier ist wichtig, dass der Designflow für die Implementation aktiviert ist (roter Kreis).



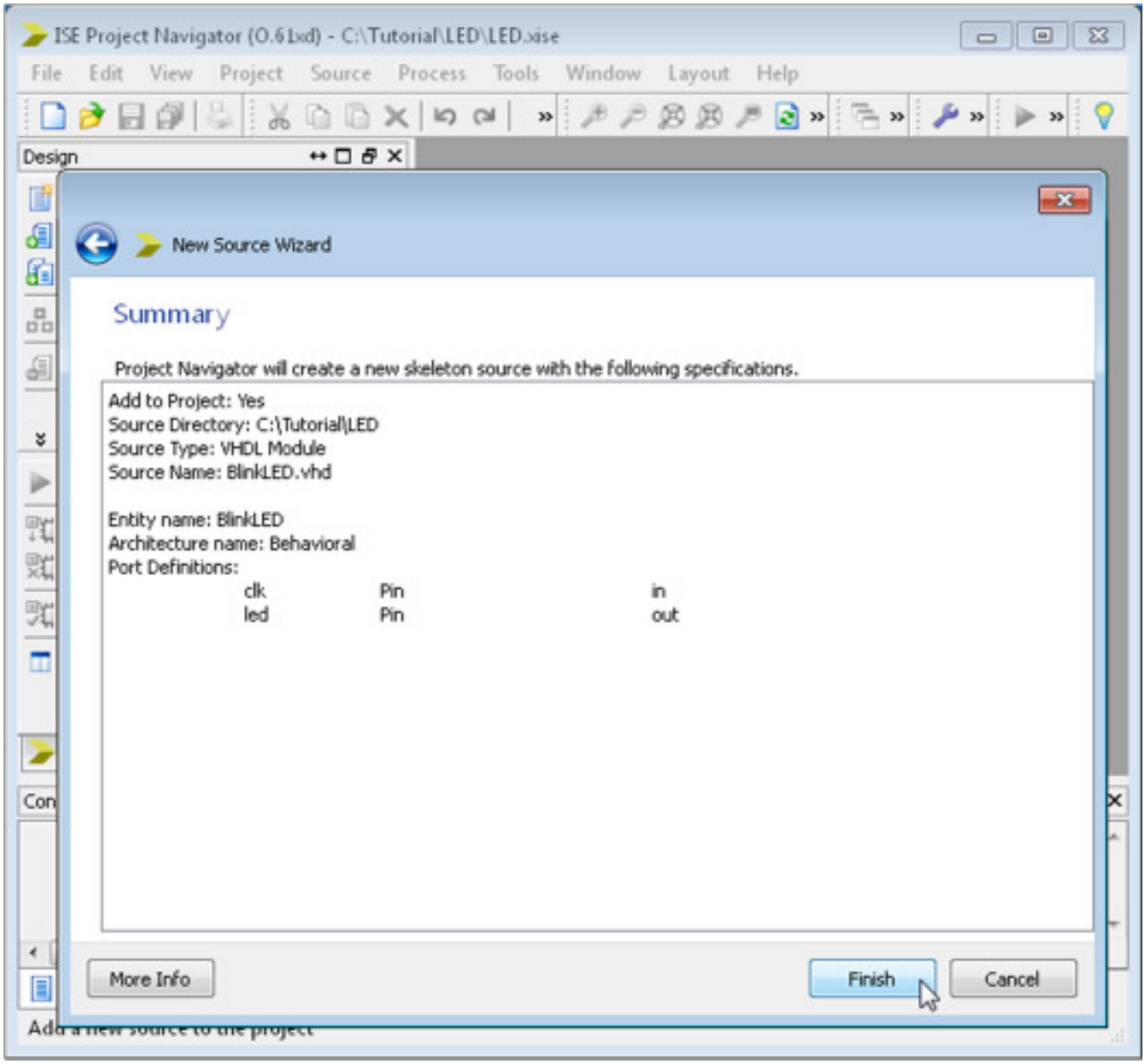
Damit wird der „New Source Wizard“ aufgerufen.

Wir möchten eine neue VHDL-Datei (VHDL-Module) anlegen und geben einen passenden Namen ein.

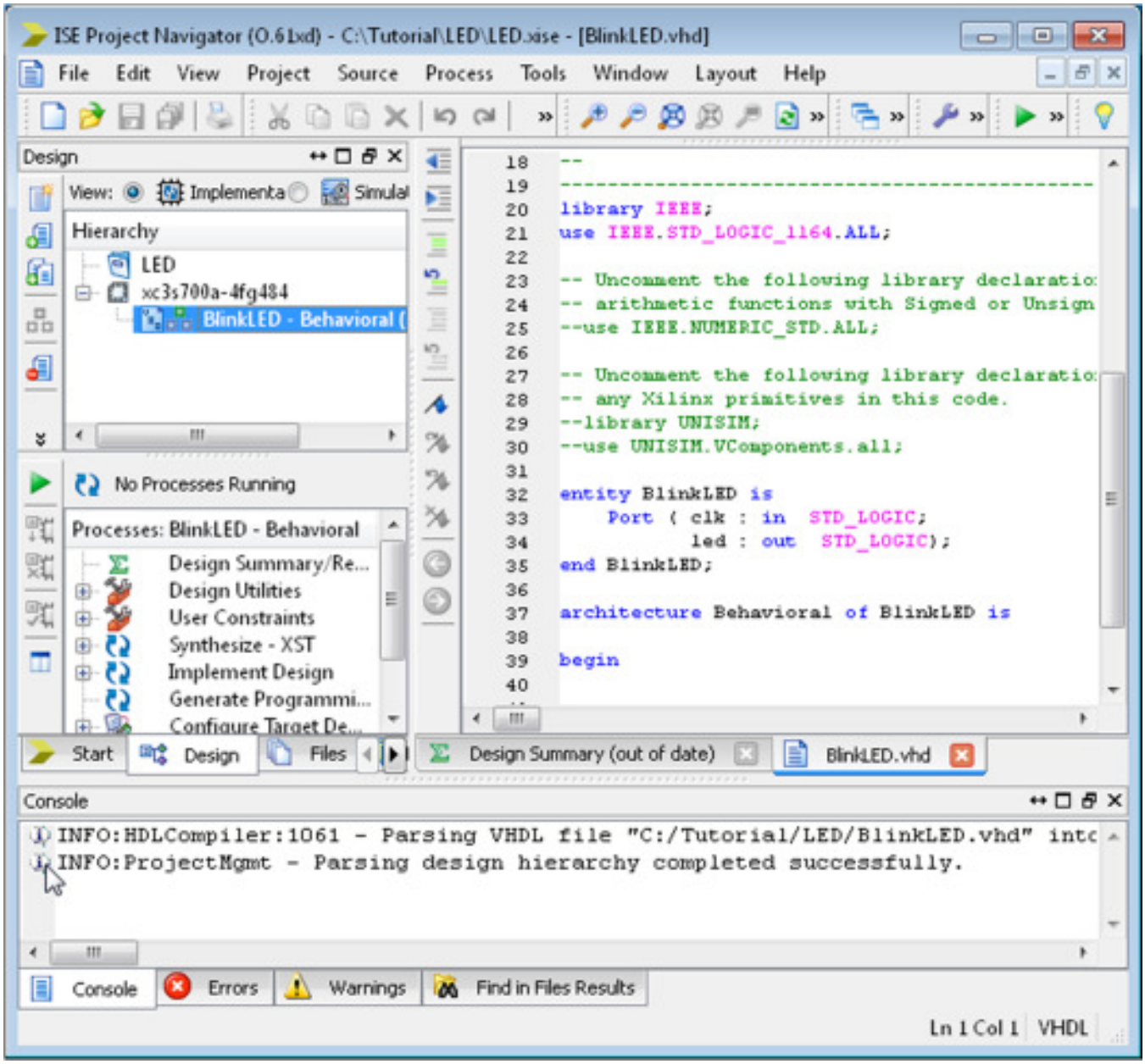


Dann werden die Portnamen und die Richtung eingegeben.

Bei den Portnamen aufpassen, dass nicht Schlüsselwörter wie „in“, „out“ oder der Entity-Name (hier „BlinkLED“) verwendet werden.

