BERGISCHE UNIVERSITÄT WUPPERTAL
FACHBEREICH C - PHYSIK
ELEKTRONIKPRAKTIKUM

Version 4.2013, TEX: 10. April 2013

Versuch EP10 Digitalelektronik

Teil 2: Programmierbare Logikbausteine (CPLD und FPGA)

I. Zielsetzung des Versuches

Im vorherigen Versuch haben Sie einfache logische Grundschaltungen und ihre Realisierung mit einfachen Logikbausteinen (Digital-ICs) kennenlernt.

Im heutigen Versuch soll es um programmierbare Logikbausteine gehen (CPLD, FPGA). Sie sind heute sehr preisgünstig und erlauben die einfache Realisierung auch komplexer Steuerungsaufgaben, wo früher sehr viele einfache Logikbausteine verschaltet wurden.

Die Programmierung solcher Logikbausteine (die Definition des logischen Verhaltens) kann auf zwei Arten geschehen:

- Bei sehr einfachen Problemen kann man zunächst einen Schaltplan zeichnen, in dem einfache Gatter und Speicher (Flipflops) miteinander verbunden werden. Daraus wird dann automatisch eine Programmdatei für den CPLD generiert.
- Bei allen anderen Problemen ist es besser, eine sogenannte Hardware-Beschreibungssprache zu verwenden wie VHDL oder Verilog. Wenn Sie Erfahrungen mit der Programmiersprache C haben, wird Ihnen in Verilog vieles bekannt vorkommen. Auch hier wird aus der Verilog-Datei automatisch eine Programmdatei für den CPLD generiert.

Wir werden in unserem Versuch sowohl den Schaltplanentwurf als auch das Programmieren über Verilog kennenlernen.

II. Vorkenntnisse

Grundlagen der Digitaltechnik (Versuch EP10)

Eigenschaften von Logikschaltungen (CPLD, FPGA).

Informationen zum Umgang mit der Programmierumgebung finden Sie in dieser Versuchsanleitung.

III. Theorie zum Versuch

1. Logikbausteine

1.1. Einfache und programmiere Logikbausteine

Die klassischen Logikbausteine, die es seit etwa 1960 Jahren gibt, beherrschen immer nur bestimmte einfache Funktionen. Der Aufbau komplexerer Schaltungen (und sei es nur eine 6stellige Digitaluhr) führt schnell zu einer großen Ansammlung solcher Bausteine, die viel Platz braucht und letztlich hohe Kosten verursacht.

Ein erhebliches Problem tritt auf, wenn eine solche Schaltung modifiziert werden soll. Da alle Details der Schaltung durch die festen Leitungsverbindungen definiert sind, müssen diese entweder mit Drahtbrücken geändert werden oder die gesamte Schaltung wird neu aufgebaut.

Aus diesem Grund wurden um etwa 1980 Bausteine entwickelt, deren genaues Verhalten durch eine Programmierung festgelegt werden kann und muß. Ändern sich im im Laufe einer Schaltungsentwicklung bestimmte Anforderungen, so können diese leicht durch Umprogrammierung umgesetzt werden.

Grundsätzlich ist zu unterscheiden zwischen:

- der progammierbaren Logik (CPLD, FPGA), mit der wir uns in diesem Versuch befassen, und
- den Mikrokontrollern und Mikroprozessoren, die im nächsten Versuch behandelt werden.

1.2. Programmierbare Logik

Ein programmierbarer Logikbaustein (CPLD = Complex Programmable Logic Device, FPGA = Field Programmable Gate Array) ist vereinfacht betrachtet eine Ansammlung von vielen Gattern und Flipflops. Im unprogrammierten Baustein haben sie jedoch keine Verbindung zueinander. Erst durch die Programmierung wird festgelegt, welche Gatter und Flipflops wie miteinander und mit den Anschlüssen des Bausteins verbunden werden.

Dabei kann das Programm auf zwei Arten entwickelt werden:

- Als Schaltplan: Die Programmierumgebung auf dem PC stellt die Symbole für Gatter und Flipflops bereit, die dann miteinander verbunden werden.
- Als Programmiersprache: Das Verhalten der Schaltung wird durch einen Text (ein Programm) mit logischen Gleichungen usw. beschrieben. Solche Sprachen (Hardware Definition Languages, HDLs) wie VHDL oder Verilog haben z.T. große Ähnlichkeit mit Programmiersprachen für Computer wie z.B. C.

Ist das Programm geschrieben, oder der Schaltplan gezeichnet, kann das Verhalten der Schaltung am PC simuliert werden.

Schließlich wird eine Programmdatei erzeugt (Compilierung/Synthese) und vom PC in den Logikbaustein übertragen. Das Programm bleibt im Baustein auch nach dem Abschalten der Versorgungsspannung erhalten (Flash-Speicher, wie z.B. bei einer Speicherkarte oder einem USB-Stick).

2. Unterschiede zwischen programmierbarer Logik und Mikrocontrollern

Programmierbare Logik arbeitet „in Echtzeit“. Die Eingangssignale werden sofort von den Gattern und Flipflops in der gewünschten Weise verknüpft und an die Ausgänge gegeben. Es können dabei sehr viele Bits gleichzeitig verarbeitet werden.

Daher sind diese Bausteine sehr schnell und können oft Signalfrequenzen von mehreren 100 MHz verarbeiten.

Komplexe Rechenfunktionen sind aber — vor allem bei den einfacheren Bausteinen — nicht oder nur mühsam zu realisieren.

Mikrocontroller arbeiten nach dem Prinzip eines Computers. Sie arbeiten ein bestimmtes Programm *nacheinander* (sozusagen Zeile für Zeile) ab, nehmen Eingangssignale entgegen, verknüpfen und vergleichen sie, geben sie danach als Ausgangssignale aus oder speichern etwas ab.

Daher sind diese Bausteine relativ langsam und können nur Signalfrequenzen von einigen MHz verarbeiten. Das liegt auch an den geringen Taktfrequenzen, die auch aus Gründen geringer Stromaufnahme oft nur bei höchstens einigen 10 MHz liegen.

Komplexe Rechenfunktionen sind aber leicht zu realisieren.

Das Programm kann aus elementaren Programmierbefehlen (d.h. im Assemblercode) aufgebaut werden. Üblich weil wesentlich einfacher und bequemer ist aber die Verwendung einer Hochsprache wie BASIC oder C.

3. Beispiele und technische Details

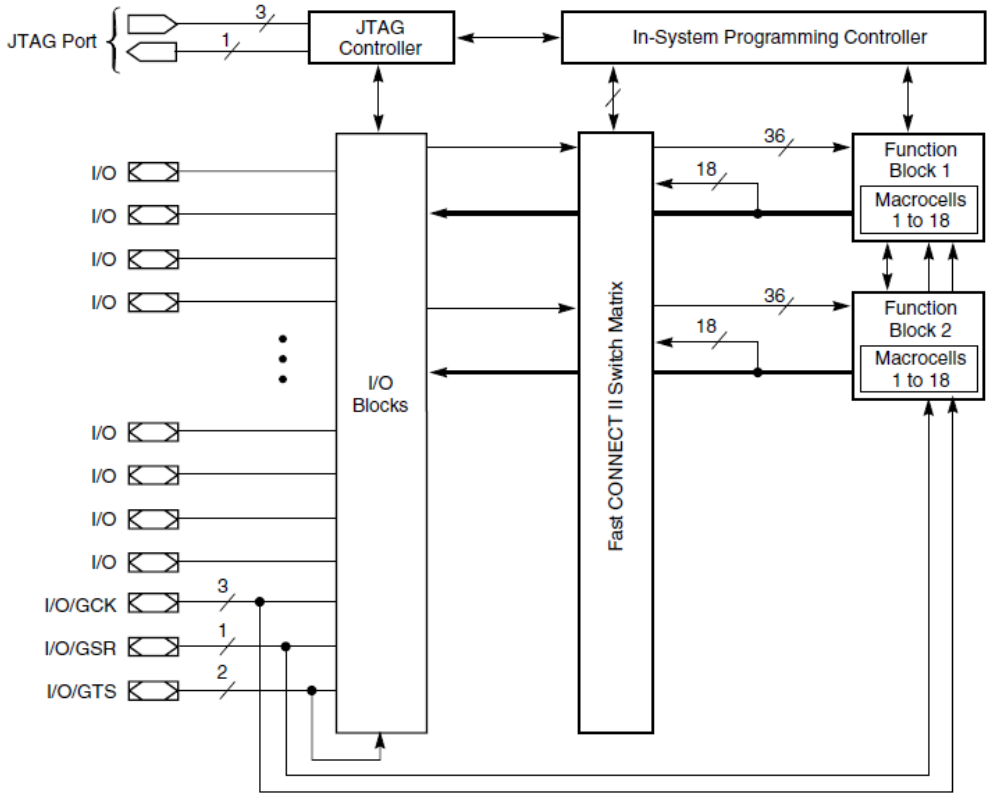
3.1. Programmierbare Logik am Beispiel der CPLDs XC9536 und XC9572

Als CPLD verwenden Sie des Typ XC9536 (oder XC9572) des Herstellers Xilinx.

Es handelt sich um einen der einfachsten Bausteine. Sein 44poliges Gehäuse bietet 36 Anschlüsse, die frei als Ein- oder Ausgänge definiert werden können. Die übrigen Anschlüsse dienen zur Spannungsversorgung und zur Programmierung.

Für die Programmierung des CPLD ist die genaue Kenntnis des Innenlebens oft nicht wichtig. Dennoch sollte man sie in groben Zügen kennen, um die Grenzen des Bausteins abschätzen zu können. So hat der XC9536 nur 36 Flipflops, es ist also nicht möglich, einen 37stelligen Binärzähler damit aufzubauen. Man muß dann einen größeren Baustein nehmen: Der XC9572 hat 72 Flipflops, ist aber etwas teuerer.

