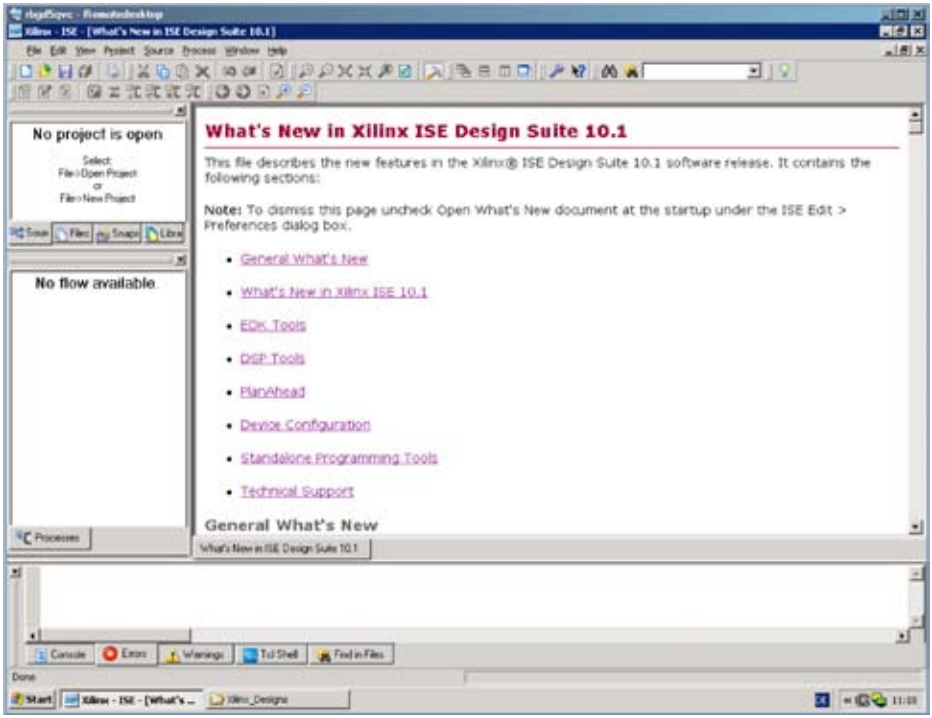
Kurzanleitung Xilinx ISE WebPack

Diese Kurzanleitung basiert auf der Version 10.1 mit ServicePack 3.
Die Software ist kostenlos auf der Xilinx Internetseite erhältlich (nur in englischer Sprache!).
Sie müssen sich auf „http://www.xilinx.com" registrieren, und können sich die Software dann kostenlos herunterladen. Um das ServicePack 3 zu installieren, laden Sie einfach die Updates für die ISE Vollversion herunter. Diese Updates gelten auch für das kostenlose WebPack. Das Tool bietet auch eine automatische Update Funktion am Ende der Installation, die aber erfahrungsgemäß deutlich länger dauert, als ein getrennter Download und manuelle Installation. Für die Installation benötigen Sie Admin-Rechte auf dem Rechner, ausgeführt werden kann die Software auch als normaler Benutzer.
Das WebPack beinhaltet alle Tools, die Sie zur Programmierung Ihres CPLD-Boards benötigen.

Starten des Tools und Anlegen eines neuen Projektes

Nach der Installation erscheint auf Ihrem Desktop ein Icon, mit dem Sie das Tool starten:

Beim ersten Start erscheint die Entwicklungsoberfläche:



Das Programm arbeitet projektorientiert.
Neben den Menüs und Symbolleisten im oberen Fensterbereich sehen Sie vier weitere Unterfenster.

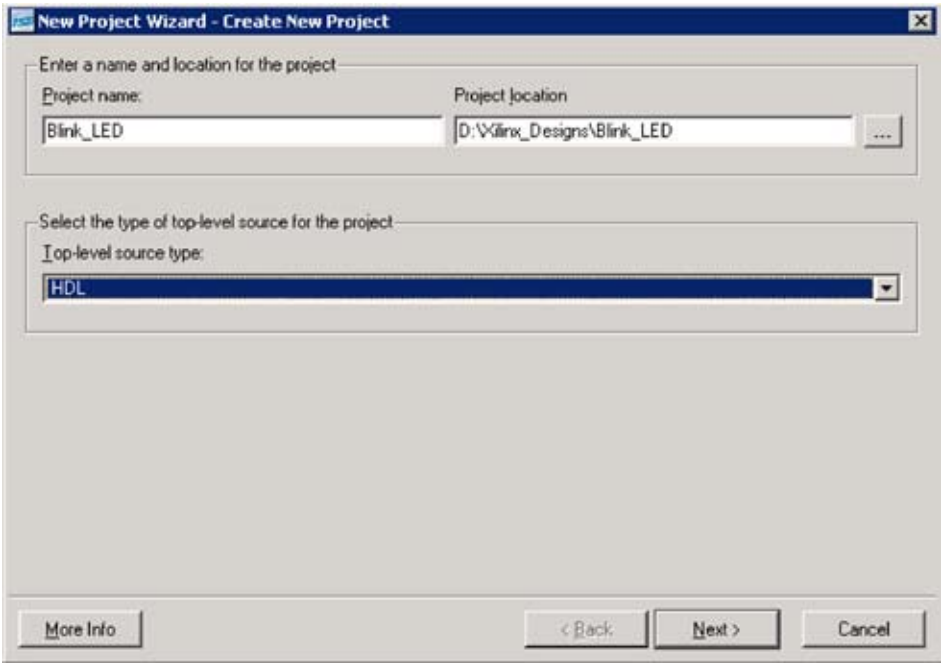
Auf der linken Seite im oberen Teil wird die Projekt-Struktur dargestellt. Hier finden Sie in hierarchischer Darstellung alle Komponenten, die an Ihrem Design beteiligt sind. Das sind in erster Linie die VHDL Beschreibungen, Testbenches und sog. Constraint-Files. Durch die Baumdarstellung sind die Abhängigkeiten erkennbar. Die Xilinx ISE erkennt automatisch Module, die evtl. in höheren Hierarchieebenen instantiiert werden und stellt sie entsprechend dar.

Im unteren Teil auf der linken Seite sehen Sie den Prozessablauf. Hier sehen Sie bei geöffnetem Projekt, welche Arbeitsschritte für das jeweilige selektierte Element zur Verfügung stehen.