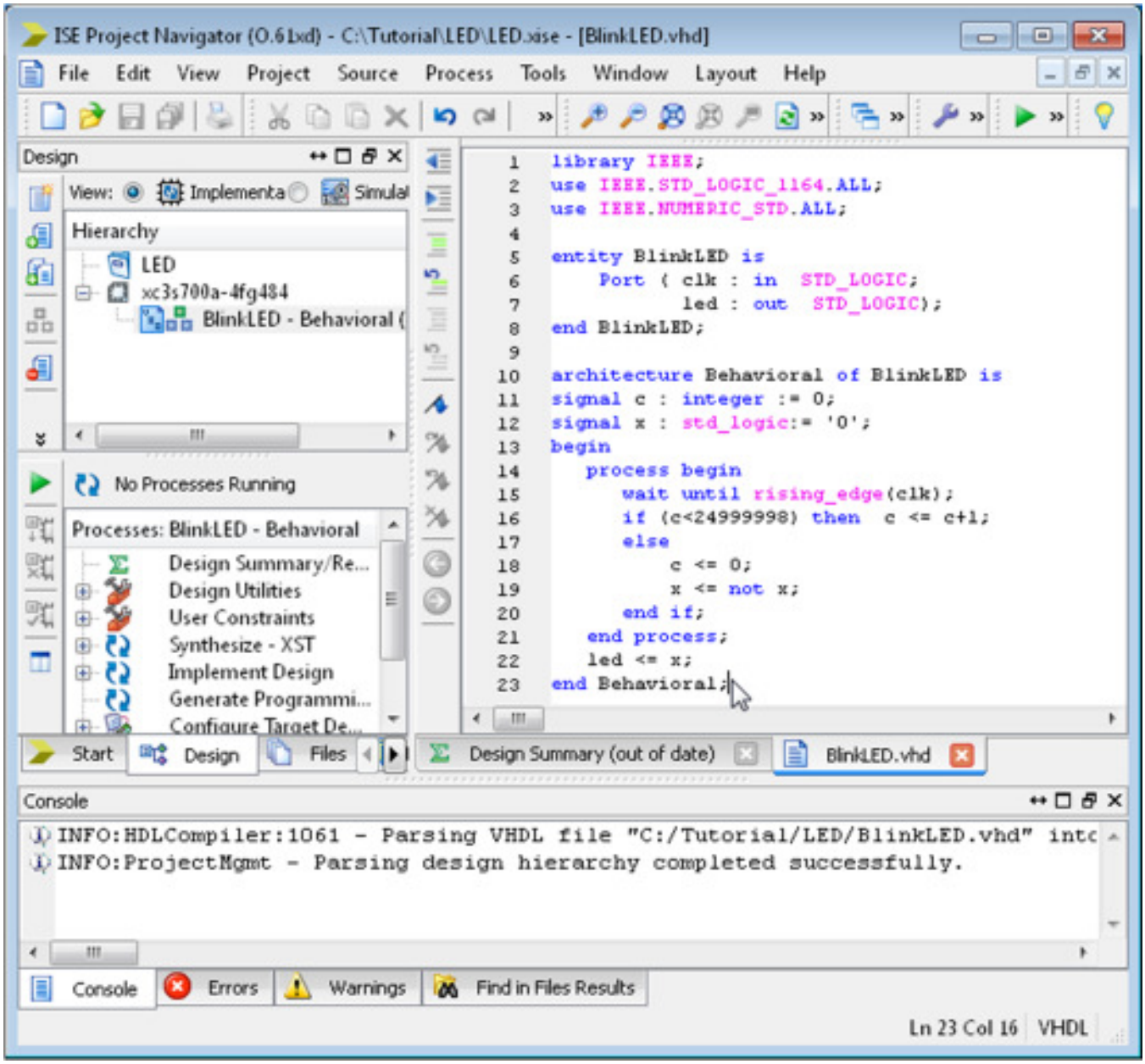


Nach einer kurzen Zusammenfassung und dem Druck auf [Finish]…

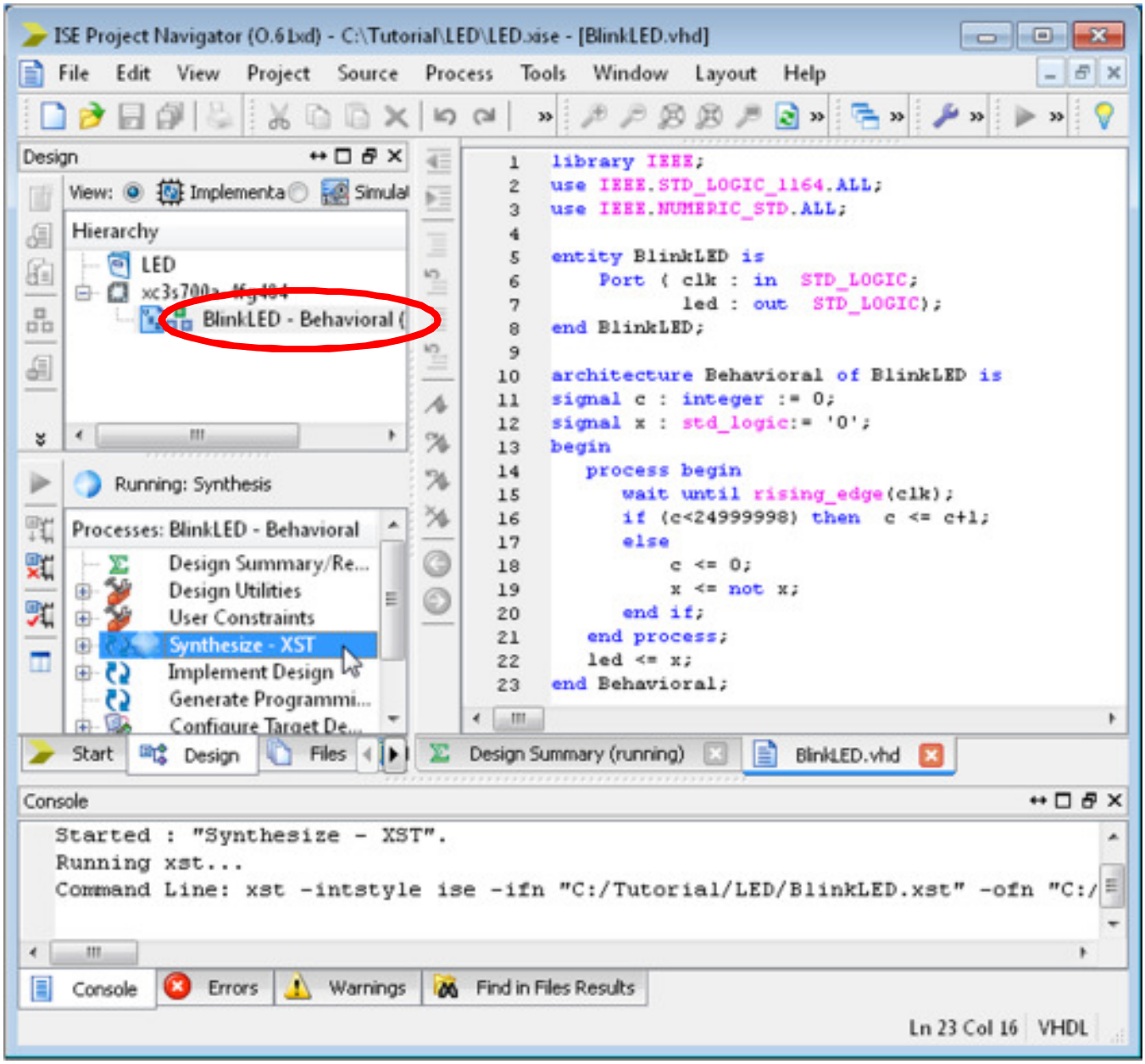


… wird ein Quelltext angelegt.
Nach etlichen Zeilen unnötiger Kommentare folgt dann der eigentliche VHDL-Körper.
Wer diesen Default-Quelltext anpassen will, kann sich mal die Anleitung auf
http://www.lothar-miller.de/s9y/archives/27-Xilinx-New-Source.html
genauer ansehen. Dort ist beschrieben, wie der Script, der die VHDL-Datei erzeugt, den Wünschen entsprechend geändert werden muss.

Beim Anlegen und jedesmal beim Speichern des Quelltextes wird dieser geparst und grobe Fehler (doppelte Namen, Schlüsselwörter als Ports…) angemeckert.

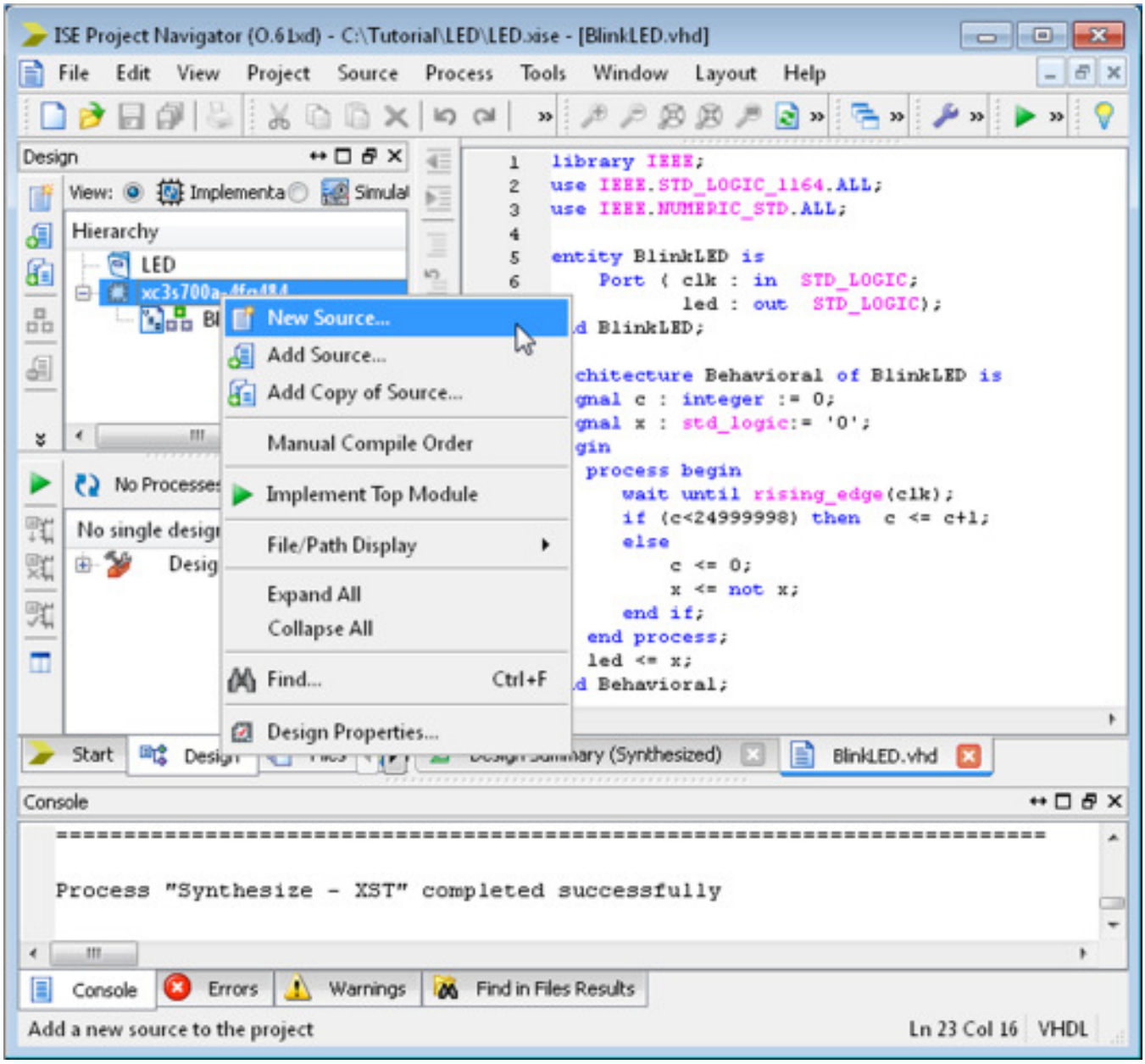


Jetzt kann der Code für eine blinkende LED eingegeben werden (siehe Anhang).