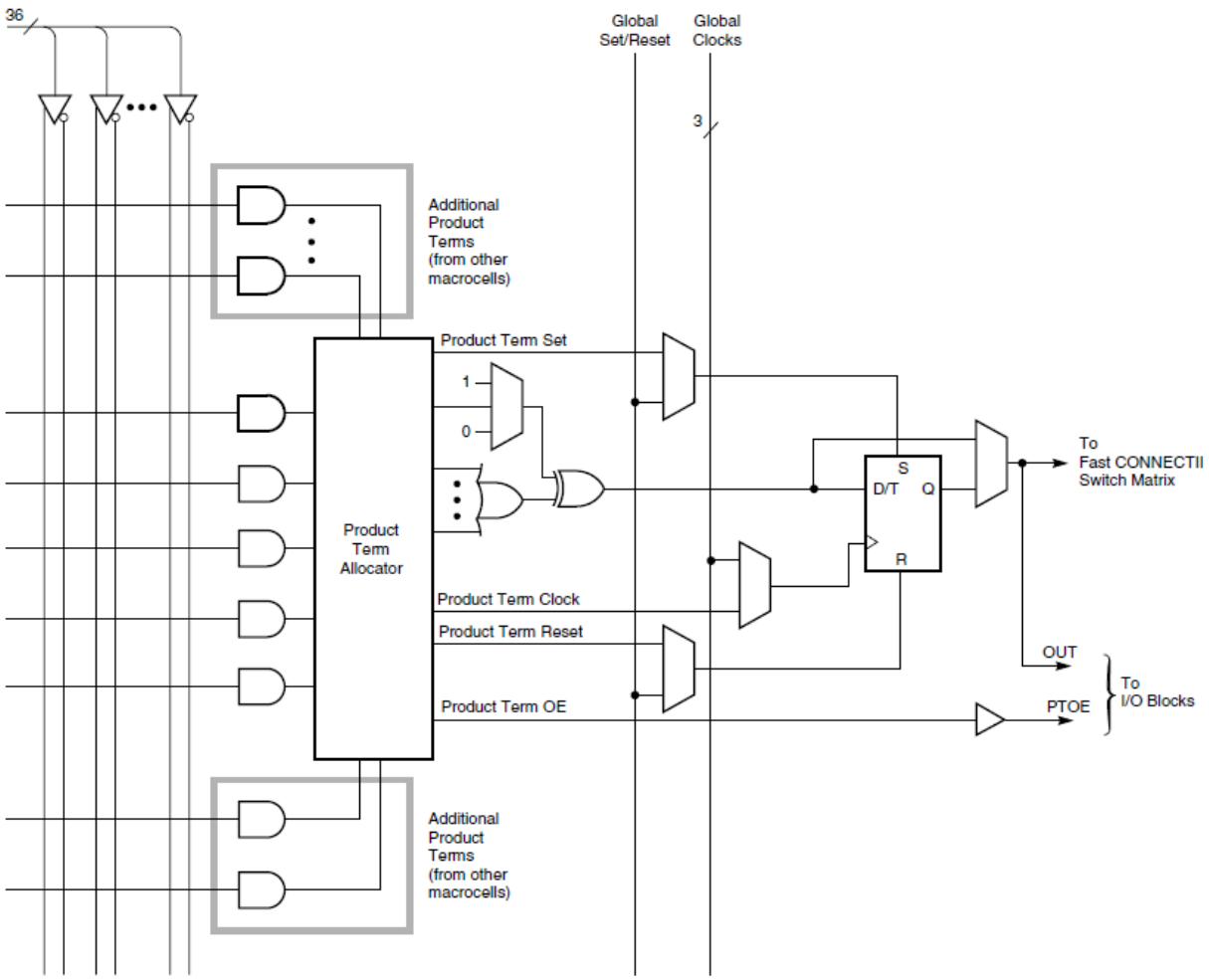
Die folgende Abbildung zeigt die Architektur des CPLDs.



Im Bild werden mehrere Funktionsgruppen deutlich:

- Links befinden sich die 36 I/O (frei als Inputs oder Outputs programmierbar).
- Im Mittelteil können über ein Schaltnetzwerk (switch matrix) Verknüpfungen definiert werden.
- Rechts sind 2 Funktionsblöcke mit je 18 sogenannten Makrozellen. Sie enthalten jeweils ein Flipflop und weitere verschiedene Verknüpfungsmöglichkeiten. (Der XC9572 hat 4 solcher solcher Funktionsblöcke)
- Links oben ist die Programmierschnittstelle. Über sie werden in den Netzwerken Speicherstellen gesetzt oder gelöscht und auf diese Weise definiert, ob im Netzwerk eine Verbindung geschaltet ist oder nicht.
- Links unten können einzelne I/O für Sonderaufgaben eingesetzt werden und z.B. ein für alle Flipflops der Makrozellen gemeinsames Taktsignal (global clock GCK) oder Setz/Rücksetzsignal (global set reset GSR) verteilen.

Werfen wir noch einen Blick auf die Makrozellen. Sie sind folgendermaßen aufgebaut:



Auch hier gibt es Funktionsbereiche:

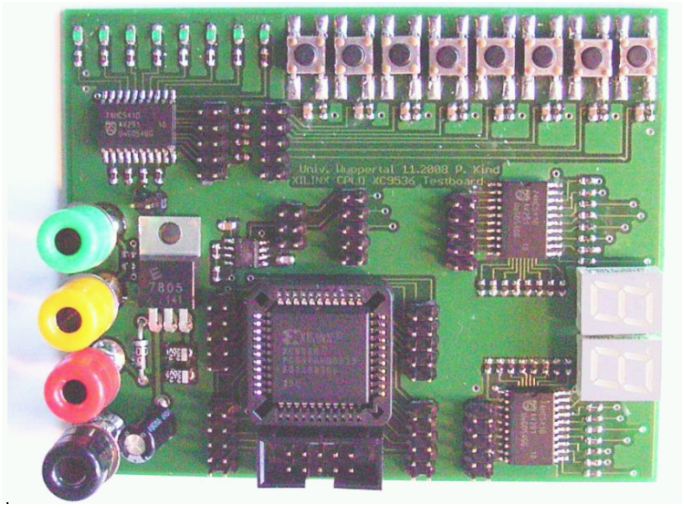
- Von links oben kommen die bis zu 36 Eingänge, sie werden jeweils invertiert und nichtinvertiert weitergeleitet (senkrechte Linien an der linken Seite).
- Diese 72 Signale können beliebig mit zahlreichen AND-Gattern verknüpft werden (waagerechte Linien linke Seite). Theoretisch kann jedes AND-Gatter eine UND-Verknüpfung aller 72 Signale erzeugen. Die AND-Ausgänge werden über eine Matrix namens „Product term allocator“ verknüpft.
- An deren Ausgang stehen nun verschiedene Signale zur Verfügung:
  - *Product Term Set*, *Product Term Clock und Product Term Reset* können als Setz-, Takt oder Rücksetzsignal an das Flipflop gehen.
  - Über ein großes OR-Gatter kann das Signal an den Dateneingang (D/T) des Flipflops gehen. (Das nachgeschaltete EXOR-Gatter betrachten wir an dieser Stelle nicht weiter.)
- In diesem Zusammenhang sind die „Auswahlglieder“ von großer Bedeutung. Es sind die trapezförmigen Symbole. Aufgrund der Programmierung wird z.B. festgelegt, ob der Set-Eingang (S) des Flipflops das Signal *Product Term Set* oder aber das Signal *Global Set/Reset* bekommt. Ähnliches gilt für den Reset-Eingang (R) und den Clock-Eingang.
- Das Ausgangssignal *OUT* (rechts unten) kann entweder der *Q*-Ausgang des Flipflops sein oder dessen Eingangssignal *D/T*, d.h. dann wird das Signal am Flipflop vorbeigeleitet.

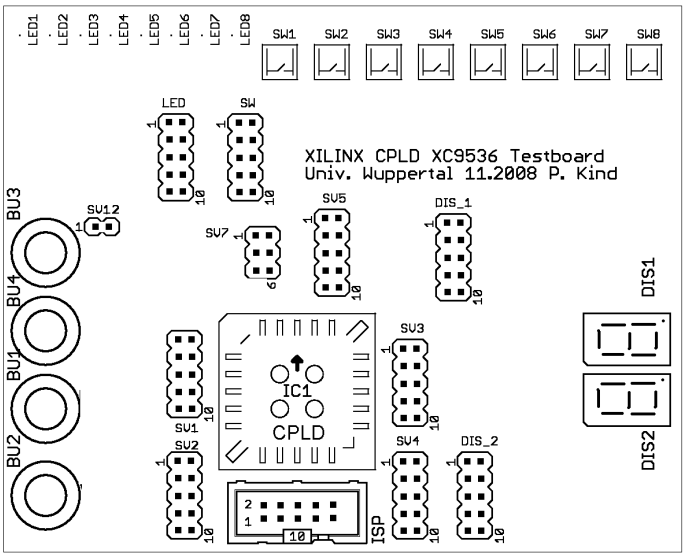
IV. Versuchsdurchführung

1. Hardware: Das CPLD-Board und der Programmieradapter

Der CPLD-Baustein befindet sich auf einer Platine (8 cm x 10 cm), die neben der Spannungsversorgung auch Eingabeelemente (8 Taster), Ausgabeelemente (8 LEDs und 2 Siebensegmentanzeigen) sowie einen Taktgenerator und Anschlüssen für Stromversorgung und das Programmierkabel enthält. Die Verbindung zwischen dem CPLD und den Ein- und Ausgabeelementen geschieht über kurze 10polige Flachbandkabel.