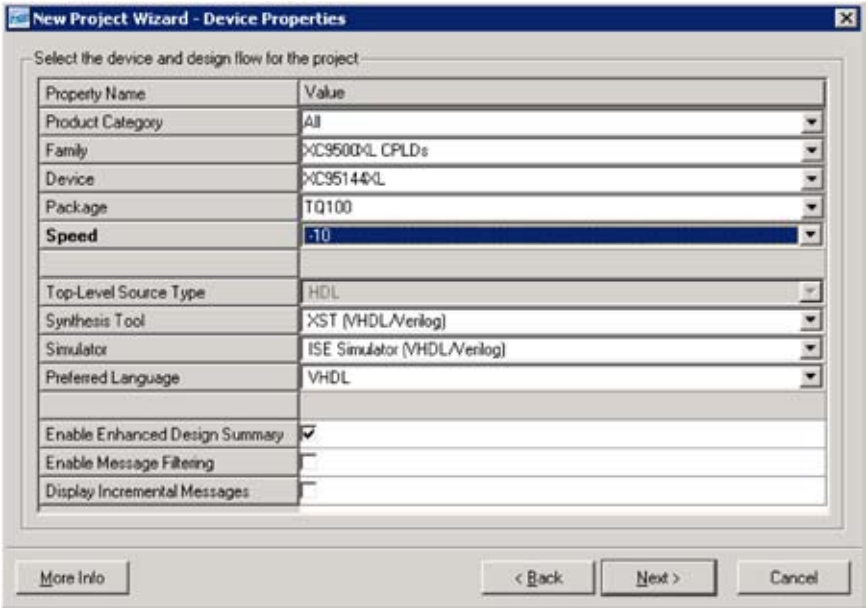
Das rechte Hauptfenster ist der eigentliche Arbeitsbereich. Hier können Sie im Texteditor die VHDL-Files bearbeiten, sich Hilfsinformationen (hierzu muss man leider online sein) oder Designstatistiken ansehen.

Das unterste Fenster ist der kommunikative Teil der ISE. Hier werden während der Designimplementierung Informationen angezeigt. In erster Linie interessant sind die „Warnings" und „Errors". Diese werden später mit zwei Links versehen. Diese zeigen auf die entsprechende Zeile im VHDL Code sowie auf die online Hilfefunktion von Xilinx. Oft findet man dort hilfreiche Tipps für die Fehlerbeseitigung.

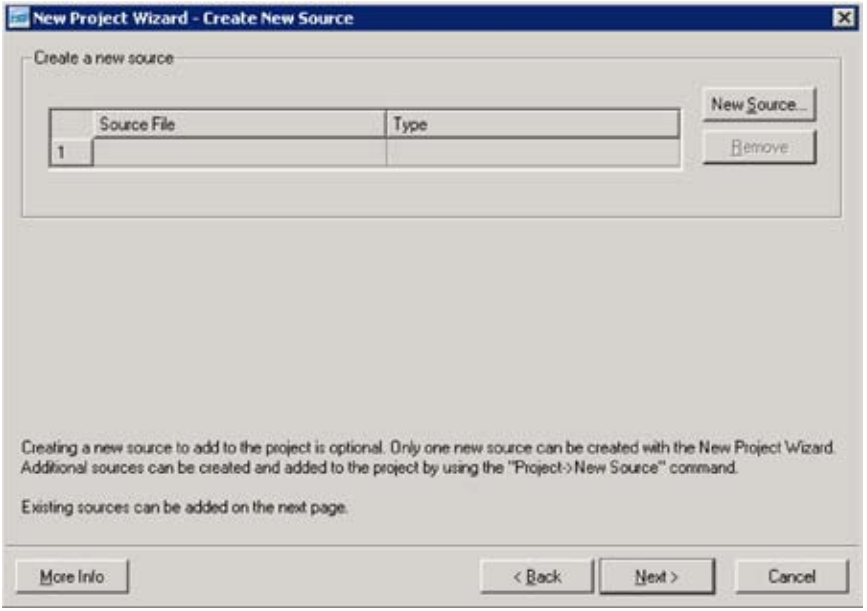
Ein neues Projekt legen Sie über das Menü „File", Unterpunkt „New Project..." an. Folgendes Dialogfenster erscheint:



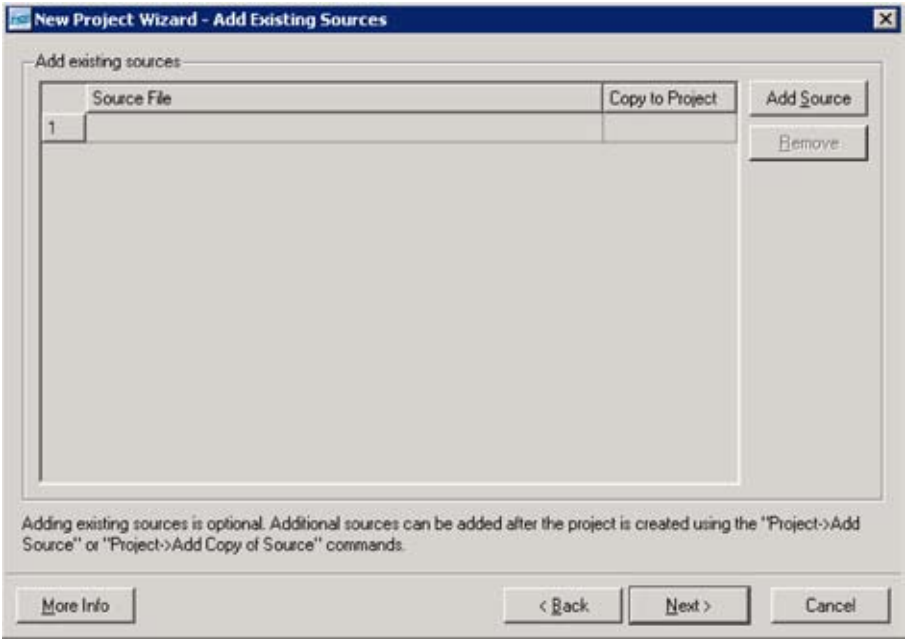
Legen Sie das Verzeichnis, in dem das Projekt angelegt werden soll fest. Für jedes Projekt wird dann automatisch ein eigenes Verzeichnis mit allen dazugehörigen Dateien angelegt (und das können eine ganze Menge werden). Im Anschluss wählen Sie noch einen Projektnamen. Hier in dieser Beschreibung beschränken wir uns auf die Designeingabe im „HDL", also Textformat. Theoretisch sind auch Eingaben über einen Schaltplan Editor oder Netzlisten möglich.