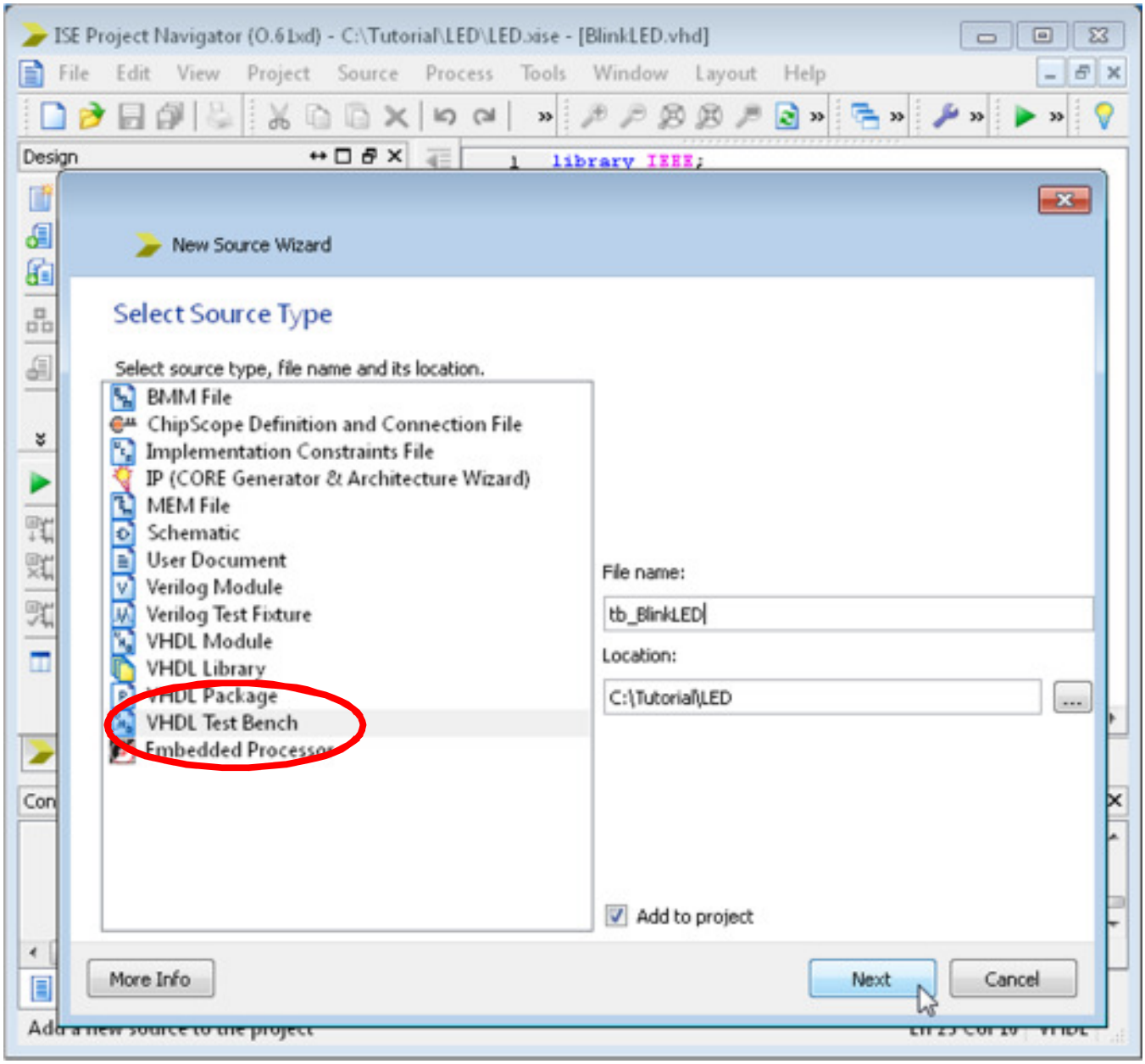


Nach Eingabe des Codes kann die Synthese z.B. mit einem Doppelclick auf [Synthesize – XST] gestartet werden.
Alternativ wäre auch ein Rechtsclick auf [Synthesize – XST] und anschließender Auswahl von [RUN…] möglich.

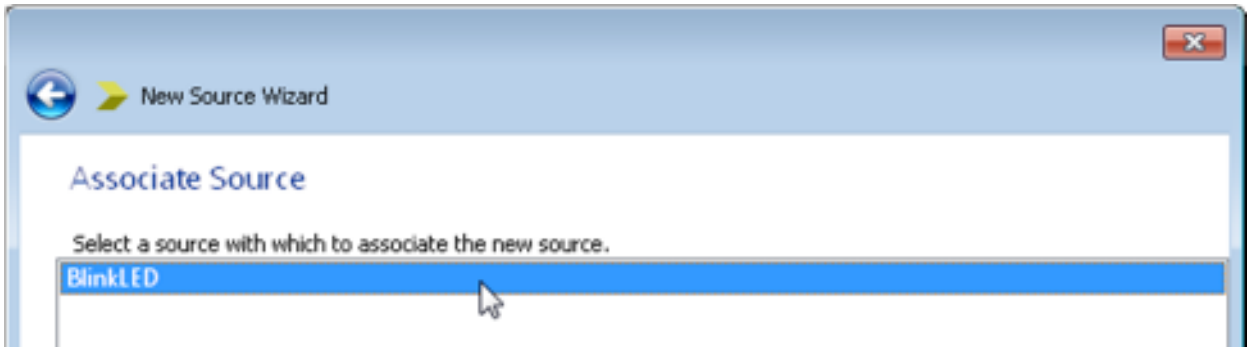
Anlegen einer ultrasimplen Testbench und eine Simulation



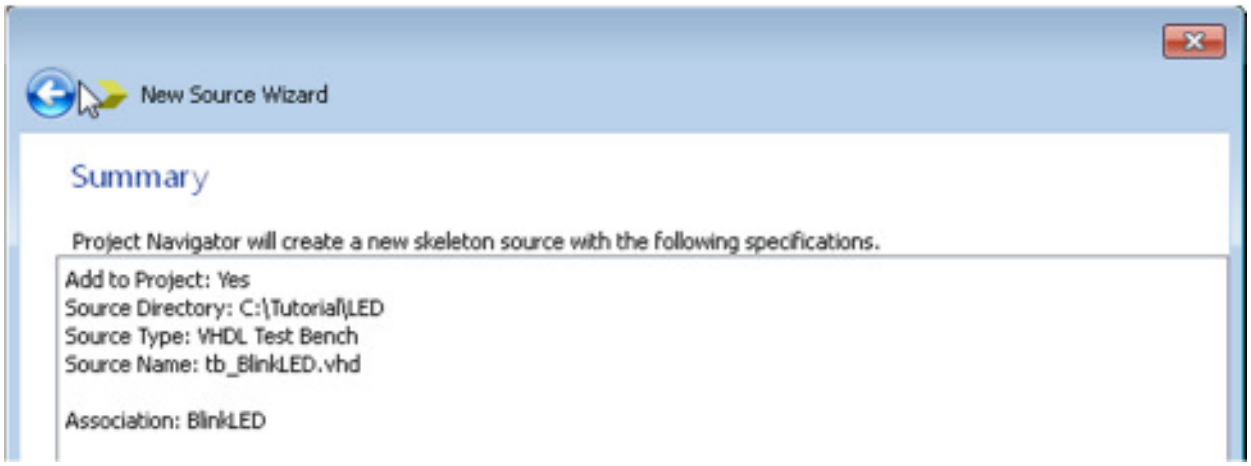
Die Testbench wird wie zuvor der VHDL Code mit dem „New Source Wizard“ angelegt.
Dazu wird mit einem Rechtsclick auf das Device [New source…] ausgewählt.



Dieses Mal geht es um eine [VHDL Test Bench]. In der Liste wird der entsptrechende Eintrag aktiviert und ein passender Name für die Testbench eingetragen.

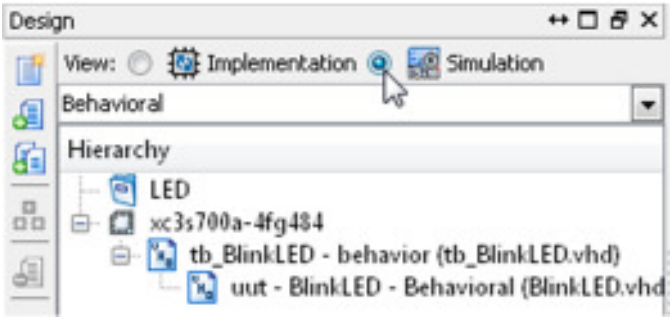


Im nächsten Schritt wird das zugehörige VHDL-Modul ausgewählt.
In unserem kleinen Projekt gibt es natürlich nur 1 VHDL Modul zur Auswahl.

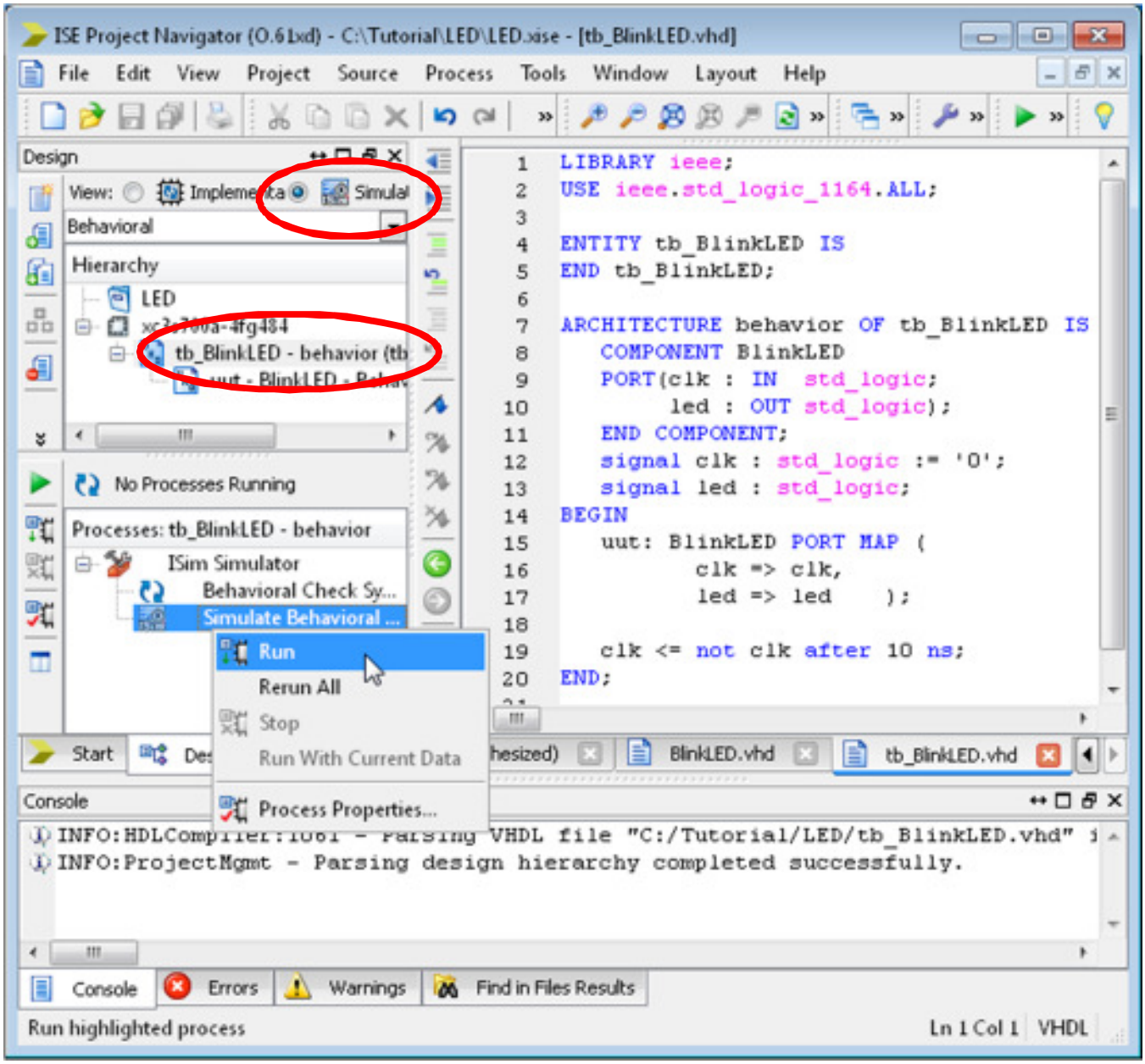


Nach dessen Auswahl zeigt der Wizard eine Zusammenfassung an.