Das Foto zeigt das Board, darunter eine Skizze mit den wichtigsten Bauelementen.





Das große quadratische Bauelement etwa in der Mitte ist der CPLD, um ihn herum gruppiert sind die zugehörigen Pinreihen SV1 bis SV5 für die CPLD-Anschlüsse.

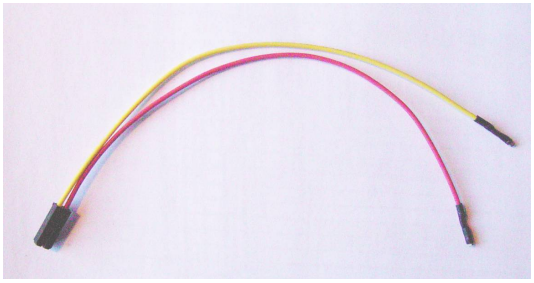
Genau unter dem CPLD ist ein besonderer Stecker ISP, hier wird das Programmierkabel angeschlossen.

Links oben erkennen Sie die 8 LEDs, darunter die zugehörige Pinreihe LED.

Rechts oben sind die 8 Taster SW1 bis SW8, unter SW1 die zugehörige Pinreihe SW.

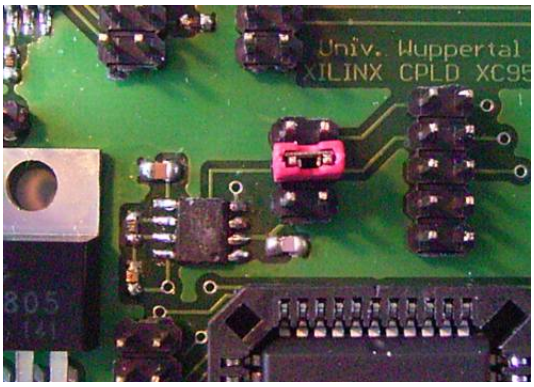
An der rechten Seite sind die beiden Siebensegmentanzeigen DIS1 und DIS2, links daneben die zugehörigen Pinreihen DIS_1 und DIS_2.

An der linken Seite sind die 4 Bananenbuchsen zu erkennen. Für die Spannungsversorgung (9 Volt) ist die rote (plus) und schwarze (minus) Buchse. Die darüberliegende gelbe und grüne Buchse ist über je einen 10-kΩ-Widerstand mit den beiden Pins von SV12 verbunden. Dort kann ein Prüfkabel (Foto) angeschlossen werden, um über die Bananenbuchse Signale zum Oszilloskop zu geben.

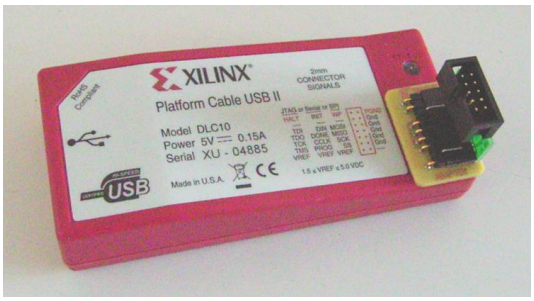


Prüfkabel für SV12

Auf dem Board ist auch ein Taktgenerator. Die Pinreihe SV7 ermöglicht es, sein Taktsignal an den CPLD zu führen. Dazu muß ein Jumper auf die mittleren Pins gesteckt werden (Foto).



Die auf dem PC erzeugte Programmdatei wird über ein USB-Kabel an einen Programmieradapter (Foto) übertragen und von dort über ein 10poligen Flachbandkabel an den Stecker ISP unter dem CPLD.



2. Anleitung CPLD-Programmierung über Schaltpläne mit Xilinx ISE

Die folgende Anleitung zeigt Ihnen Schritt für Schritt, wie Sie mit dem Programm „Xilinx ISE 10.1“ (im folgenden kurz „ISE“) eine kleine Schaltung mit dem CPLD realisieren.

Sie werden zunächst die Schaltung mit Hilfe von ISE als Schaltplan zeichnen.

Danach wird ISE diesen Schaltplan in eine Datei umrechnen, die Sie mit einem Programmieradapter vom PC in den CPLD übertragen.

Wenn das erfolgreich verlaufen ist, wird Ihr CPLD sich so verhalten, wie Sie es im Schaltplan vorgegeben haben. Sie werden dann einige Messungen am CPLD vornehmen.

Anschließend verändern Sie den Schaltplan, erzeugen eine neue Datei, programmieren damit den CPLD neu, und machen weitere Messungen und Beobachtungen.

2.1. Getting started

Das Öffnen der Design Suite

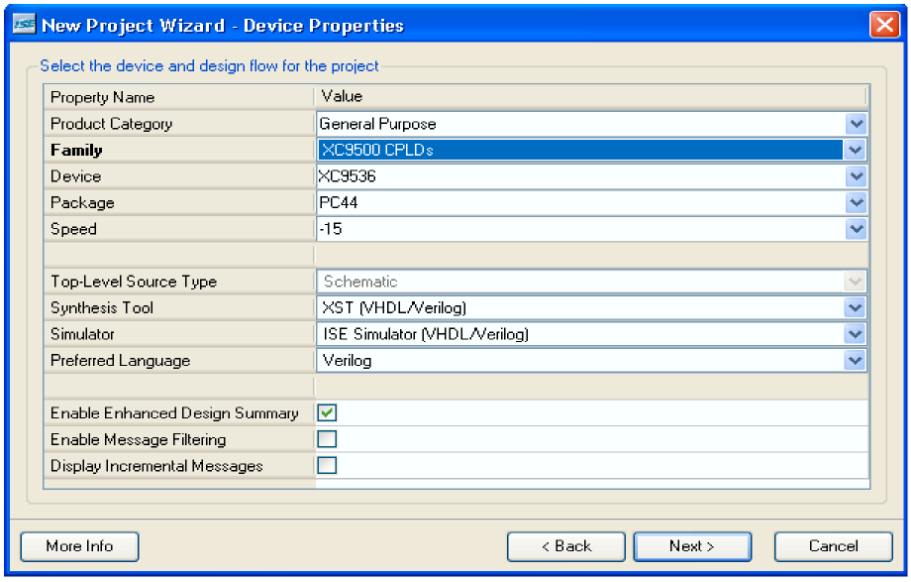
Entweder: Desktop → Xilinx ISE 10.1

Oder: Start → Alle Programme → Xilinx ISE Design Suite 10.1 → ISE → Project Navigator

Ein neues Projekt

Sie sehen nun die Entwicklungsumgebung und möglicherweise ein älteres Projekt.

- Schließen Sie *Tip of the day*
- Starten Sie ein neues Projekt unter *File* (oben links) → *New Project*
- *New Project Wizard* öffnet sich.
- Geben Sie den Namen und den Zielordner unter *Project name* und *Project location* ein. Zum Beispiel: `tutorial1` und `C:/Xilinx/tutorial1`. (Auf Ihrem PC ist evtl. der Laufwerksbuchstabe anders als „C:“)
- Wählen Sie als *Top-level source typ „Schematic“*, um mittels eines Schaltplans zu programmieren
- Klicken Sie *Next*
- Das Fenster zur Konfiguration der *Device Properties* erscheint.
- Stellen Sie die *Device Properties* wie auf der Abbildung unten ein. (ACHTUNG! Prüfen Sie, ob in Ihrer Schaltung ein XC9572 sitzt. Dann müssen Sie als Device natürlich auch XC9572 statt XC9536 einstellen!)



- Klicken Sie *Next*
- Es folgt die Erstellung einer *Source*, die festlegt, wie Sie bei der Programmierung vorgehen, d.h. ob mittels eines Schaltplans (Schematic) oder mittels einer Programmiersprache (VHDL).
- Klicken Sie auf *New Source*
- Es erscheint ein Popup-Fenster *Select Source Typ*, in dem Sie links *Schematic* auswählen und diese benennen. Zum Beispiel: `tutorial1_sch`
- Klicken Sie *Next*
- Sie erhalten nun eine Zusammenfassung über Ihre Source. Beenden Sie diese mit *Finish*
- Erneut öffnet sich ein Popup-Fenster. Klicken Sie *Yes*, damit die Source im selben Ordner gespeichert wird wie das restliche Projekt.
- Sie befinden sich wieder im *Create New Source*-Fenster. Klicken Sie *Next*.
- Im darauf folgenden Fenster bietet sich Ihnen die Möglichkeit, weitere Sources hinzuzufügen. Klicken Sie erneut *Next*
- Beenden Sie die allgemeine Zusammenfassung mit *Finish* und damit die Vorbereitung des Projektes.

2.2. Der Schaltplan

Sie sehen nun wieder die Entwicklungsumgebung vor sich und können mit der Erstellung Ihres Schaltplans beginnen.