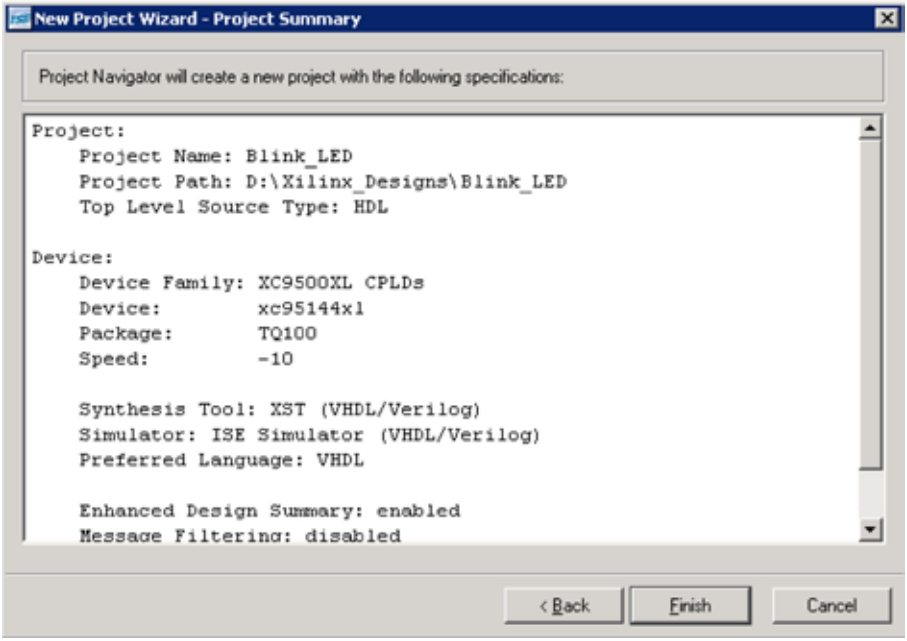
In diesem Fenster definieren Sie, für welchen Baustein Sie den Code schreiben, und welche Design-Sprache Sie verwenden wollen. In unserem Fall ist es ein Xilinx CPLD aus der „XC9500XL" Familie, genauer ein „XC95144XL" in einem „TQ100" Gehäuse. Der hier verwendete Baustein hat die Geschwindigkeitseinstufung „10".



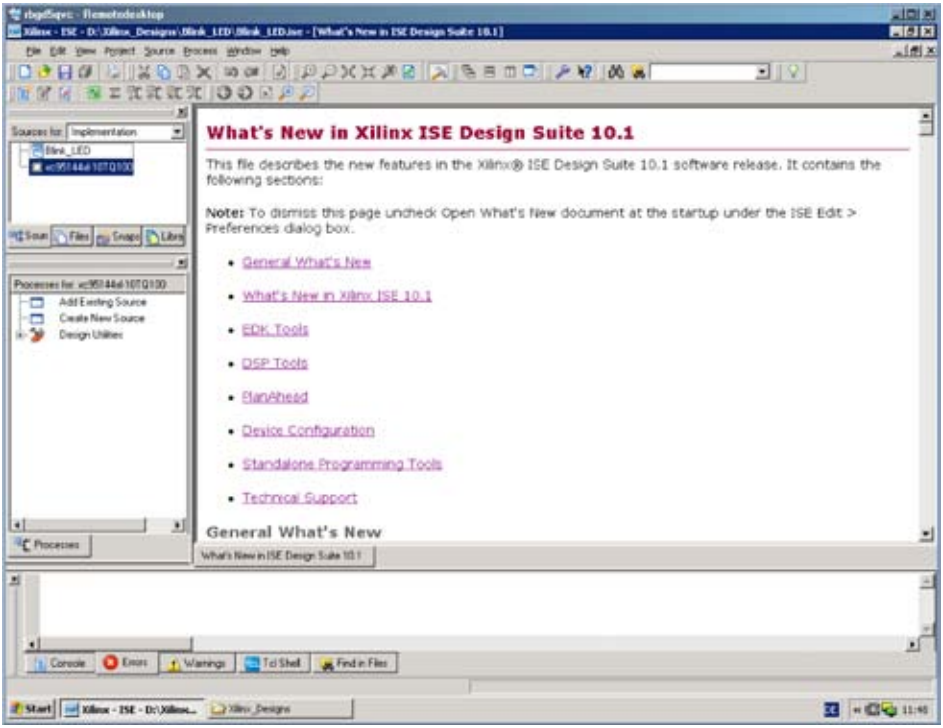
Im nächsten Arbeitsschritt hätten Sie die Möglichkeit, neue Files zu erstellen und gleich in das Projekt mit einzubinden. Vorerst übergehen wir diese Möglichkeit einfach mit „Next >".



Nun hätten Sie die Möglichkeit bereits existierende VHDL (oder andere) Design Dateien in das aktuelle Projekt einzufügen. Diesen Schritt übergehen wir mit „Next >" und wechseln so in das abschliessende Konfigurationsfenster, in welchem noch einmal eine Zusammenfassung der getätigten Einstellungen zu sehen ist. Wenn Sie mit den Angaben zufrieden sind, beenden Sie den Dialog mit „Finish" um in das Hauptfenster zurück zu gelangen.

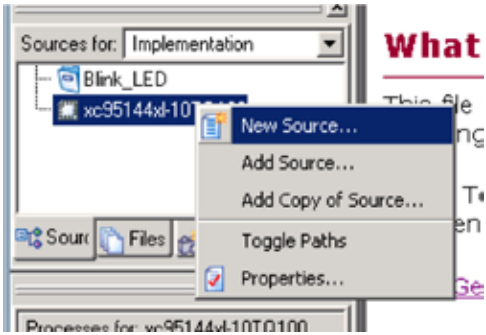


Das sollte nun wie folgt aussehen:

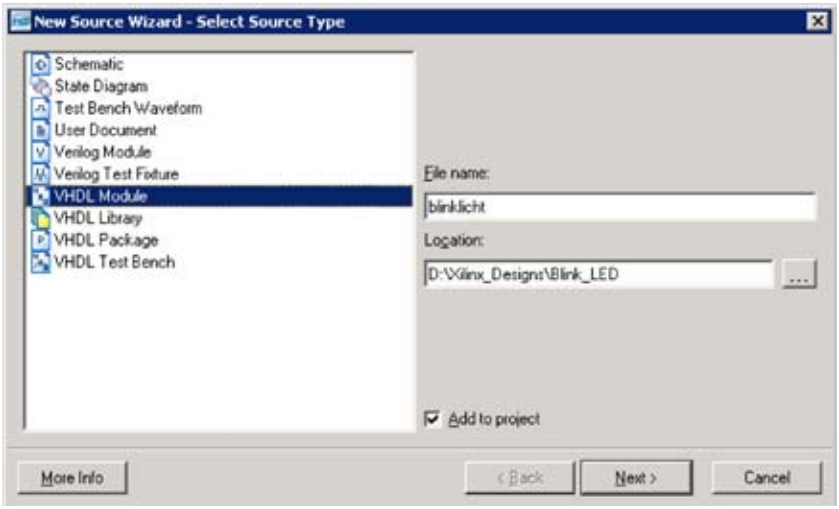


Erstellen und Einbinden eines VHDL-Files

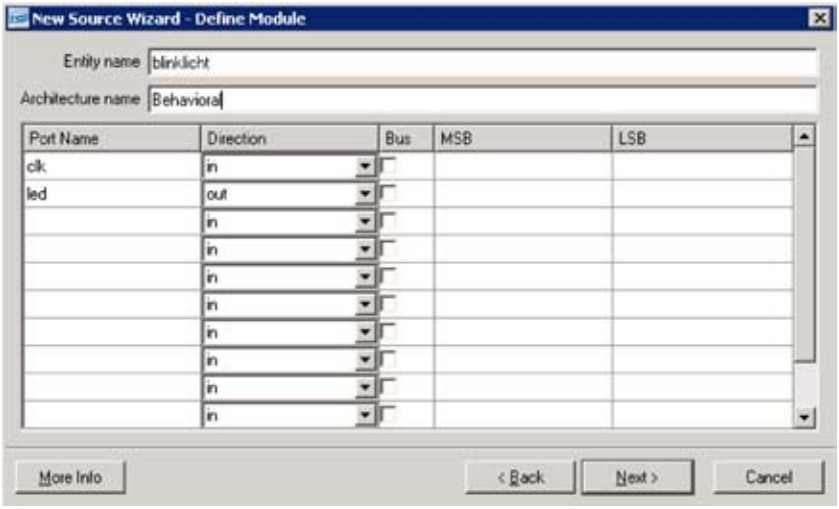
Um das Verhalten Ihres Bausteins zu beschreiben, müssen Sie nun ein VHDL (oder anderes HDL File) erzeugen. Klicken Sie dazu mit der Rechten Maustaste auf Ihren Baustein im Projektfenster und wählen Sie „New Source...".



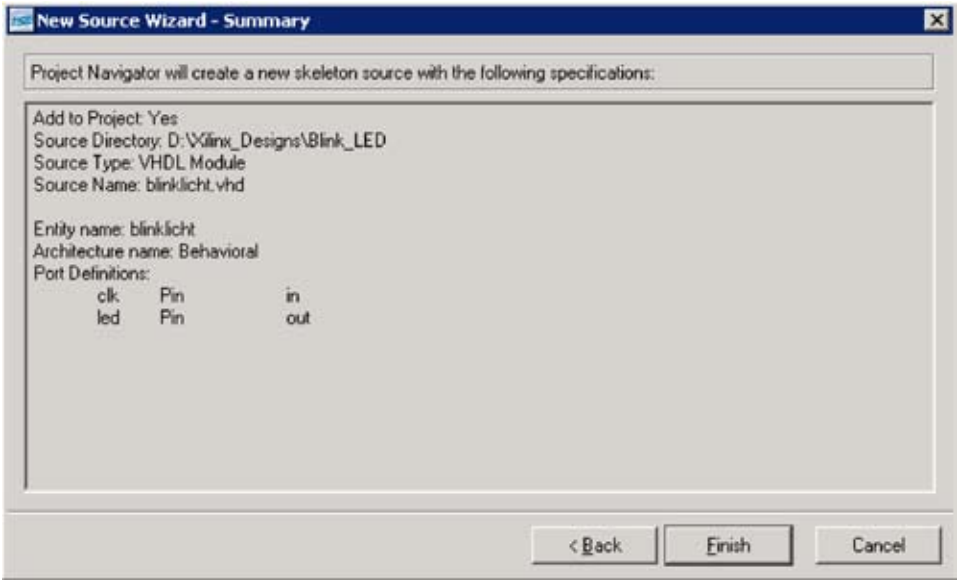
Es erscheint erneut ein Wizard, der Sie durch den Erstellungsprozess begleitet. Über dieses Hilfstool wird durch die ISE schon ein entsprechender VHDL-Rahmen erstellt, der Ihre Arbeit wesentlich erleichtert. Selbstverständlich können Sie den Wizard auch übergehen und alle Komponenten Ihres Designs selbst erzeugen.



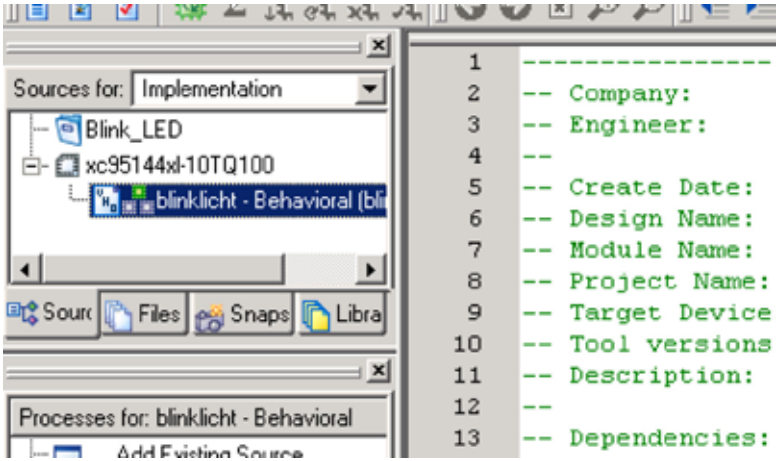
Im folgenden erstellen wir ein VHDL File mit dem Namen „blinklicht". Das Zielverzeichnis ist als Standard Ihr Projekt verzeichnis. In der linken Auswahlbox wählen wir „VHDL Module". Dies ist die entsprechende Auswahl für die Erzeugung von synthetisierbaren Code, der auf dem Baustein zur Ausführung kommen soll. Über die Schaltfläche „Next >" gelangen wir zur nächsten Seite.