Jetzt muss der Designflow von der Implementation auf die Simulation umgeschaltet werden:

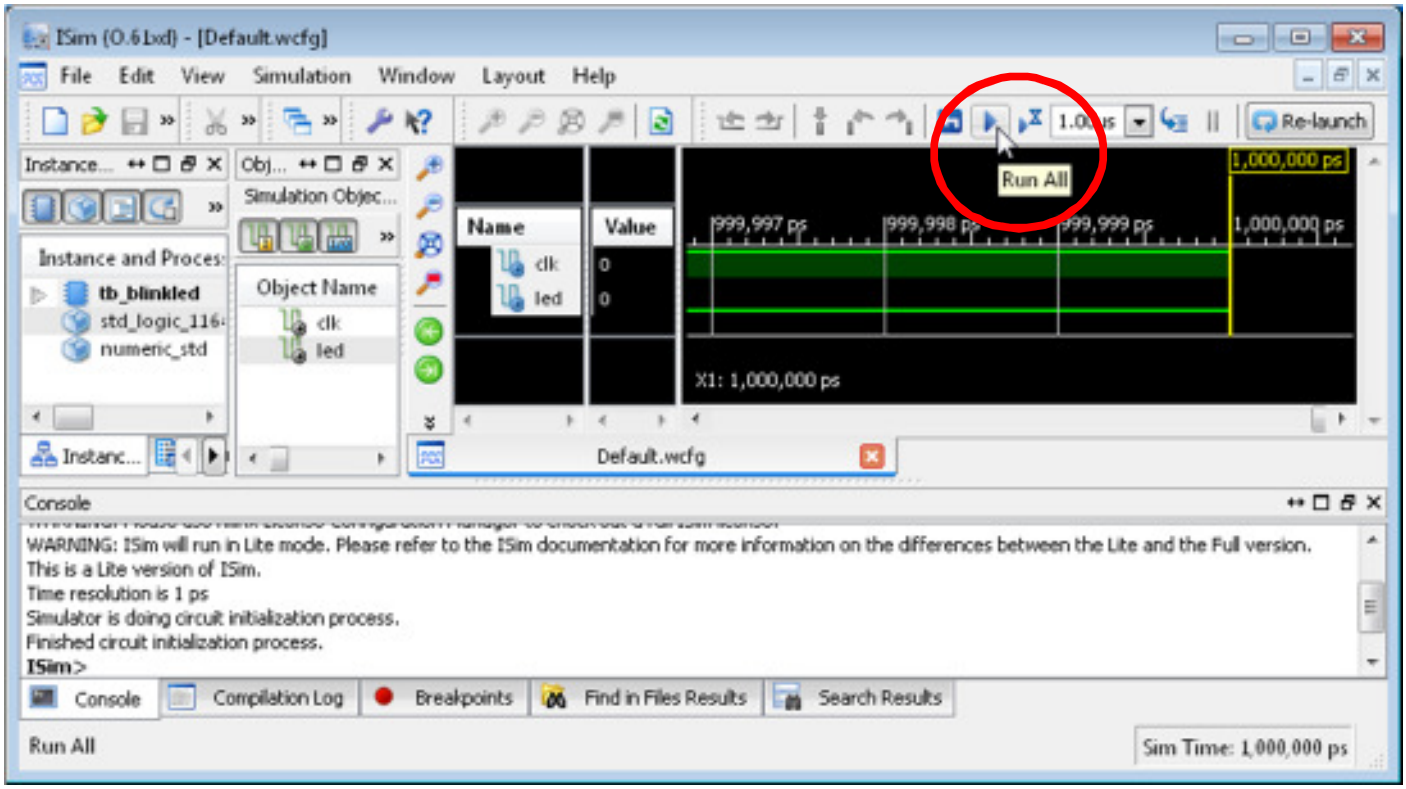


Dadurch nimmt die Testbench den obersten Platz in der Hierarchie ein, das VHDL-Modul ist an die Testbench angehängt.

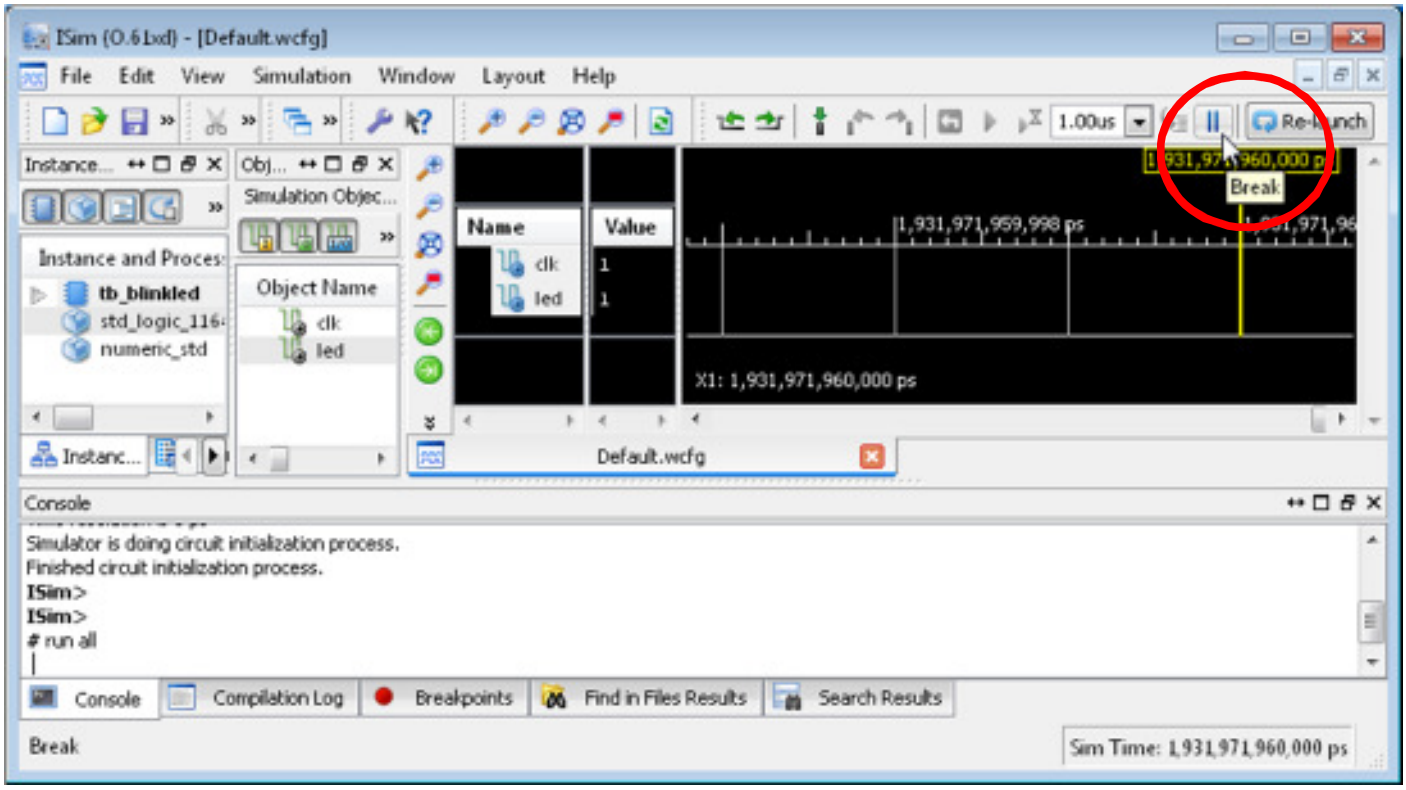


Jetzt kann die automatisch erstellte Testbench-Vorlage entsprechend der gewünschten Testumgebung angepasst werden. Hier soll mal einfach nur ein 50MHz Takt an den clk Eingang angelegt werden.

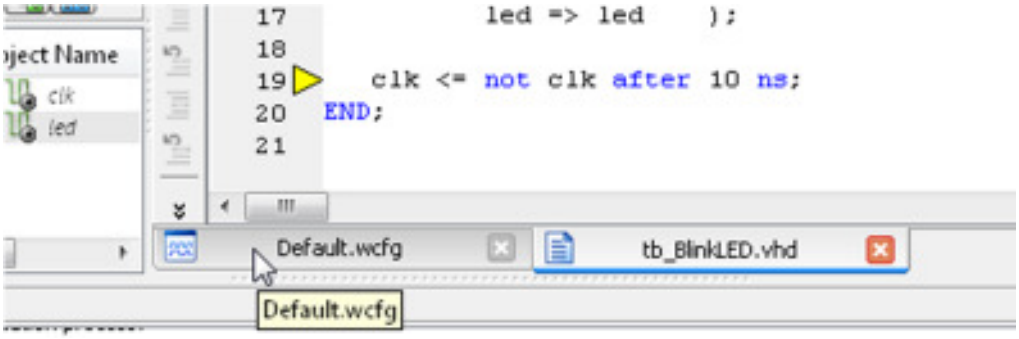
Nach erfolgter Anpassung kann die Simulation begonnen werden:
Dazu muss im Hierarchy-Fenster Testbench selektiert sein.
Darauf erscheint das Prozessfenster [ISim Simulator], wo die Verhaltenssimulation mit [Simulate Behavioral Modell] mit [Run] gestartet werden kann.