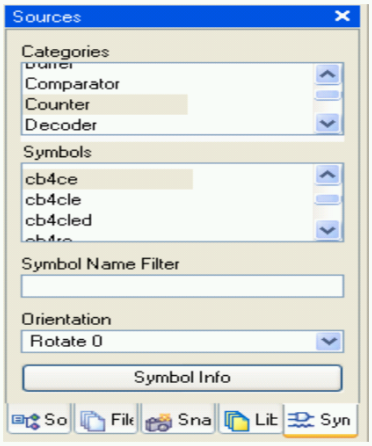
Generelles

Zoom in: F8;
Zoom out: F7;
Zoom to box: F9 + „kastenziehen“;
Zoom to full view: F6;
Undo, Redo, Copy, Cut, Paste standardmäßig unter *Edit* (links oben)

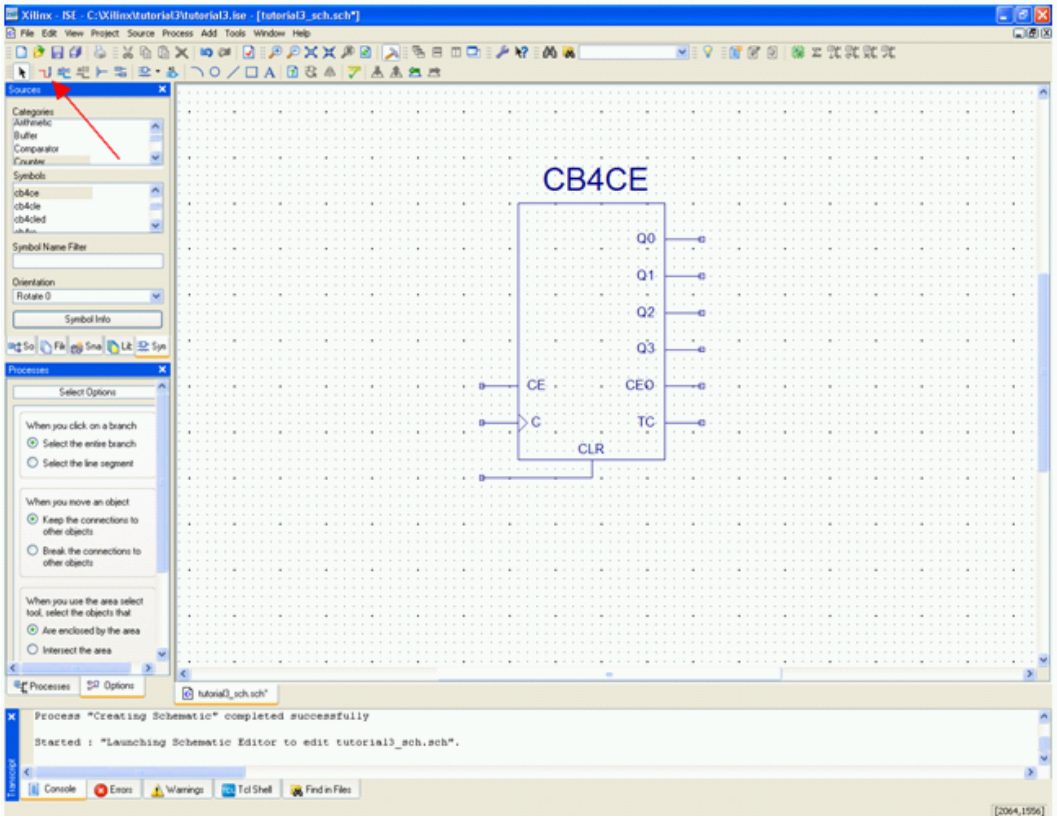
Erstellen eines Schaltplans

Im folgenden werden Ihnen die wichtigsten Funktionen von ISE anhand eines Beispieles gezeigt. Es handelt sich dabei um einen einfachen Zähler, der am Ende auf den LEDs Ihres Boards läuft.

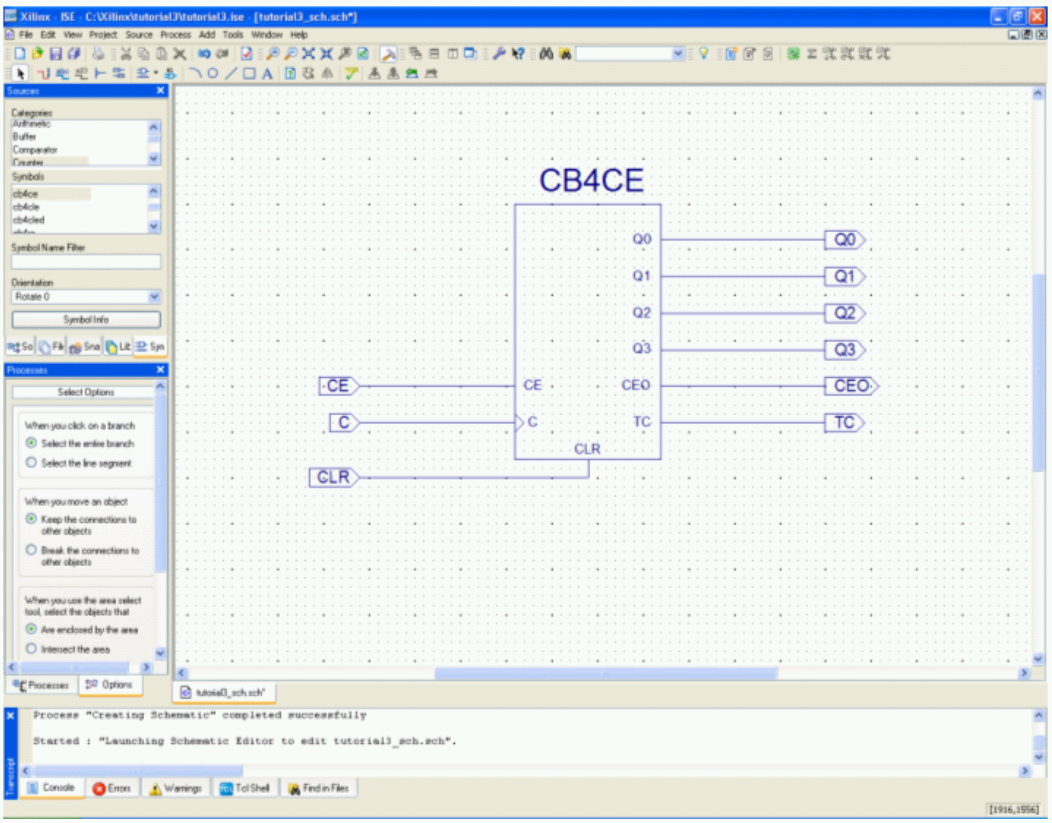
- In der Leiste am linken Rand können Sie das benötigte Bauteil auswählen. Suchen Sie wie in der Abbildung gezeigt den *cb4ce* heraus.



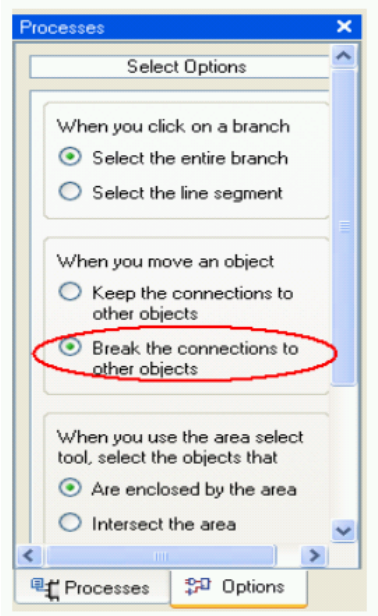
- Sie können auch unter *Symbol Name Filter* `cb4ce` eingeben.
- Klicken Sie nun einmal auf die Bezeichnung (also *cb4ce*). Das Symbol kann jetzt mit der Maus auf die für den Schaltplan vorgesehene Fläche in der Mitte gesetzt werden (absetzen mit Linksklick).
- Plazieren Sie das Schaltsymboles zentrisch. Gefällt Ihnen die Position nicht, können Sie diese verändern, indem Sie den CB4CE mit gedrückter linker Maustaste verschieben.
- Zoomen Sie nun näher. Es sollte nun so aussehen:



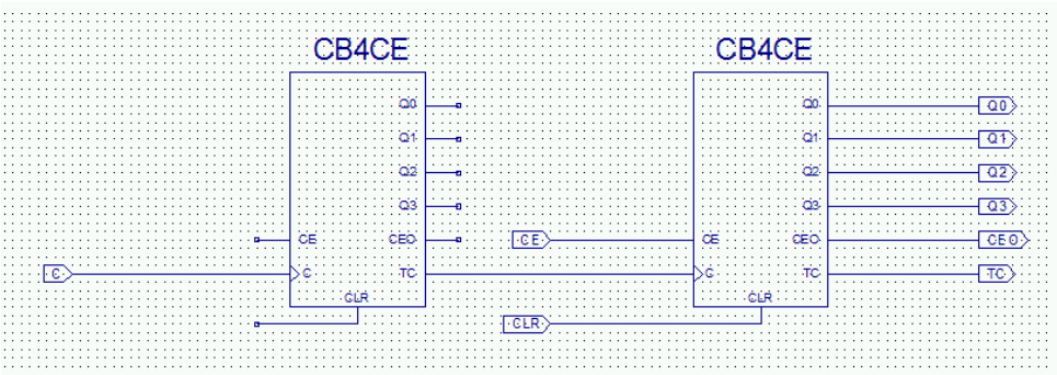
- Wählen Sie nun die Funktion *Add Wire*, auf die der rote Pfeil in der Abbildung oben zeigt, um Leitungen in Ihren Schaltplan zu zeichnen.
- Fügen Sie an jeden der Anschlüsse des CB4CE eine Leitung, indem Sie erst auf den *Add Wire*-Button, dann auf das Quadrat am Ende des Anschlusses klicken, danach die Leitung ziehen und auf den Endpunkt einmal links klicken und zum Schluss Escape drücken.
- Versehen Sie nun das andere Ende der Leitung mit einem „I/O-Marker“, indem Sie auf den vierten Button rechts vom *Add Wire*-Button drücken und dann auf das rote Quadrat am Ende der Leitung linksklicken.
- Die Richtung der Marker legt automatisch fest, ob es sich um einen In- oder Output handelt.
- Ändern Sie nun zum besseren Verständnis die Namen der Marker entsprechend der Anschlüsse, mit denen Sie verbunden sind. Rechtsklicken Sie auf z.B. Marker *XLXN_2* und wählen dann *Object Properties*
- Es erscheint ein Popup-Fenster mit einer Tabelle in der Mitte. Schreiben Sie in das Feld, in dem *XLXN_2* steht, den Namen `CE`, da dieser Marker mit dem Clockenable-Anschluss (CE) des CB4CE verbunden ist. Haben Sie alle Marker entsprechend benannt, sollte es so aussehen:



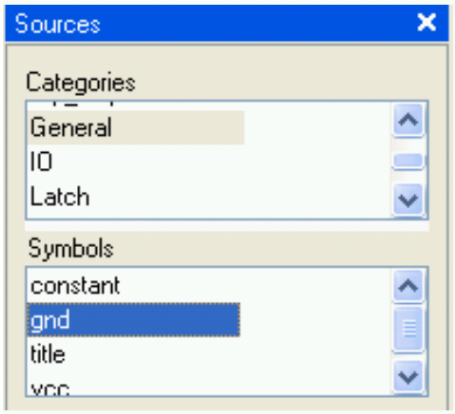
Schalten Sie einen zweiten, gleichartigen Zähler vor den ersten, um die Zählgeschwindigkeit zu drosseln. Trennen Sie dazu erst die Leitungen von C, indem als erstes links *Break the connections to other objects* auswählen:



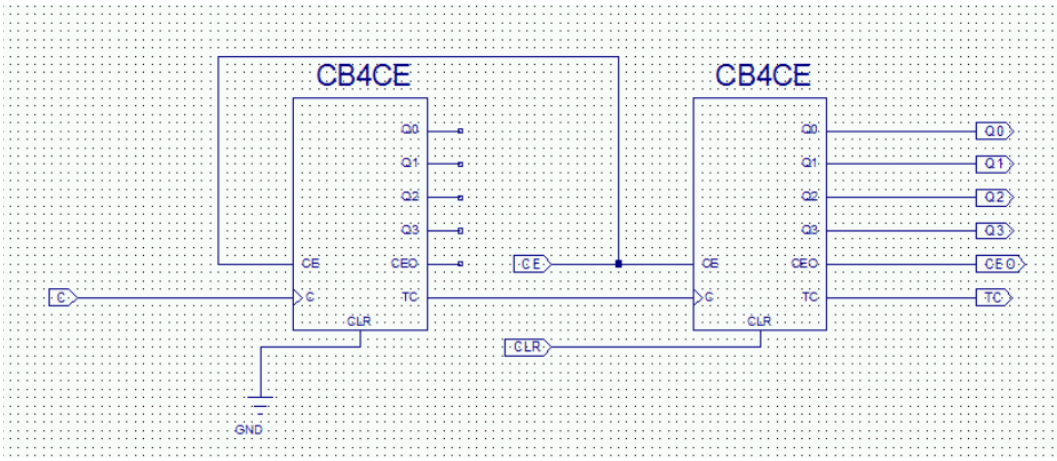
- Ziehen Sie dann gedrückter, linker Maustaste einen Kasten, der sowohl den Marker C als auch die angeschlossene Leitung einfaßt, und bewegen diese beiden weg.
- Nun fügen Sie in den Schaltplan einen zweiten CB4CE ein, wie oben beschrieben und positionieren diesen links neben dem ersten. (Nicht auf die Marker CE und CLR!)
- Verbinden Sie den Ausgang *TC* des zweiten CB4CE mit dem Eingang *C* des ersten mittels *Add Wire*
- Fügen Sie den vorhin abgetrennten Marker C mit seiner Leitung (ziehen einen Kasten der sowohl Marker als auch Leitung komplett einschließt) an Eingang *C* des zweiten CB4CE.
- Dies sollte nun wie folgt aussehen:



- Klicken Sie auf den *Add Wire*-Button.
- Klicken Sie jetzt auf die Mitte der Leitung zwischen Marker CE und Eingang CE des rechten CB4CE und verbinden sie so mit ihrer neuen Leitung.
- Setzen Sie den Endpunkt der neuen Leitung auf den Eingang *CE* des linken CB4CE. Ziehen Sie einfach eine gerade Linie dorthin; die Automatik besorgt den Rest.
- Legen Sie den freien *CLR*-Eingang auf Masse, indem Sie ihn mit dem Schaltsymbol für Masse verbinden, das sie unter *Sources*, *General* finden:



Der Schaltplan sollte nun so aussehen:



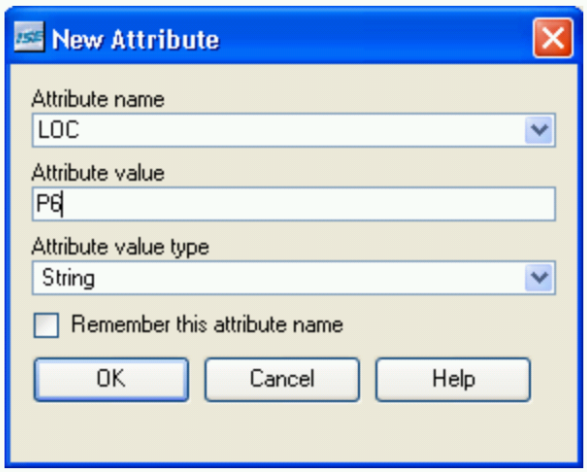
Damit ist der Schaltplan fertig.

Anmerkung für Digitalexperten: Die Verbindung vom TC-Ausgang an den C-Eingang der nächsten Stufe (= CB4CE-Symbol) ist schaltungstechnisch nicht ganz sauber, weil die Stufen nicht gleichzeitig schalten. Korrekt wäre: C-Marker (Clock) an die C-Eingänge *aller* Stufen, CE-Marker nur an den CE-Eingang der ersten Stufe, CEO-Ausgang jeder Stufe an den CE-Eingang der nachfolgenden Stufe. Das Clocksignal wird dann bei den nachfolgenden Stufen nur durchgeschaltet, wenn ein Übertrag stattfindet (diese Stufe um 1 weiterzählen muß) und alle Stufen schalten gleichzeitig (Synchronzähler). Da die CPLDs aber sehr schnell schalten, werden Sie den Unterschied mit dem Oszilloskop kaum messen können.

2.3. Verknüpfung der Marker mit Pins

Da die Marker noch nichts mit den tatsächlichen Pins des Bausteins zu tun haben, müssen diese einander zugewiesen werden.

- Öffnen Sie die *Object Properties* des Markers C per Rechtsklick auf diesen Marker, danach Linksklick auf *Object Properties*.
- Rechts im Popup-Fenster sehen Sie einen Button: *New*. Klicken Sie auf diesen.
- Stellen Sie das nachfolgende kleinere Popup-Fenster so ein:



- *Attribute value* wird mit der Nummer des wirklichen Pins versehen, der C mit einem Clocktakt versorgt, hier also *P6* für Pin 6.
- Bestätigen Sie mit *OK*
- Sie sehen nun bei den *Object Properties* in der mittigen Tabelle eine neue Zeile, die besagt, dass dieser Marker mit Pin 6 verbunden ist.
- **Achtung, wichtig! Ändert sich die Belegung der Pins ihres Devices, müssen Sie dies auch hier in der Programmierung ändern!**
- Weisen Sie den Markern die die folgenden Pins zu:

Marker	Pin
C	P6
CE	P2
CLR	P9
TC	P24
CEO	P22
Q0	P12
Q1	P13
Q2	P14
Q3	P18

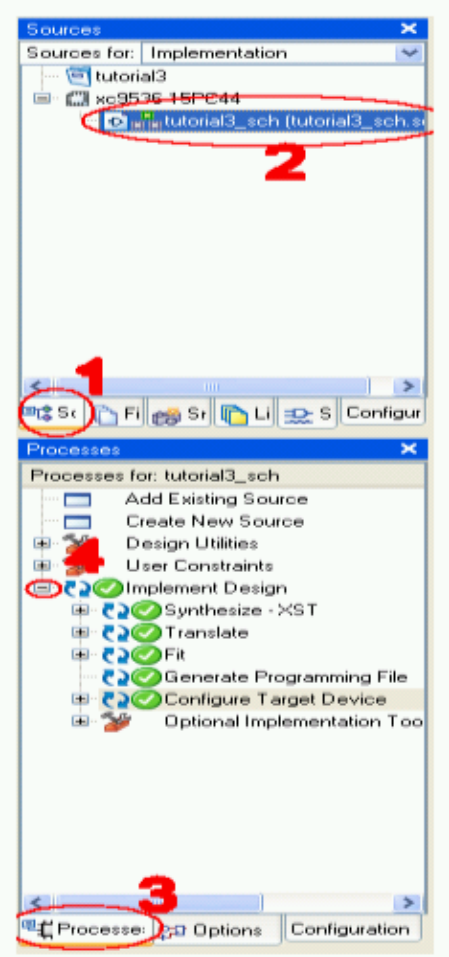
2.4. Implementierung

Nachdem Sie den Markern ihre Pins zugewiesen haben, können Sie nun die Programmierung in den Baustein übertragen (implementieren).

Spätestens jetzt sollten Sie das Programmiergerät (die rote Box) über USB an den PC und über das Programmierkabel an Ihr CPLD-Board anschließen. Außerdem muß Ihr Board mit 9 Volt versorgt werden (rote Buche = Plus, schwarze Buchse = Minus). Achten Sie auf die Lampe in der roten Programmierbox, die sich neben dem Programmierkabel befindet:

- Lampe grün = alles o.k.
- Lampe gelb = Board hat keine Spannung oder Programmierkabel fehlt
- Lampe aus = USB-Verbindung fehlt.