In diesem Fenster definieren wir die Schnittstelle, die unser Design in die nach Außen hin haben soll. In unserem Bespiel verwenden wir einen Eingang „clk" und einen Ausgang „led". Selbstverständlich können im VHDL Code im weiteren neue ‚Ports' hinzugefügt werden, aber hier nimmt einem das Tool schon einige Arbeitsschritte ab, und erstellt automatisch eine passende Entity. Die Vorschläge für Entity- und Architecture Namen übernehmen wir für unser Beispiel. Mit dem Button „Next >" kommen wir zur abschliessenden Zusammenfassung, hier werden die gemachten Einstellungen nochmals angezeigt. Der Wizard wird mit „Finish" beendet, das File wird erzeugt und in das Projekt eingehängt.



Sie sollten das File jetzt im Projektnavigator unter Ihrem Baustein finden:



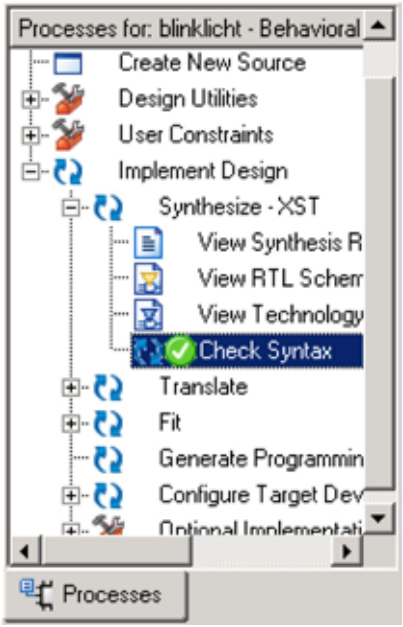
Im Hauptfenster wird die Datei anschließend zum Editieren geöffnet.

Erstellen eines VHDL-Files

Diese Kurzanleitung soll keinen VHDL-Kurs darstellen. Zum allgemeinen Verständnis sind aber ein paar Erläuterungen hilfreich. Ein VHDL Design besteht im wesentlichen aus zwei Teilen. Der erste Teil ist die sog. „entity". Hier werden die Schnittstellen definiert, über die das Design mit der Umwelt (intern zwischen Komponenten oder über physikalische Pins) kommuniziert. Es sind noch andere Definitionen möglich, die aber hier nicht weiter benötigt und erläutert werden.
Der zweite Teil ist die sog. „architecture". Das ist die eigentliche Beschreibung der Funktion. Eine Architecture kann wiederum mehrere Prozesse oder einfache logische Verknüpfungen enthalten.

Unser Beispiel hier soll ein einfaches Blinklicht werden. An einem Clock Eingang des CPLDs befindet sich ein 16 MHz Oszillator. Diese Frequenz ist für ein Blinklicht deutlich zu hoch. Die einfachste Möglichkeit eine Frequenz zu verkleinern, ist ein Binär-Zähler. Mit jeder Stufe des Zählers wird die Frequenz um den Faktor 2 geteilt. Um auf eine Blinkfrequenz von ca. 2Hz zu kommen, benötigen wir somit einen 22-stufigen Zähler. Das MSB (höchstwertige Bit) dieses Zählers können wir daher auf die LED ausgeben. Dadurch entsteht ein einfaches Blinklicht mit ca. 2Hz und einem Taktverhältnis von 1:1.

Das wohl wichtigste Tool während der Eingabe des VHDL Codes ist der Syntax Checker. Diesen finden Sie im Prozessfenster unter „Implement Design" -> „Synthese XST". Durch einen Doppelklick aktivieren Sie die Funktion.



Bei korrekt verwendeter Syntax wird das Element grün. Sollten Fehler auftreten, wird das Element rot gefärbt und im unteren Fenster die fehlerhaften Zeilen angezeigt.

Tipp: Die Zeile der Fehlermeldung ist nicht immer unbedingt der Fehler. Es kann z.B. passieren, dass Sie in der vorherigen Zeile den Strichpunkt vergessen haben und die Synthese dadurch die darauf folgende Zeile noch zum vorherigen Befehl zählt. Bei Fehlern, die nicht ersichtlich sind, also auch die vorherigen bzw. nachfolgenden Zeilen überprüfen.

Unser Beispielprogramm sieht nun wie folgt aus:

```
----------------------------------------------------------------------------
-- Company:        Pollin Electronic GmbH
-- Engineer:       Armin Schön
--
-- Create Date:    12/09/2008
-- Design Name:    2Hz Blinklicht
-- Module Name:    blinklicht - Behavioral
-- Project Name:   Blink_LED
-- Target Devices: XC95144XL
-- Tool versions:  10.1, SP3 (WebPack)
-- Description:    Einfaches Blinklicht
--
-- Dependencies:   ---
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

---- Uncomment the following library declaration if instantiating
---- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

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

architecture Behavioral of blinklicht is

  SIGNAL counter : STD_LOGIC_VECTOR (21 downto 0);

begin

  los_gehts: PROCESS (clk)
               BEGIN
                 IF RISING_EDGE(clk) THEN
                              counter <= counter + '1';
                            END IF;
               END PROCESS los_gehts;

  led <= counter(21);

end Behavioral;
```

Nach dem Header werden die benötigten Bibliotheken eingebunden. VHDL Funktionen basieren größtenteils auf Bibliotheken. Diese müssen in die entsprechenden VHDL-Files eingebunden werden, damit die Funktionen bekannt sind. Die hier aufgeführten Bibliotheken werden von der ISE automatisch eingeführt und reichen für die meisten Designs aus.

In der Entity sehen wir die Definition der beiden Signale „led" und „clk". Beide sind vom Typ „STD_LOGIC". Das ist die meist verwendete Logik-Variante in VHDL. Sie beherrscht neben '1' und '0' auch verschiedene andere Level wie 'X', 'Z' oder weitere. Der „Port" „clk" ist als Eingang und „led" als Ausgang definiert.

An letzter Stelle im Code folgt nun die Architecture. Hier wird ein Signal definiert, dass unseren internen Zähler darstellt. Innerhalb der Architecture findet sich ein Prozess (hier mit dem Namen „los_gehts". Dieser Prozess wird aktiviert, wenn sich das Signal „clk" ändert. Wird eine steigende Flanke festgestellt , so erhöht sich der Wert unseres Signals „counter" um eins. Zuletzt wird nun das MSB unseres Zählers über den Port „led" ausgegeben.