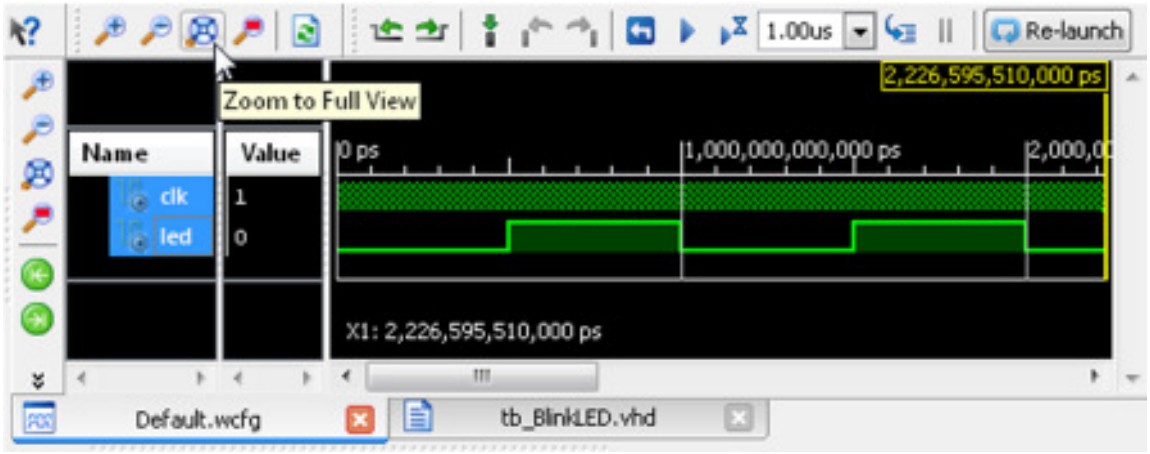
Der Simulator wird gestartet und läuft für 1us (Defaulteinstellung). Unsere LED tut natürlich noch gar nichts. Deshalb lassen wir den Simulator mit Run All einfach mal weiterlaufen.



Nach einigen zigtausend Nanosekunden Simulationszeit halten wir den Simulator wieder an. Dabei nicht ungeduldig werden: in der Realität hat diese Simulation dann je nach Rechner gute 3-5 Minuten gedauert.



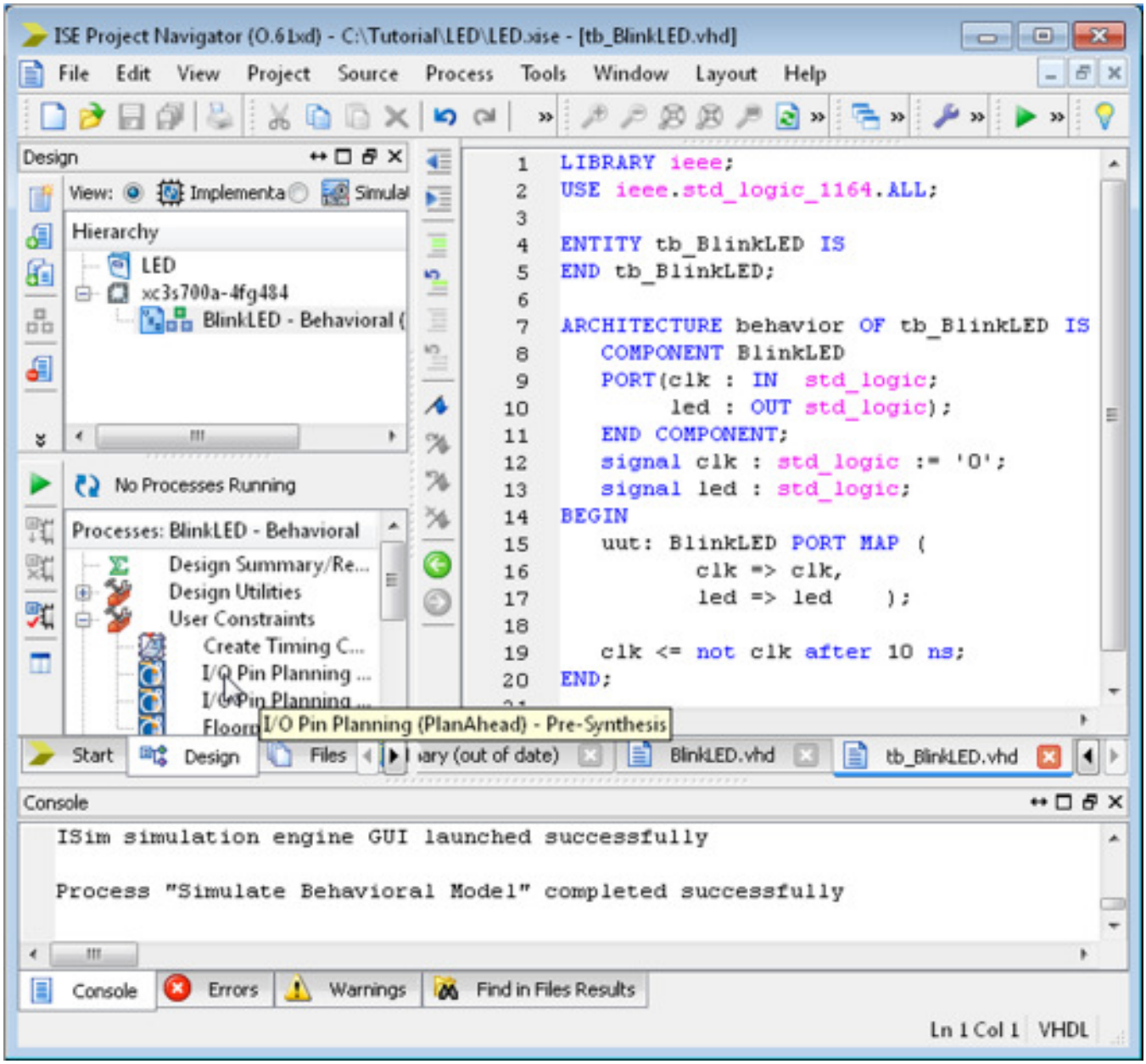
Der Simulator wird irgendwo im VHDL Code anhalten. Deshalb muss jetzt erst mal auf den Waveform Tab zurückgewechselt werden.



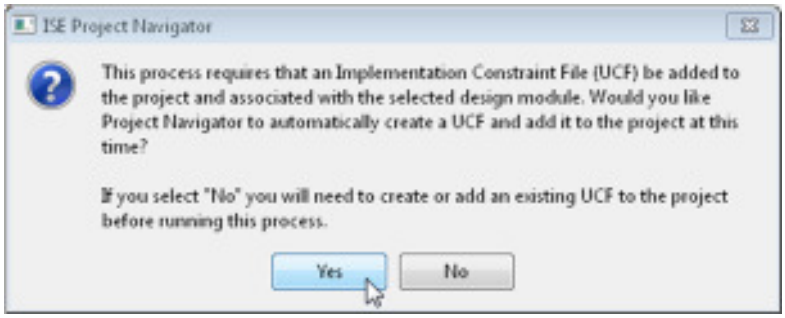
Jetzt können wir mit dem Button [Zoom to Full View] ansehen, ob die LED etwas tut.
Und tatsächlich: wie erwartet ist sie abwechselnd 500ms low und 500ms high.

Nochmal zu Klarstellung:
Diese Testbench hat ihren Namen eigentlich nicht verdient, weil dort nichts automatisch getestet wird. Es ist eigentlich nur eine Stimulidatei für den Takt. Erst mit Prüfungen (Assertions) wird automatisch getestet.

Pinzuordnung mit PlanAhead. Erstellen einer UCF-Datei

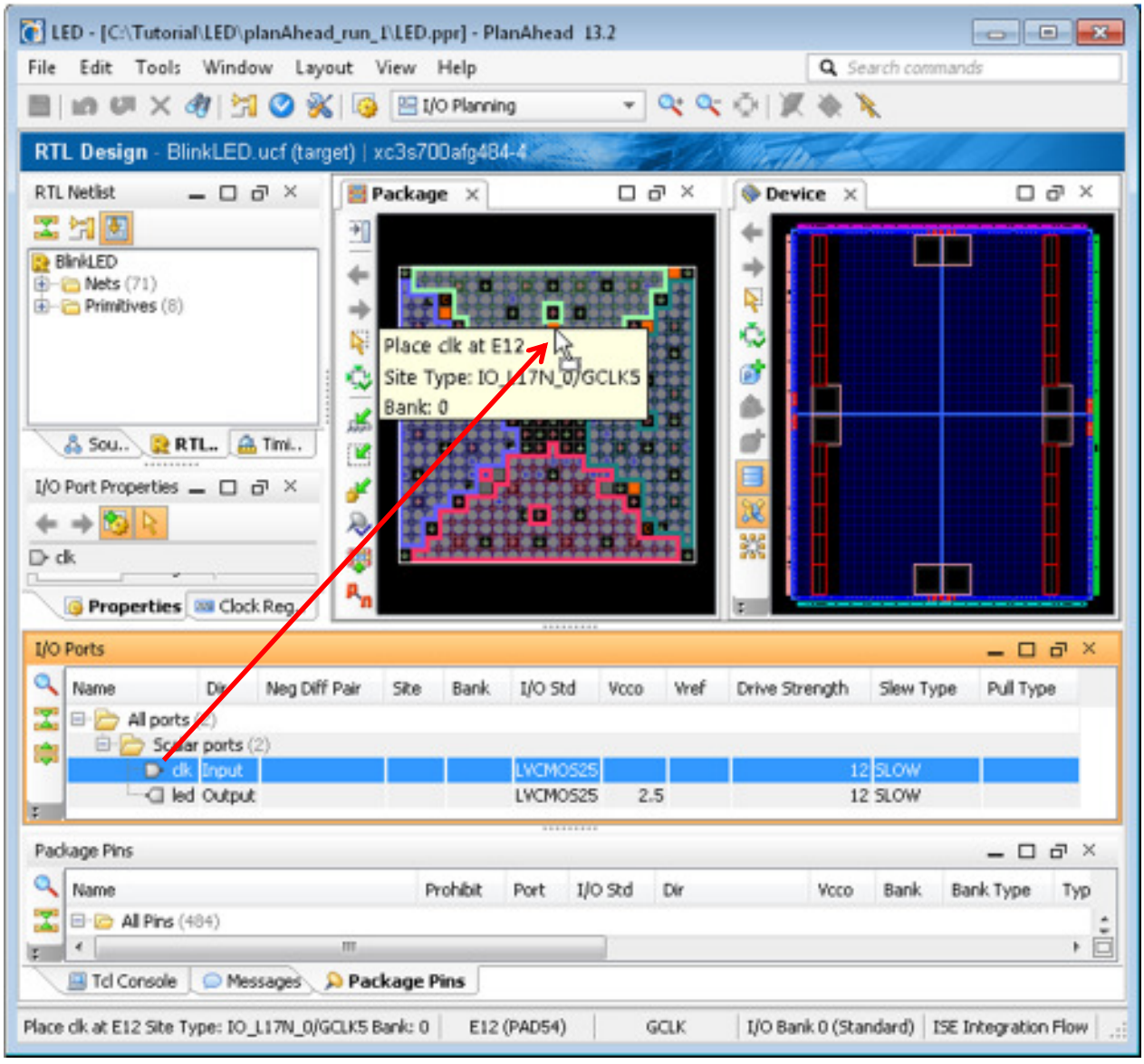


Wenn die Simulation gezeigt hat, dass das Design funktional in Ordnung ist, kann Hardware implementiert werden. Dazu wird der Designflow wieder auf [Implementation] umgeschaltet, und bei den [User Constraints] die IO-Pinzuordung aufgerufen. Dies geschieht mit Aufruf von [I/O Pin Planning (PlanAhead)].