- Ändern Sie die Einstellungen der linken Taskleiste:

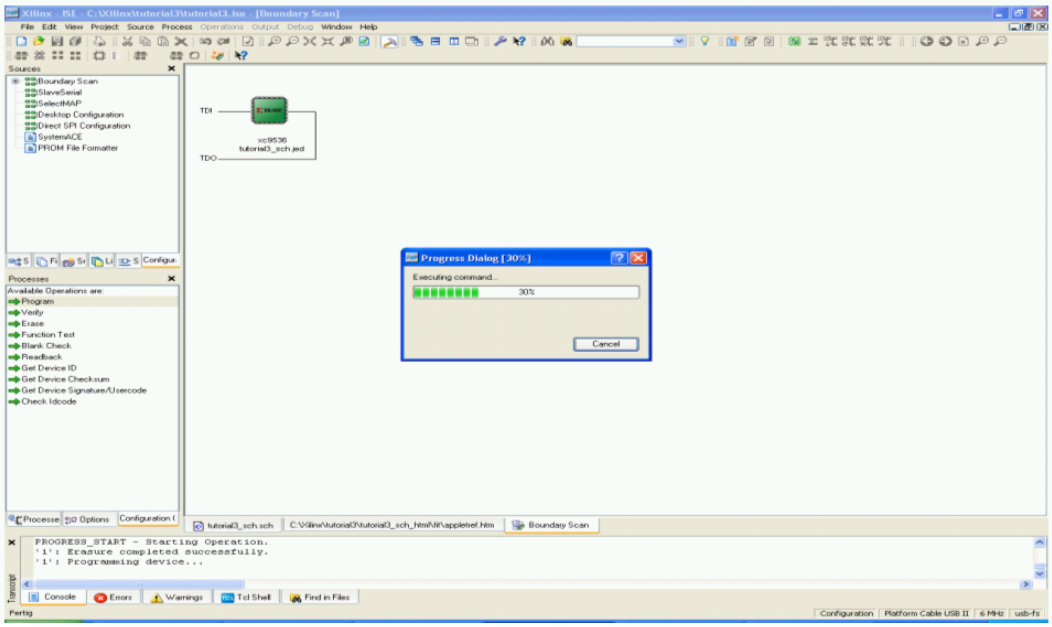


Wählen Sie die rot markierten Optionen der Reihenfolge nach aus, also:

1. Klicken Sie auf die Registertabe *Source*
2. Klicken Sie auf Ihren Dateinamen (die sch-Datei)
3. Klicken Sie ganz unten links auf die Registertabe *Processes*
4. und doppelklicken danach *Implement Design*. Achtung: Falls der Schaltplan geändert wurde, ist es besser rechtszuklicken und dann linksklicken auf *Rerun all.*

- Wenn Sie möchten, können Sie sich unter *Fit* den Fitter-Report ansehen.
- Doppelklicken Sie jetzt *Configure Target Device*, um ihr Programm in den CPLD zu laden.
- Ein Popup-Fenster des Project Navigators öffnet sich und zeigt eine Warnung an, die Sie ignorieren können.

- Als nächstes erscheint ein Fenster des Unterprogrammes *ImPACT*; dieses Fenster können Sie über *Finish* ebenfalls schließen.
- Der Schaltplan wird geschlossen und der *Boundary Scan* des Unterprogramms *ImPACT* wird geöffnet.
- Wieder öffnet sich ein Popup-Fenster, das eine Datei benötigt, die es in den Baustein laden kann, ein „jed-File“. Dieses wurde soeben beim Implementieren erstellt. Wählen Sie es aus und klicken Sie dann auf *Open*.
- Im nächsten Fenster können Sie die Programmierungseinstellungen verändern. Klicken Sie *OK*, wenn *Verify* und *Erase before programing* ausgewählt sind.
- Links sehen Sie jetzt eine Liste: *Available Operations are:* → *Program ...*
- Klicken Sie → *Program*.
- Zum Schluß können Sie mitverfolgen, wie der Baustein programmiert wird.



- Ist der Baustein fertig programmiert erscheint *Programing Suceeded*. Glückwunsch.

Weitere Funktionen wie das Auslesen des Programms oder die Prüfung der „Device-ID “ sind im Anhang dieser Anleitung beschrieben.

Programmieren mit Verilog Eine Einführung in das Programmieren mit Verilog wird im Rahmen des Versuchsteils 4. gegeben.

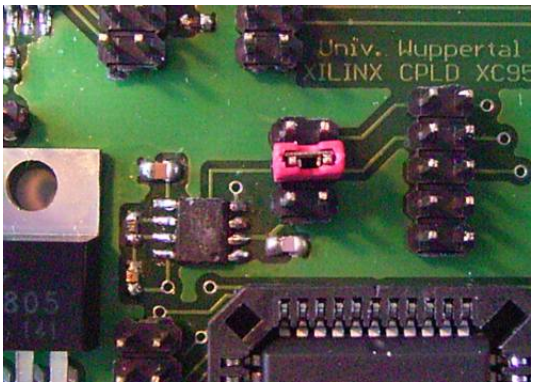
3. Messungen am CPLD und Änderungen der Schaltung

3.1. Messungen an der jetzigen Schaltung

Das Board wird mir einer Versorgungsspannung von ca. 9 V an (rot = plus, schwarz = minus) betrieben; diese haben Sie ja schon zum Programmieren angeschlossen.

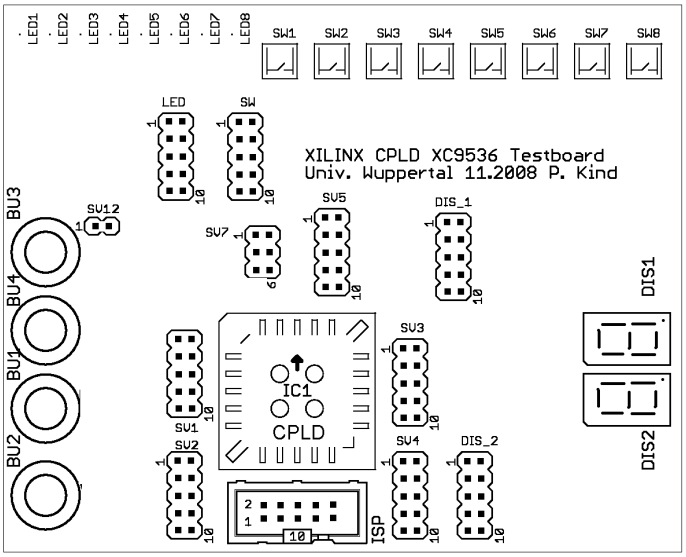
Geben Sie ein Taktsignal auf den CPLD. Es soll vom Taktgenerator zum Pin 6 (GCK1) des CPLDs gehen, den Sie ja als Zählereingang (Signal C) programmiert haben.

Dazu setzen Sie auf der 6poligen **Pinreihe CLK** einen Jumper in die Mitte wie abgebildet. Dann ist diese Verbindung auf dem Board geschaltet

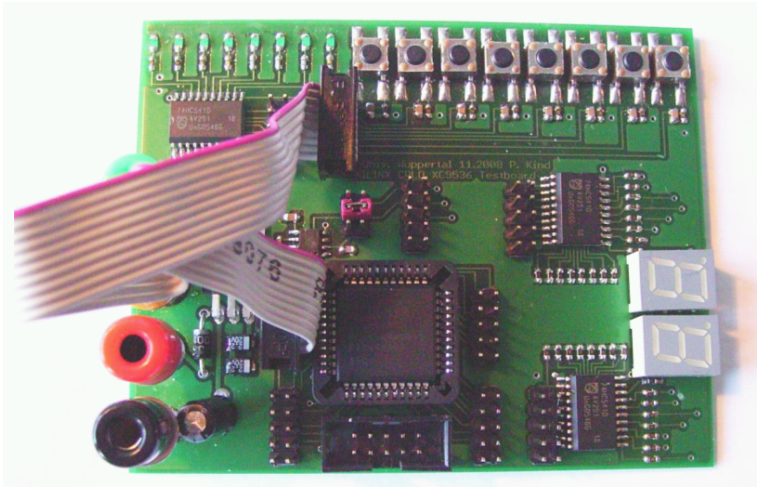


Die Signale CE (Clock-Enable) und CLR (Clear) werden von zwei Tastern erzeugt.

Verbinden Sie die **Pinreihen SV1 und SW** mit einem 10poligen Kabel (Flachbandkabel mit Buchsen). Achten Sie darauf, daß die rote Kante des Flachbandkabels an beiden Buchsen zur Oberkante des Boards weist (wo die Taster sind). Hier noch einmal die Abbildung, wo die verschiedenen Steckerleisten sind:



Auf diese Weise ist Taster **SW1 mit CE** verbunden und **SW8 mit CLR** verbunden. Das sollte so aussehen:



Wenn Sie den Taster drücken, ist das jeweilige Signal logisch 1, sonst 0.

Die Signale Q0 bis Q3 (die Zählerausgänge geben wir zunächst an die einzelnen Leuchtdioden (LED1 bis LED8).

Verbinden Sie die **Pinreihen SV2 und LED** mit einem 10poligen Kabel (Flachbandkabel mit Buchsen). Achten Sie darauf, daß die rote Kante des Flachbandkabels an beiden Buchsen zur Oberkante des Boards weist (wo die LEDs sind).

Auf diese Weise ist **LED1 mit Q0, LED2 mit Q1, LED3 mit Q2 und LED4 mit Q3** verbunden. Wenn Q auf 1 ist, leuchtet die Diode.