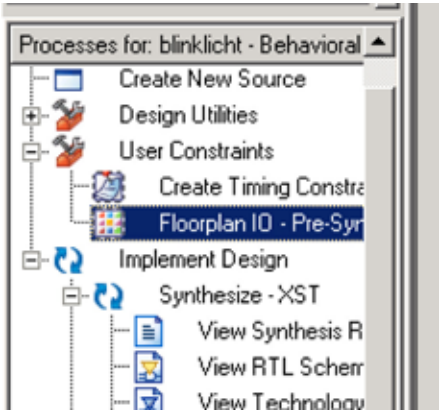
Im Unterschied zu einem Prozessor / µController laufen in einem CPLD (oder FPGA) alle Konstrukte parallel ab. Hinzufügen von weiteren VHDL Modulen oder Prozessen wirken sich also nicht direkt auf die bereits implementierten aus. So lange Ressourcen (für Logik Implementierung und Verdrahtung) zur Verfügung stehen, ist die Geschwindigkeit nur von zweitrangiger Bedeutung. Nicht verschwiegen werden darf, dass sich bei unterschiedlichem Füllgrad des Bausteins auch die Lage der Komponenten zueinander und die Verdrahtung ändert. Es kann also durchaus Änderungen in der Signallaufzeit und damit der maximalen Betriebsfrequenz ergeben. Bei CPLDs ist dies allerdings nicht kritisch.

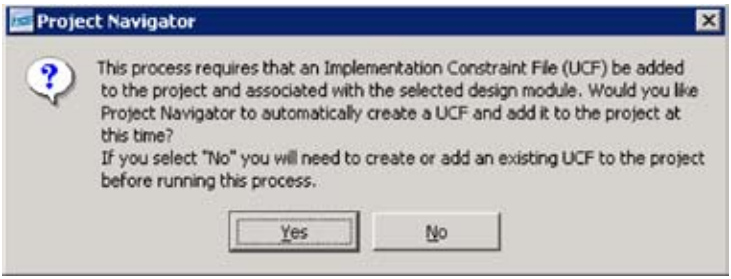
Dies können erfahrene Designer allerdings über entsprechende Vorgaben optimieren.

Implementierung

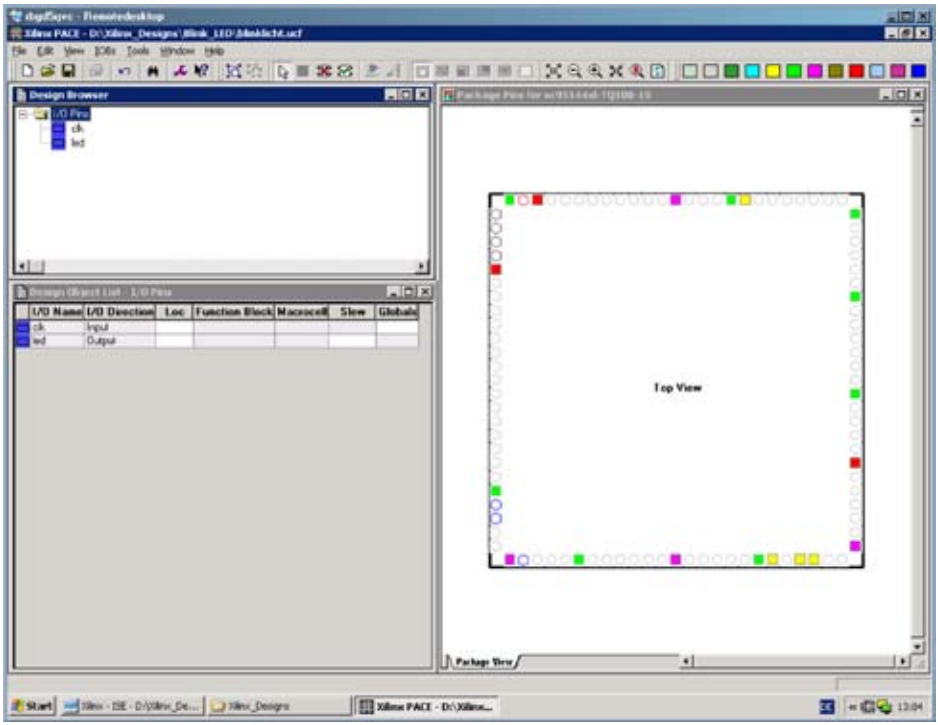
Bislang sind nur die Ports definiert. Die LED sowie der Oszillator hängen allerdings an bestimmten Anschlüssen des CPLDs. Daher ist es notwendig, den definierten Ports entsprechende Pins am Gehäuse zuzuweisen. Dies geschieht in der ISE mit einem eigenen Tool und läuft wie folgt ab:

Im Prozessfenster unter „User Constraints" findet sich das Tool „Floorplan IO".





Dieses Fenster erscheint beim ersten Start des Tools. Bestätigen Sie mit „Yes" um eine neue UCF Datei anzulegen und diese wie folgende Ansicht im Design-Browser zu öffnen.



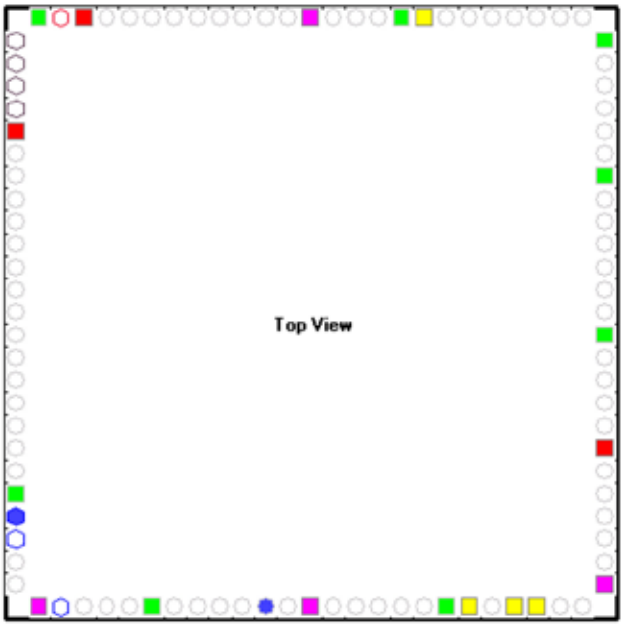
Im Desing Browser finden Sie die im Design definierten Ports. Auf der rechten Seite, sehen Sie den Baustein mit seinen 100 Pins in geometrischer Anordnung.
Farbig markierte Pins haben Sonderfunktionen und stehen nicht als IO zur Verfügung. Die drei blau umrandeten 6-eckigen Pins stellen die Clock-Eingänge dar. Diese sollten auf jeden Fall für generelle Takte verwendet werden, da sie intern anders verschaltet sind und allen Registern und FlipFlops ohne besonderen Verdrahtungsaufwand zur Verfügung stehen.
Die Ports können nun mit Drag-And-Drop auf die Pins gezogen werden oder was meist zielsicherer ist, in die Tabelle unter der Spalte „Loc" eingetragen werden. Die Pin Nummern beginnen mit ‚p', z.B. ‚p33'.