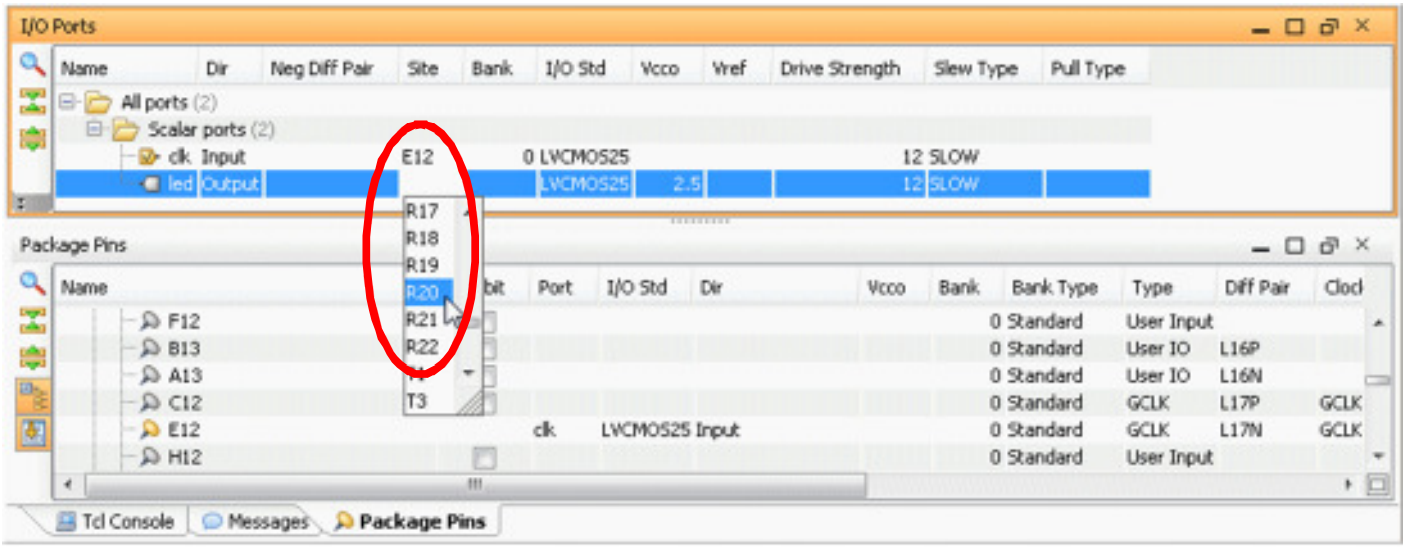


Weil dem Design noch kein User-Constraints-File UCF zugeordnet ist, kommt ein Hinweis, dass ein solches benötigt und dem Projekt zugefügt wird.

Jetzt kommt die Verknüpfung der Signale in der Port-Liste der Top-Entity an reale FPGA-Pins. Auf dem von mir verwendeten Spartan 3A Board ist der 50MHz Oszillator am Pin E12, und die LED0 am Pin R20 angeschlossen.
Hier lohnt sich aber immer ein Blick in den Schaltplan, denn durch falsch zugeordnete Pins kann man durchaus das FPGA oder externe Bausteine zerstören (in dem z.B. zwei Ausgänge miteinander verbunden werden).



Die Pinzuordnung kann nun z.B. per Drag&Drop (s.o.)geschehen, oder …



… es kann ein Pin aus einer Liste ausgewählt werden.

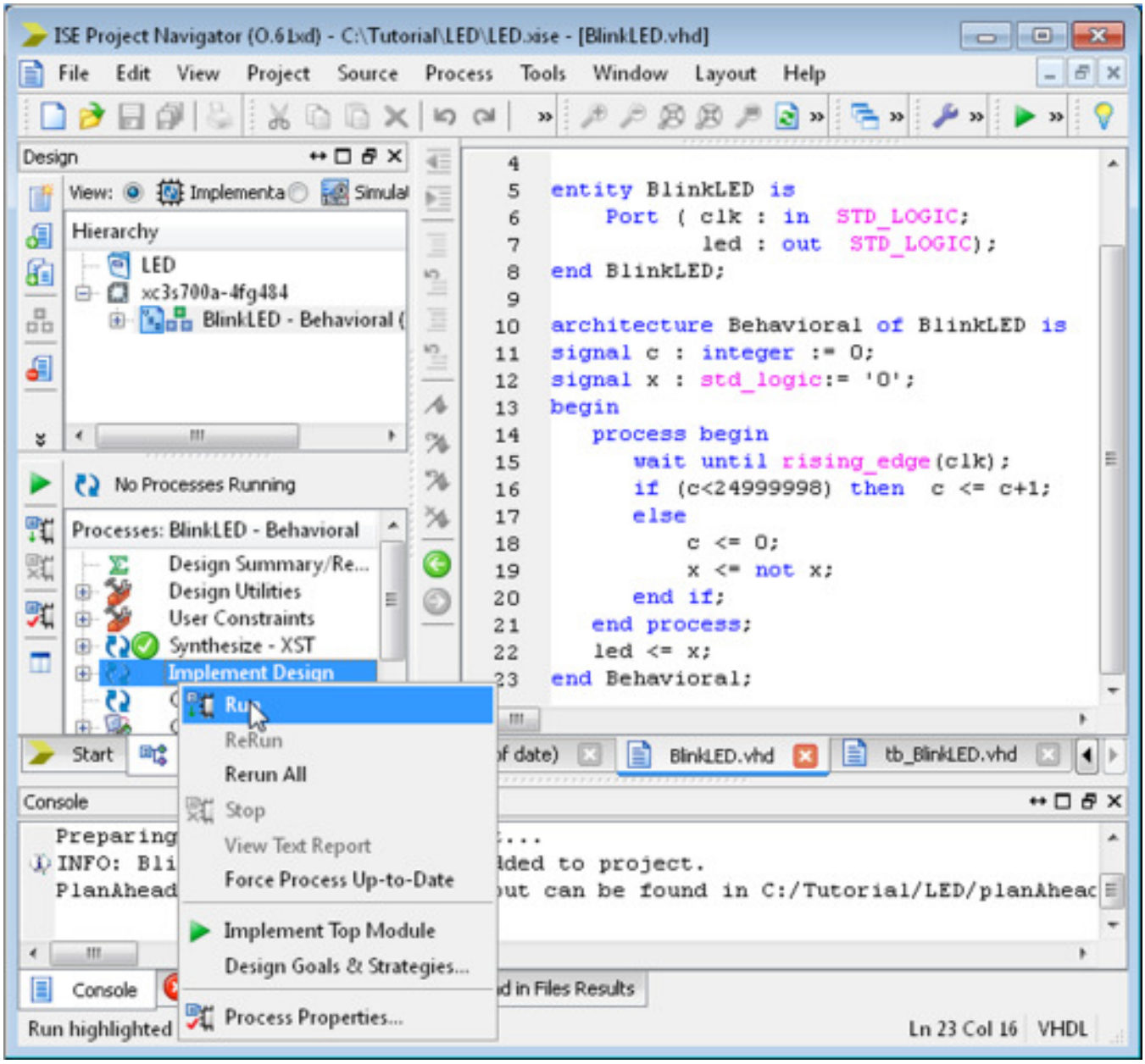
Nach der Zuordnung der Pins wird die UCF Datei gespeichert (entweder über den Speichern-Button oder indem PlanAhead geschlossen wird), und sieht dann etwa so aus:

BlinkLED.ucf

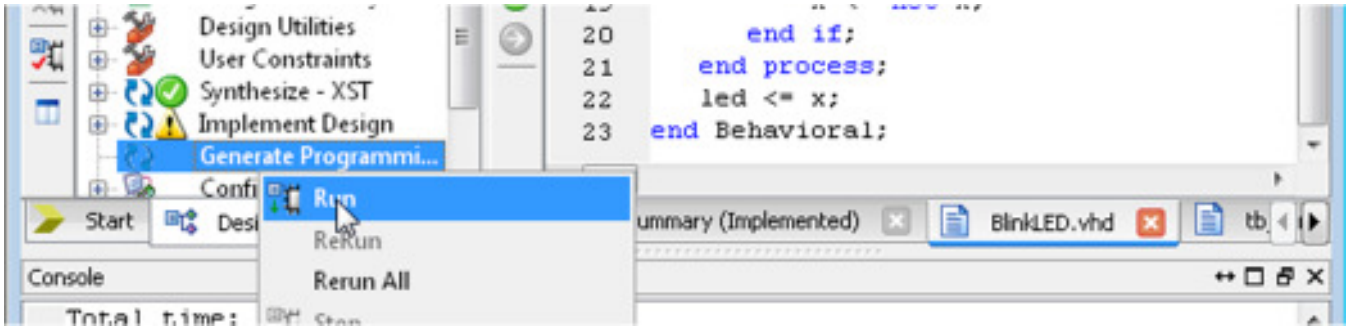
```
NET "clk" LOC = E12;
NET "led" LOC = R20;

NET "clk" IOSTANDARD = LVCMOS25;
NET "led" IOSTANDARD = LVCMOS33;
```

Wenn man sich die Datei ansieht, erkennt man leicht die vorgenommenen Pinzuordnungen. Automatisch hat PlanAhead hier auch noch IO-Standards zugewiesen (das wäre hier eigentlich noch nicht nötig, schadet aber auch nicht).

Auch hier nochmal der Hinweis, dass sich ein zweiter Blick in die UCF-Datei zur Kontrolle auf korrekt zugewiesene Signale durchaus positiv auf die Lebensdauer der beteiligten Bausteine auswirken kann: besser man findet den Fehler mit dem Auge als mit der Nase… ☺

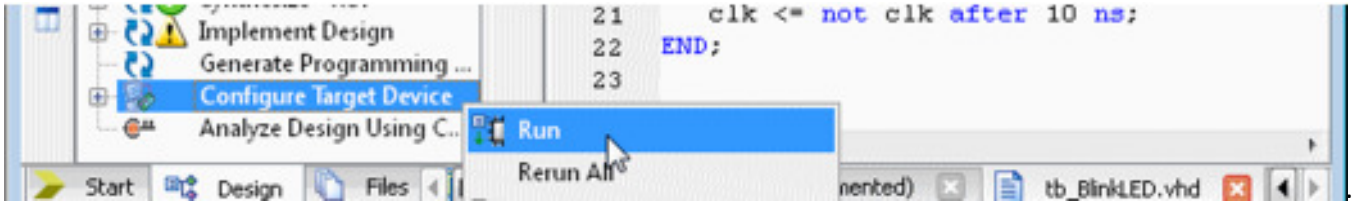


Jetzt kann das Design implementiert [Implement Design] und anschließend eine Bitstromdatei [Generate Programming File] erzeugt werden.