LED5 und LED6 sind immer aus (Signal ist immer 0), die LED7 und LED8 zeigen die Signale CEO und TC[1].

Sobald Sie den Taster SW1 (CE) drücken, läuft der Zähler. Da der Taktgenerator aber mit etwa 50 kHz schwingt, sehen Sie keine blinkenden LEDs, vielmehr scheinen alle gleichmäßig zu leuchten. Wenn Sie SW1 (CE) loslassen, bleibt der Zähler auf einem zufälligen 4-Bit-Muster stehen. Daran erkennen Sie, daß er gezählt hat. Drücken Sie auf SW8 (CLR), springt er auf 0000 zurück, alle LEDs gehen aus.

Entfernen Sie das Kabel zwischen den Pinreihen SV2 und LED, denn wir wollen uns die Zählerausgänge mit dem Oszilloskop ansehen. Dazu verwenden wir ein kleines Prüfkabel, dessen 2polige Buchse auf die Pinreihe SV12 (rechts neben der grünen Bananenbuchse) gesteckt wird.

Schließen Sie das Oszilloskop an die folgenden Bananenbuchsen des CPLD-Boards an:
Schwarz = Oszi-GND, gelb = Kanal 1 und grün = Kanal 2.

Mit den beiden freien Enden des kleinen Prüfkabels (rote und gelbe Leitung) können Sie nun an die verschiedenen Pins von SV2 gehen und dort die Signale Q0 bis Q3 abgreifen:

Pin von SV2	Signal	(kommt vom CPLD-Pin . . .)
1	Q0	12
2	Q1	13
3	Q2	14
4	Q3	18

Die Anordnung der Pins in allen Pinreihen ist wie folgt:

1	2
3	4
5	6
7	8
9	10

Messen Sie die Frequenzen an Q0 bis Q3. Stehen Sie erwartet im Verhältnis 1:2? Messen und notieren Sie unbedingt auch die Eingangsfrequenz (CLK, sie ist auch an Pin 3 (oder 5?) von SV5 vorhanden, ebenso an dem roten

[1] TC ist nur 1, wenn Q0 bis Q3 alle 1 sind (also beim Zählerstand 15). CEO (clock enable out) ist nur 1, wenn TC und der CE-Eingang 1 sind. Mit diesen Signalen kann man viele Zähler hintereinanderschalten.

Jumper). Die Frequenz lesen Sie am Oszilloskop rechts unten ab (Triggerfrequenz), das ist genauer als über das Measure-Menü des Oszilloskops.

3.2. Änderungen der Schaltung (1): 16fach-Zähler

Wir wollen nun den Takt auf etwa 1 Hz reduzieren, um verschiedene Digitalanzeigen ansteuern zu können.

Wenn wir 16 Flipflops (also z.B. 4 mal den bekannten 4fach-Zähler) hintereinanderschalten, wird die Frequenz durch 216 = 65536 geteilt. Das ist bei einer Eingangsfrequenz von rund 50 kHz die richtige Größenordnung (wir wollen keine exakten Sekundenimpulse erzeugen).

Überlegen Sie sich den Schaltplan eines solchen 16fach-Zählers, zeichnen Sie ihn und programmieren Sie den CPLD neu. Führen Sie wenigstens den letzten Ausgang der Zählerkette (also Q15) an einen Pin von SV2.

Messen Sie mit dem Oszilloskop die Periodendauer des Ausgangssignals. (Hinweis: Bei Frequenzen unter 10 Hz kann das Oszilloskop die Frequenz nicht mehr automatisch messen. Sie können aber mit der Cursorfunktion die Periodendauer messen, dann wird auch die Frequenz dazu berechnet und angezeigt.)

Wenn Sie mögen, stecken Sie das Flachbandkabel zu den LEDs auf SV2 und sehen Sie die LED(s) blinken.

3.3. Änderungen der Schaltung (2): 3-nach-8-Dekoder

Wir wollen nun die letzten drei Ausgänge (Q13, Q14, Q15) so umkodieren, daß nacheinander 8 Leuchtdioden aufleuchten. Etwas Ähnliches haben Sie beim letzten Versuch EP10 mit dem Baustein 4017 kennengelernt (dort mit 10 Ausgängen).

Überlegen Sie sich, wie Sie einen solchen 3-nach-8-Dekoder mit Gattern aufbauen könnten. Dazu folgende Tips:

- Erzeugen Sie zunächst zu jedem der drei Eingangssignale (nennen wir Sie hier A, B und C) zusätzlich mit einem Inverter das invertierte Signal (also /A, /B, /C).
- Nehmen Sie sich dann acht AND-Gatter mit je drei Eingängen. Jedes dieser AND-Gatter steuert einen der 8 Ausgänge an.
- Überlegen Sie sich für jedes dieser Ausgangsgatter (also für jede Kombinationen von A, B und C), ob Sie A oder /A, B oder /B, C oder /C an die Gattereingänge legen müssen.

  Beispiel: Im Zustand „2“ ist A=0, B=1, C=0. Folglich ist /A=1 UND B=1 UND /C=1. (Schauen Sie sich auch im Anhang der Versuchsanleitung EP10 den vereinfachten Siebensegmentdekoder an).

Die folgende Tabelle zeigt Ihnen noch einmal, in welcher Reihenfolge die Signale am Zähler erscheinen:

Zählerzustand	A = Q13	B = Q14	C = Q15
0	0	0	0
1	1	0	0
2	0	1	0
3	1	1	0
4	0	0	1
5	1	0	1
6	0	1	1
7	1	1	1

Überlegen Sie sich den Schaltplan eines solchen 3-nach-8-Dekoders, zeichnen Sie ihn und programmieren Sie den CPLD neu. Führen Sie dabei die 8 Ausgänge des Dekoders an die Pins von SV3. Sie können dann ein Flachbandkabel von der Pinreihe SV3 zur Pinreihe LED stecken. Achten Sie darauf, daß die rote Kante des Flachbandkabels an beiden Buchsen zur Oberkante des Boards weist (wo die LEDs sind).

Die Pins des CPLDs sind wie folgt mit SV3 und (falls das Kabel steckt) LED verbunden:

CPLD	SV3	LED (Pin-Nr = LED-Nr.)
1	1	1
44	2	2
42	3	3
43	4	4
40	5	5
39	6	6
38	7	7
37	8	8
(GND)	9,10	9,10 (GND)

4. Programmierung der CPLD mit der Hardware Description Language Verilog

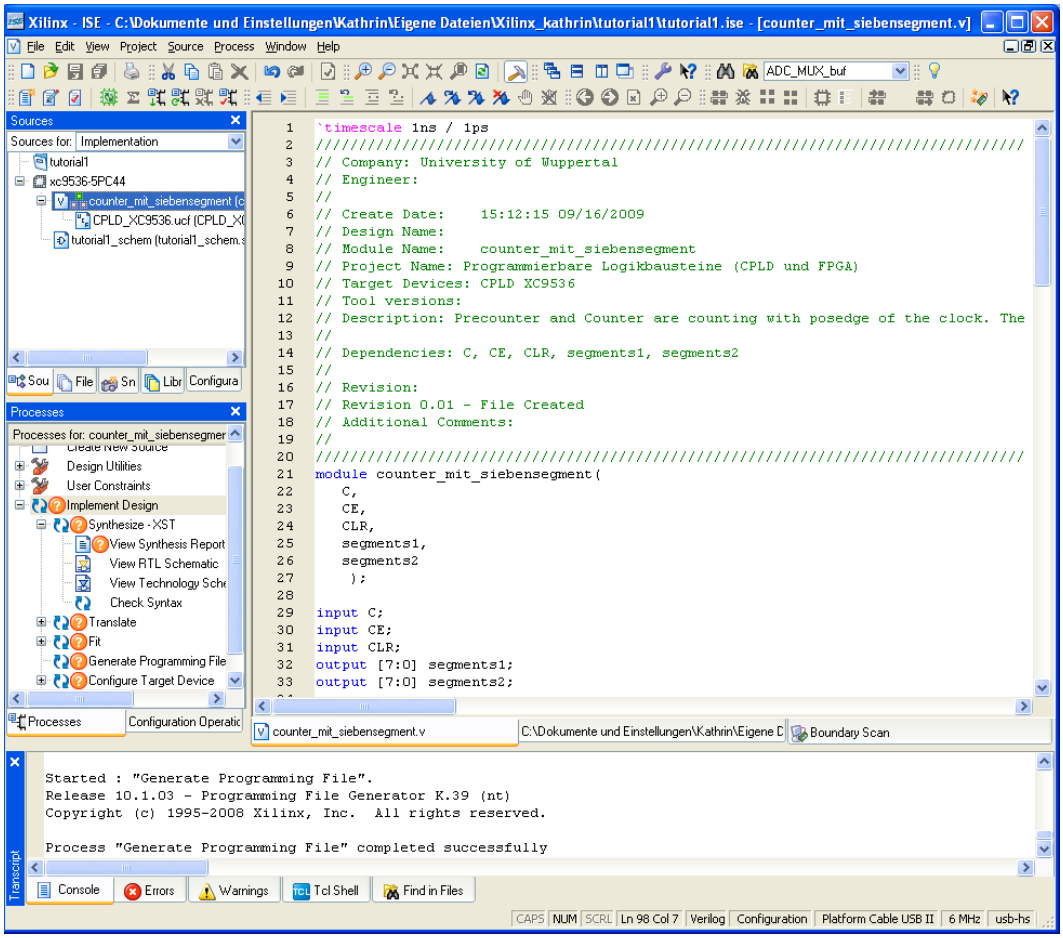
Wie Sie bereits beim letzten Aufgabenteil gesehen haben, wird bei komplizierteren Schaltungen das Erstellen des Schaltplans mühsam. Bei komplexeren Aufgaben ist es deswegen oft schneller, die Schaltung mit einer HDL, in unserem Fall Verilog, zu programmieren. Aufgabe in diesem Teil ist es, einen Vorzähler (12 bis 16 Bit) und einen 8-Bit-Zähler zu programmieren und dann den 8-Bit-Zähler auf den 8 LEDs und den beiden Siebensegmentanzeigen anzuzeigen. Dafür müssen jeweils 4 Bits des Zählers in einen 8-Bit-Zustand der Siebensegmentanzeige übergeführt werden. Im Anhang (Seite 26) finden Sie hilfreiche Verilog-Funktionen, wir geben Ihnen aber zunächst ein lauffähiges Progamm vor.