In unserem Beispiel sieht die ausgefüllte Tabelle wie folgt aus:

Design Object List - I/O Pins

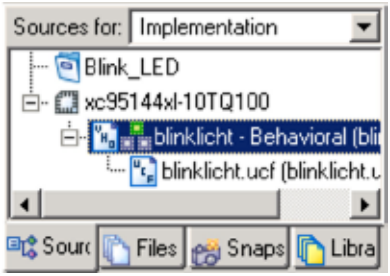
	I/O Name	I/O Direction	Loc	Function Block	Macrocell	Slew	Globals
	clk	Input	p22	1	17		CLK
	led	Output	p36	5	5	SLOW	

Neben der Pin-Nummer muss für Taktsignale noch das Feld „Globals" auf „CLK" geändert werden, um die entsprechende Taktverteilung im Baustein zu verwenden. Für Ausgänge kann noch die Anstiegsflanke ausgewählt werden. Für LEDs reicht die Einstellung „SLOW" aus. In der grafischen Darstellung des Bausteins werden belegte Pins blau markiert. Folgende graphische Darstellung entspricht unserem Beispielprogramm:

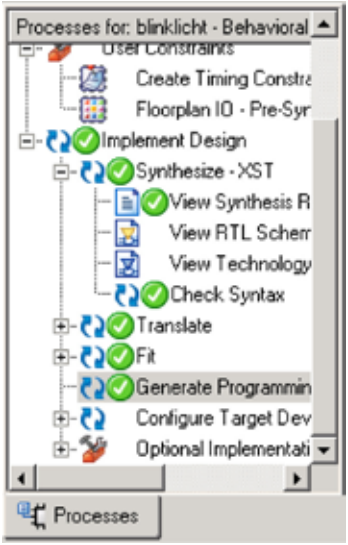


Speichern Sie diese Konfiguration nun ab, achten Sie bei der freien Version der ISE darauf, dass „XST Default: < >" aktiviert ist.

Damit ist die Pin-Zuweisung abgeschlossen und das UCF File erscheint unter dem entsprechenden VHDL File im Projektfenster:



Nun kann das ganze Design synthetisiert werden und das Programmierfile für den Baustein generiert werden.
Im einfachsten Fall wird dies durch einen Doppelklick auf „Generate Programming File" im Prozessfenster gestartet. Alle vorher notwendigen Schritte werden durch die ISE automatisch durchgeführt.



Wenn alles erfolgreich war, werden die erfolgreich durchgeführten Schritte grün markiert.

Der prinzipielle Ablauf erfolgt in drei Schritten:

1. Synthese:
Hier wird der VHDL Code in logische Funktionen übersetzt, die sich in der jeweiligen Hardware umsetzen lassen.

2. Umsetzung:
Die logischen Elemente aus der Synthese werden dedizierten Komponenten innerhalb des CPLDs zugewiesen. Über verschiedene Schalt-Matrizen werden die elektrischen Verbindungen zwischen den Komponenten hergestellt.

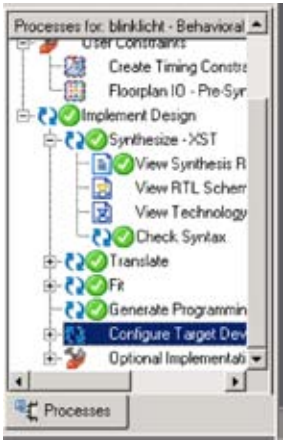
3. Programmierfile erzeugen:
Die erstellte Umsetzung wird nun in einen Bitstream verpackt, der die entsprechenden Register / Schaltstellen im CPLD konfiguriert. Das CPLDs wird durch nicht flüchtige Speicherzellen konfiguriert, d.h. auch nach Entfernen der Betriebsspannung bleibt die letzte Programmierung erhalten.

Diese Schritte können auch einzeln durchgeführt werden. Nach Änderung am Design müssen die betreffenden Schritte erneut wiederholt werden.

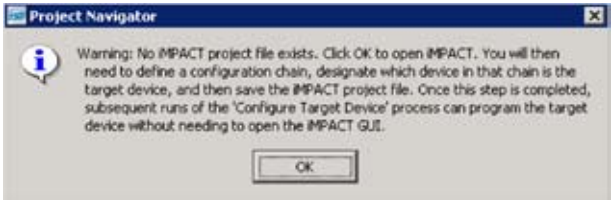
Programmieren des Bausteins

Das CPLD wird über eine „JTAG" Schnittstelle konfiguriert. Dies kann über entsprechende Hardware Programmer erfolgen oder über den auf dem CPLD Board implementierten Parallel zu JTAG Adapter. Das Board wird über ein Parallelkabel mit dem Parallelport des PCs verbunden. Die verschiedenen Treiber, welche in der ISE implementiert sind, erkennen diese Programmierschnittstelle automatisch. Stellen Sie sicher, dass das Board mit dem PC verbunden ist und betriebsbereit.