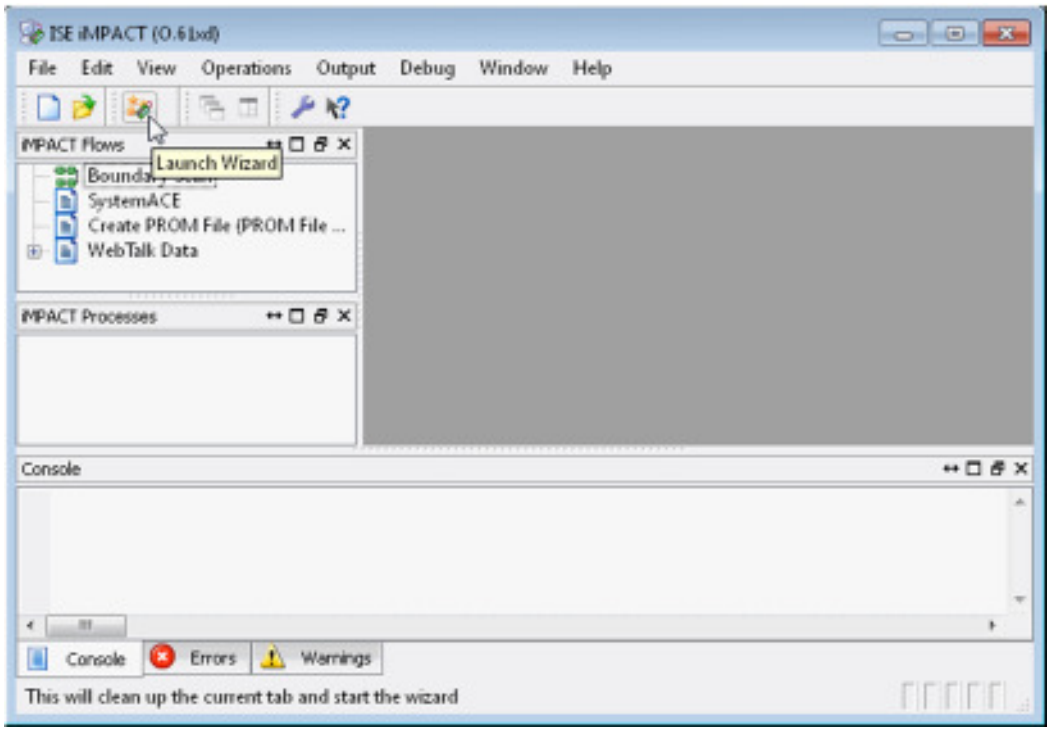
Man kann diese Schritte einzeln oder auch mit einem direkten Click auf das gewünschte Ergebnis ausführen.

Wenn alles ohne Fehler durchläuft, wird [Configure Target Device] aufgerufen



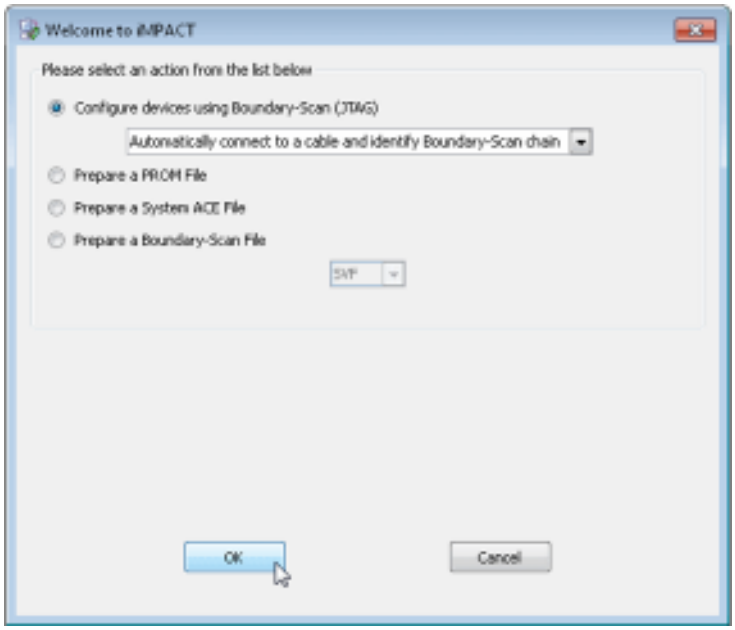
Damit wird die Programmiersoftware IMPACT gestartet.

Programmierung des FPGAs mit IMPACT

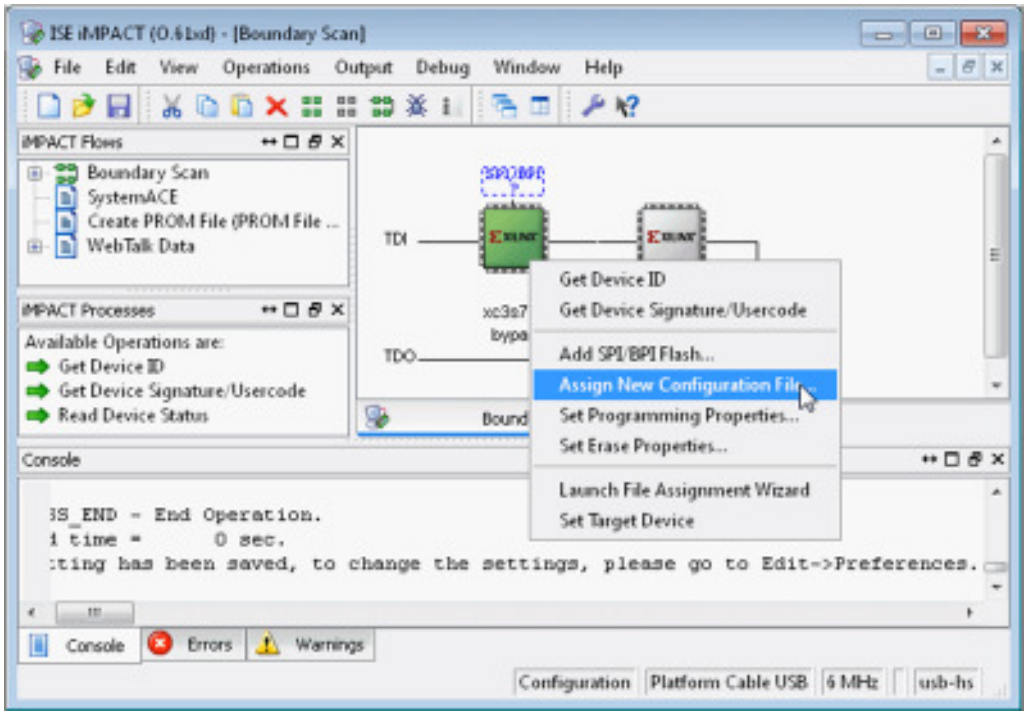


So sieht IMPACT nach dem Start aus: kein Projekt ist geöffnet, keine Verbindung zu einem Programmierkabel und Device ist hergestellt.

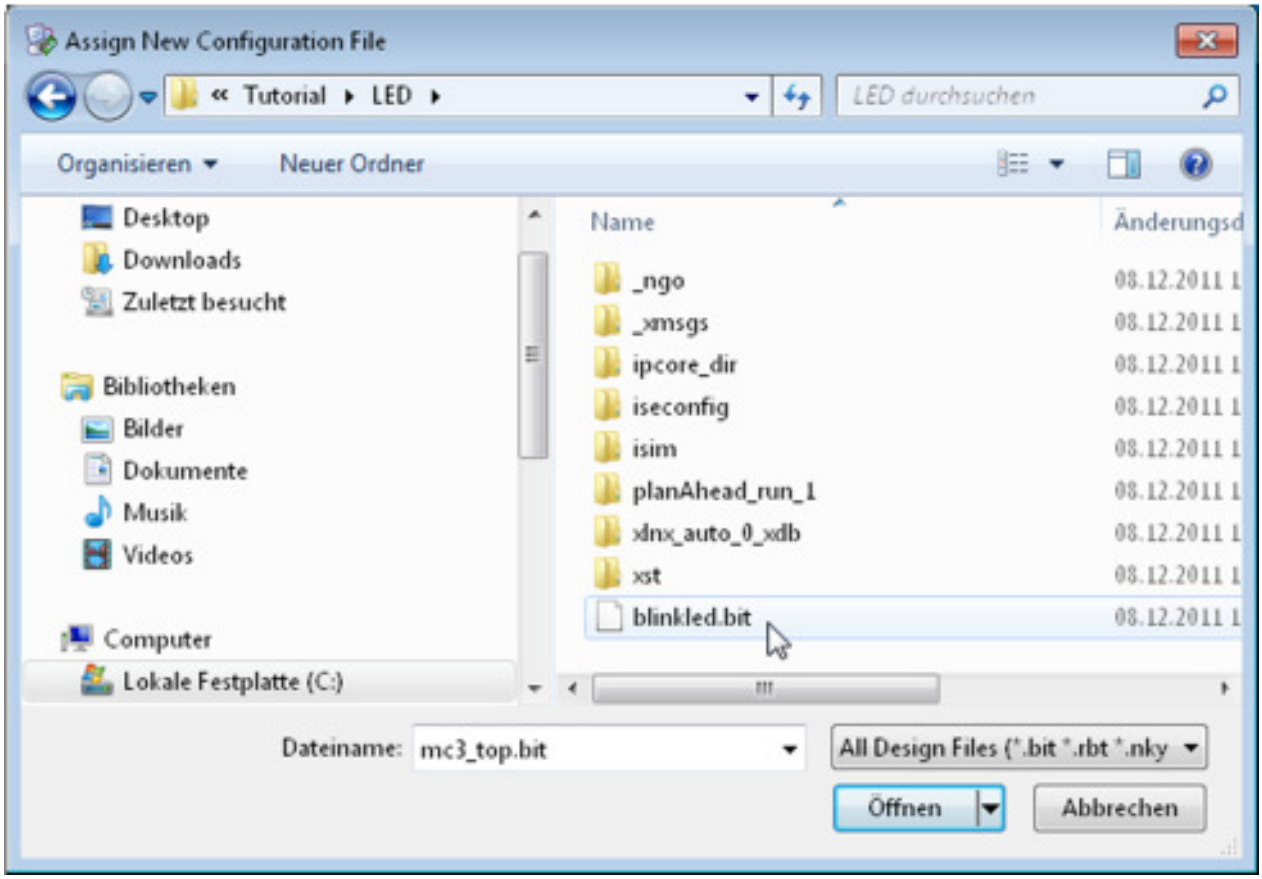
Dies ändern wir mit dem Button [Launch Wizard].



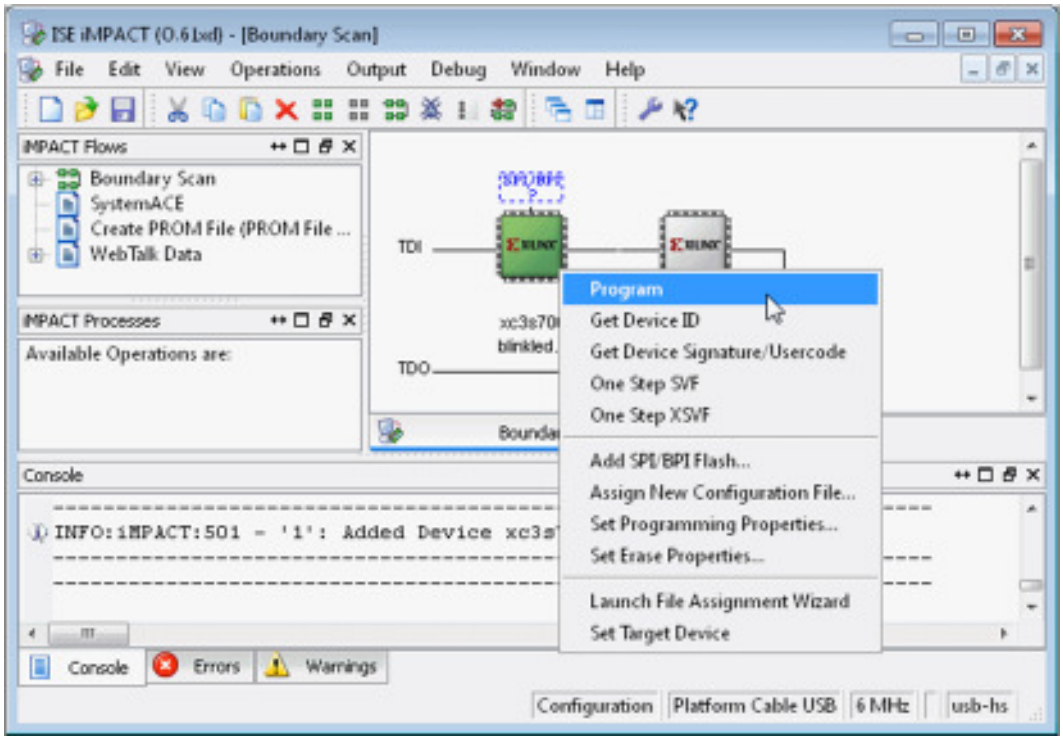
Mit viel Vertrauen geben wir IMPACT die Kontrolle ab und selektieren die automatische Auswahl (die eigentlich immer funktioniert).



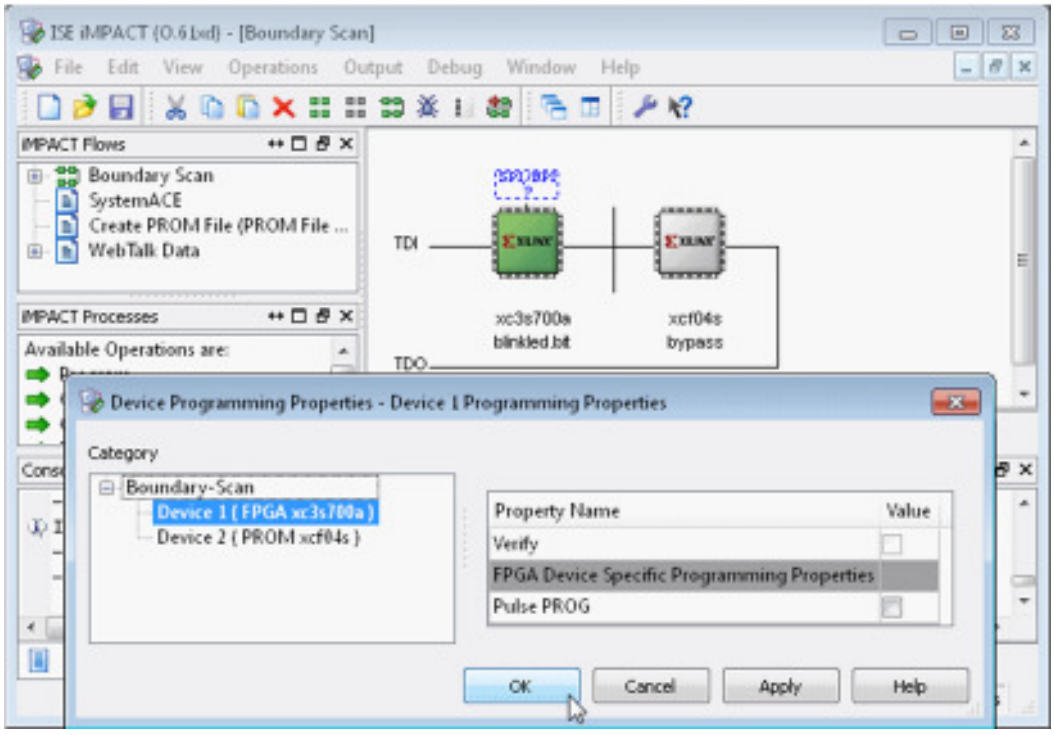
Es wird ein Programmierkabel erkannt und die JTAG Kette aufgebaut.
Mit einem Rechtsclick auf das Bausteinsymbol können wir dem FPGA seins Bitstromdatei zuordnen,



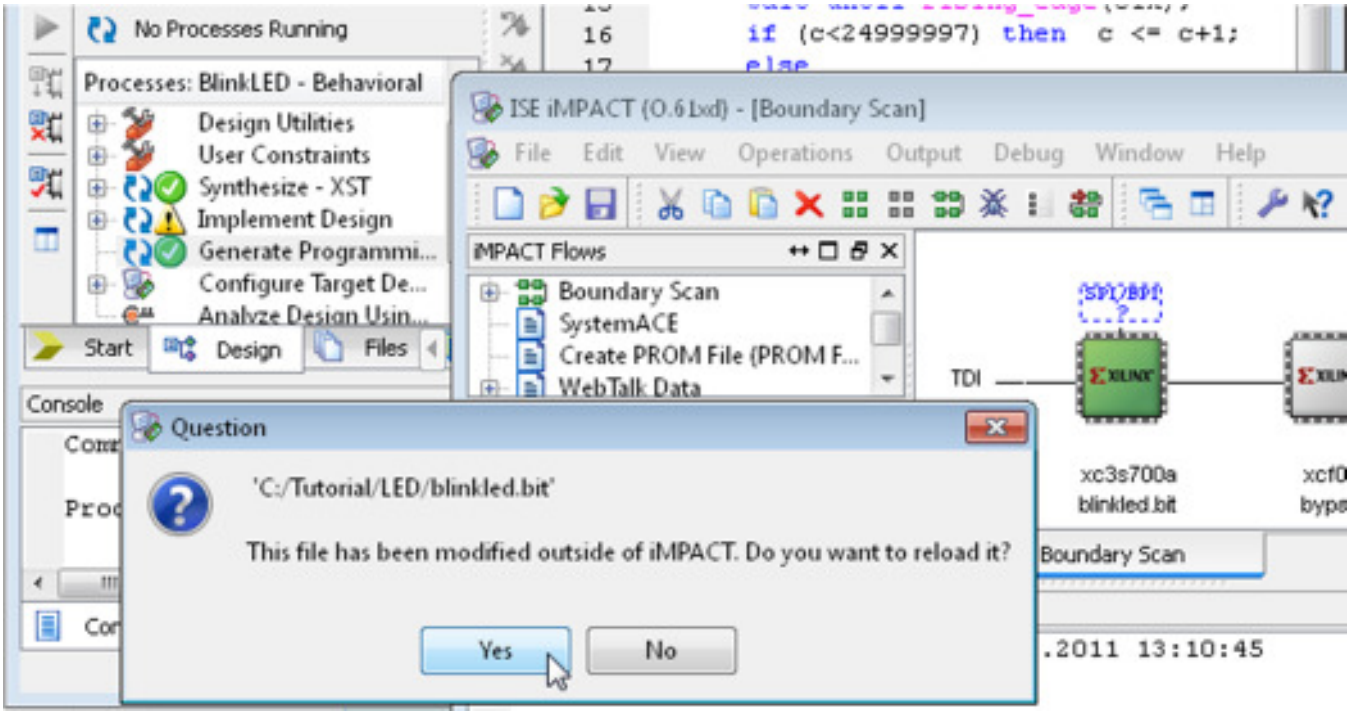
Die Auswahl ist gering: hier gibt es nur 1 Bitstromdatei „blinkled.bit“



Die Programmierung des FPGAs wird nun mit einem Rechtsclick auf das Bausteinsymbol und Auswahl von [Program] gestartet.



Das nun folgende Popup können wir ohne Änderungen mit [OK] bestätigen. Die Programmierung beginnt und wird hoffentlich mit „Programming Succeeded“ abgeschlossen. Die LED blinkt… ☺



Nach einer Änderung wählen wir in ISE nur noch den Menüpunkt [Generate Programming File], nicht [Configure Target Device], weil sonst IMPACT noch mal gestartet wird!

Danach können wir das FPGA mit in IMPACT einem Rechtsclick auf das Bausteinsymbol und Auswahl von [Program] erneut konfigurieren.
IMPACT erkennt, dass ich die Bitstromdatei geändert hat, meldet dies und fragt, ob die geänderte Datei verwendet werden soll. Natürlich bleibt hier nur die Antwort „Yes“.

Der VHDL Code für das Blinklicht

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

entity BlinkLED is
    Port ( clk : in  STD_LOGIC;
           led : out STD_LOGIC);
end BlinkLED;

architecture Behavioral of BlinkLED is
signal c : integer range 0 to 24999999 := 0;
signal x : std_logic := '0';
begin
   process begin
      wait until rising_edge(clk);
      if (c<24999999) then
          c <= c+1;
      else
          c <= 0;
          x <= not x;
      end if;
   end process;
   led <= x;
end Behavioral;
```

Der Inhalt der UCF Datei

```
NET "clk" LOC = E12;
NET "led" LOC = R20;

NET "clk" IOSTANDARD = LVCMOS25;
NET "led" IOSTANDARD = LVCMOS33;
```

Die Testbench für den Blinker

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.all;

ENTITY tb_BlinkLED IS
END tb_BlinkLED;

ARCHITECTURE behavior OF tb_BlinkLED IS
   COMPONENT BlinkLED
   PORT( clk : IN  std_logic;
         led : OUT std_logic );
   END COMPONENT;

   signal clk : std_logic := '0';
   signal led : std_logic;
BEGIN
   uut: BlinkLED PORT MAP (clk => clk, led => led);

   clk <= not clk after 10 ns;  -- 50MHz Takt
END;
```