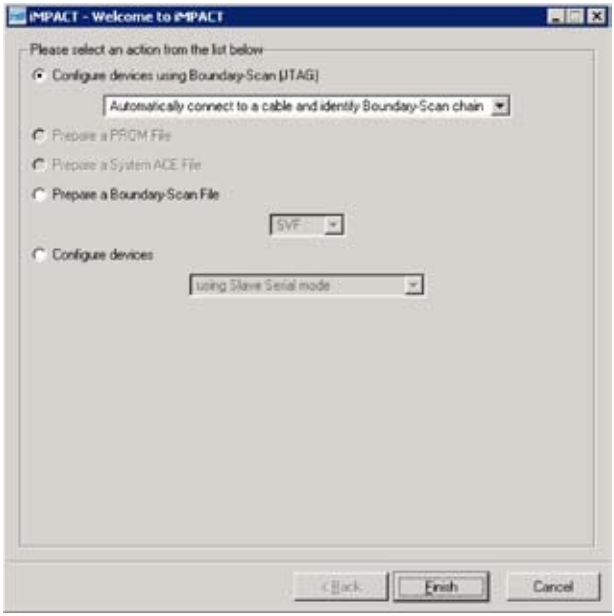
Starten Sie nun im Prozess-Fenster das Programmiertool durch einen Doppelklick:



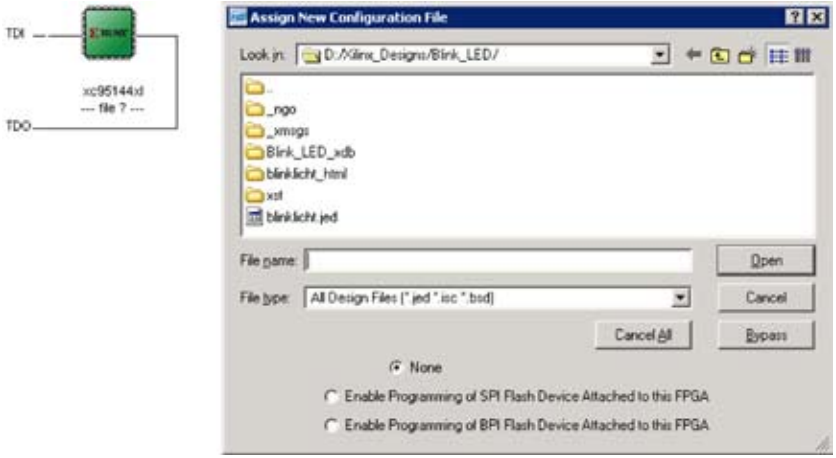
Beim ersten Start erscheint ein Hinweisfenster, dass noch kein iMPACT Project File existiert. Bestätigen Sie dieses Fenster einfach mit „OK".



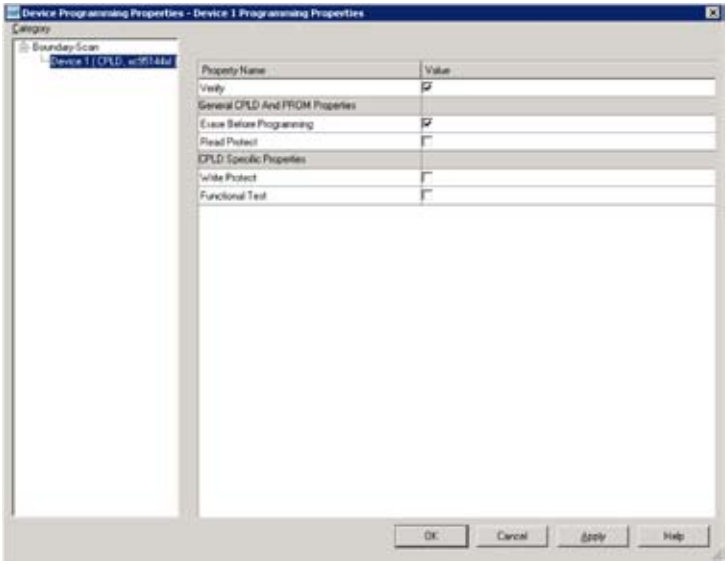
Im Folgenden erscheint ein Fenster, das Sie mit „Finish" schließen können. Stellen Sie nur sicher, dass die automatische Identifizierung der JTAG-Kette und der Kabel, wie auf dem nachfolgenden Bild zu erkennen, aktiviert ist.



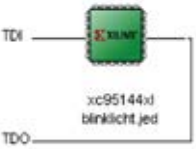
Der Baustein sollte richtig erkannt werden und auf einem neuen Tab in der Kette (hier nur ein Baustein) erscheinen. Sie können dem Baustein nun das entsprechende Bit-File (blinklicht.jed) zuordnen:



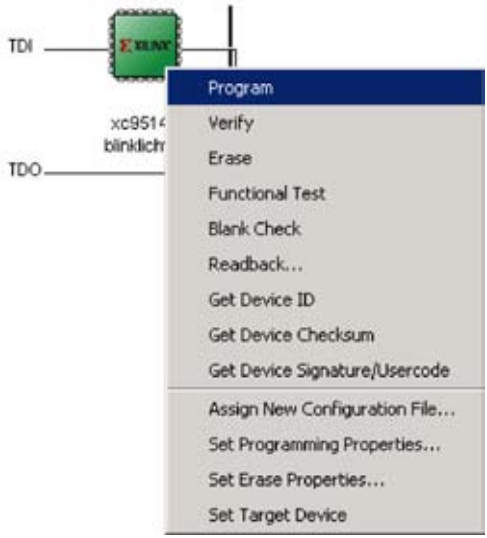
Im darauf folgenden Fenster können Sie noch genauer definieren, was beim Programmieren geschehen soll. Die Standard-Einstellungen können für unser Beispielprogramm übernommen werden.



Das entsprechende File wurde dem Baustein zugewiesen:



Mit der rechten Maustaste können Sie nun die entsprechenden Funktionen auswählen.



Ein Fortschrittsfenster zeigt an, wie weit die Programmierung fortgeschritten ist. Beim vorliegenden Baustein dauert sie in etwa 6 Sekunden.

Und schon blinkt eine der LEDs ... meine zumindest.

Im Hauptfenster gibt es nach der Implementierung noch ein paar Hinweise über die Auslastung des verwendeten Bausteines.

Design Name	blinklicht
Fitting Status	Successful
Software Version	K.39
Device Used	XC95144XL-10-TQ100
Date	12-9-2008, 1:41PM

RESOURCES SUMMARY

Macrocells Used	Pterms Used	Registers Used	Pins Used	Function Block Inputs Used
22/144 (16%)	21/720 (3%)	22/144 (16%)	2/81 (3%)	43/432 (10 %)

PIN RESOURCES

Signal Type	Requiered	Mapped	Pin Type	Used	Total
Input	0	0	I/O	1	74
Output	1	1	CCK/IO	1	3

Viel Spass bei den eigenen Designs ... vieles kann man anders machen ... vieles kann man mehr machen, aber ich hoffe, diese Anleitung hat ein wenig bei den ersten Schritten mit der Xilinx Entwicklungsumgebung geholfen.