4.1. Ein neues Projekt

Wir beginnen mit einem ganz neuen Projekt, diesmal soll es aber nicht über einen Schaltplan, sondern über eine Verilog-Datei definiert werden.

- Klicken Sie im Menü *File* auf *New Project*
- Im Fenster *Create New Project* tragen Sie einen passenden Namen ein, wählen ein Verzeichnis und (**wichtig!**) wählen als Top level: *HDL*. Danach klicken Sie *Next*.
- Im Fenster *Device Properties* stellen Sie wie im erste Versuchsteil den Baustein ein (XC9536 oder XC9572). Danach klicken Sie *Next*.
- Im Fenster *Create New Source* könnten Sie eine neue leere Verilog-Datei erzeugen (ganz leer ist sie nicht, ein paar Voreinstellungen werden schon gegeben). Wir wollen aber mit einer bereits vorhandenen Datei anfangen. Klicken Sie daher sofort auf *Next*.
- Jetzt sind Sie im Fenster *Add Existing Sources*. Klicken Sie auf den Button *Add Source* und wählen Sie folgende Verilog-Datei unter folgendem Pfad: `C:\EP10_CPLD_dateien\EP10_00.v`
- Klicken Sie dann im Fenster *Add Existing Sources* nochmals auf *Add Source*, denn ...
- ... die Zuordnung der Signale zu den Anschlüssen des Bausteins ist in einer weiteren Datei (*.ucf = User Constraint File) bereits definiert worden[2]. Wählen Sie (im gleichen Ordner) die Datei: `CPLD_XC9536.ucf`. Wenn Sie beide Dateien (.v und .ucf) geladen haben, wählen Sie *Next*.
- Sie sehen noch eine Zusammenfassung im Fenster *Project Summary*. Klicken Sie danach auf *Finish*.
- Und im Fenster *Adding Source File* auf *OK*.
- Nun sehen Sie die Verilog-Datei im linken Fenster *Sources*. Der ucf-File ist der Verilog-Datei zugeordnet. Klicken Sie doppelt auf die Verilog-Datei, um Sie zu öffnen. Ihr Programm sollte nun dem folgenden Screenshot ähneln.

[2]Der ucf-File legt die Pinbelegung der Inputs und Outputs fest und ist hier von uns vorgegeben. Er entspricht der Verknüpfung der Marker mit Pins bei der Schaltplanprogrammierung. Die Inputs C, CE und CLR befinden sich auf denselben Pins wie in den vorherigen Versuchsteilen. Der Output segments1 liegt auf der Steckerleiste SV3 und segments2 liegt auf Steckerleiste SV4. Sie können sich den ucf-File ansehen, wenn Sie einfach auf ihn klicken und dann im Fenster *Processes* unter *User Constraints* auf *Edit Constraints* klicken.


Xilinx - ISE - C:\Dokumente und Einstellungen\Kathrin\Eigene Dateien\Xilinx_kathrin\tutorial1\tutorial1.ise - [counter_mit_siebensegment.v]
File Edit View Project Source Process Window Help
ADC_MUX_buf
Sources
Sources for: Implementation
tutorial1
xc9536-5PC44
counter_mit_siebensegment (c
CPLD_XC9536.ucf (CPLD_XC
tutorial1_schem (tutorial1_schem.s
Sou File Sn Libr Configura
Processes
Processes for: counter_mit_siebensegmer
Design Utilities
User Constraints
Implement Design
Synthesize - XST
View Synthesis Report
View RTL Schematic
View Technology Sche
Check Syntax
Translate
Fit
Generate Programming File
Configure Target Device
Processes
Configuration Operatic
1 `timescale 1ns / 1ps
2 //
3 // Company: University of Wuppertal
4 // Engineer:
5 //
6 // Create Date: 15:12:15 09/16/2009
7 // Design Name:
8 // Module Name: counter_mit_siebensegment
9 // Project Name: Programmierbare Logikbausteine (CPLD und FPGA)
10 // Target Devices: CPLD XC9536
11 // Tool versions:
12 // Description: Precounter and Counter are counting with posedge of the clock. The
13 //
14 // Dependencies: C, CE, CLR, segments1, segments2
15 //
16 // Revision:
17 // Revision 0.01 - File Created
18 // Additional Comments:
19 //
20 //
21 module counter_mit_siebensegment(
22 C,
23 CE,
24 CLR,
25 segments1,
26 segments2
27);
28
29 input C;
30 input CE;
31 input CLR;
32 output [7:0] segments1;
33 output [7:0] segments2;
counter_mit_siebensegment.v
C:\Dokumente und Einstellungen\Kathrin\Eigene D
Boundary Scan
Started : "Generate Programming File".
Release 10.1.03 - Programming File Generator K.39 (nt)
Copyright (c) 1995-2008 Xilinx, Inc. All rights reserved.
Process "Generate Programming File" completed successfully
Transcript
Console Errors Warnings Tcl Shell Find in Files
CAPS NUM SCRL Ln 98 Col 7 Verilog Configuration Platform Cable USB II 6 MHz usb-hs

4.2. Ein einfaches Verilog-Programm für die 8 LEDs

Wir haben bereits eine Verilog-Datei vorbereitet, die eine sehr einfache Funktion realisiert: Ein 8-Bit-Zähler (counter) wird auf den 8 LEDs angezeigt. Damit der Wechsel der LED-Zustände nicht zu schnell geht, wird das Taktsignal (ca. 50 kHz) durch 12 bis 16 Stufen (einen prescaler) auf etwa 1 Hz heruntergeteilt.

Betrachten Sie den folgenden Programmcode: Hinter dem Doppelschrägstrich // sind Kommentare zu den einzelnen Programmzeilen. Alle Kommentare werden in grüner Schrift dargestellt. Es gibt noch einige weitere Programmzeilen die auskommentiert sind, weil wir sie erst später brauchen. Sie sind im folgenden Text zur Übersichtlichkeit nicht gezeigt.

```
`timescale 1ns / 1ps     // Diese Zeitskala (Schrittweite, Auflösung)
                         // ist nur bei Simulationen wichtig
//////////////////////////////////////////////////////////////////////////////////
// Company: // Engineer:                       VERSCHIEDENE KOMMENTARE
// Create Date:    12:22:32 01/08/2010
// ...
//////////////////////////////////////////////////////////////////////////////////

module testXC9572(     // Jedes Verilog-Programm ist ein Modul mit einem Namen
   C, CE, CLR,                    // und in runden Klammern müssen
   segments1, segments2, LED_OUT  // alle Ein-/Ausgangssignale genannt werden
    );

 input C;                      // Signale als EINgänge definieren
 input CE; input CLR;          // C = Clock, CLR = Clear, jeweils 1 Signal
 output [7:0] segments1;       // Signale als AUSgänge definieren
 output [7:0] segments2;       // hier haben wir z.B. Signalgruppen von
 output [7:0] LED_OUT;         // 8 Bits, daher die Angabe [7:0]

 // reg [7:0] segments1;       // Das ist erstmal noch auskommentiert,
 // reg [7:0] segments2;       // weil wir es erst später brauchen.
  reg [14:0] prescaler;        // reg = Register, das ist praktisch ein D-Flipflop.
  reg [7:0] counter;           // Hier haben wir Register mit 15 bzw. 8 Bit.

  assign LED_OUT = counter;    // Der Inhalt des 8-Bit-Zählers "counter" wird
                               // mit assign direkt den
                               // 8 Bits von LED_OUT zugeordnet.

                                        // Unser Zähler soll mit Signal C getaktet
 always @ (posedge C or posedge CLR)    // und mit CLR zurückgesetzt werden.
   begin                                // (posedge = ansteigende Flanke)
     if(CLR)                            // Wenn also CLR auf 1 ist ...
       begin
         prescaler = 0;                 // dann setzte counter und prescaler auf 0
         counter = 0;
       end
     else                               // andernfalls (also wenn CLR = 0) ...
       begin
         prescaler = prescaler + 1;     // ... erhöhe den prescaler um 1
         if(prescaler == 0)             // und wenn dieser auf 0 ist (das passiert
           begin                        // einmal in 2 hoch 15 Takten!) dann ...
             counter = counter + 1;     // ... erhöhe den counter um 1
           end
       end
   end

endmodule
```

Im Fenster *Processes* unter *Implement Design* → *Synthesize-XST* können Sie auf *Check Syntax* klicken und die Syntax Ihrer Programmierung überprüfen.

Das Programm sollte fehlerfrei sein, daher kompilieren Sie das Programm, wie Sie es soeben mit dem Schaltplan gemacht haben und programmieren den CPLD genauso wie vorher.

Nun machen Sie wie gewohnt Doppelklick auf *Implement Design* bzw. nach Änderungen Rechtsklick auf *Implement Design* und dann *Rerun All*. Auch diesmal sehen Sie den Fortschritt der Implementierung.

Schließlich doppelklicken Sie wie gewohnt auf *configure Target device*. Prüfen Sie aber vorher, ob das Programmiergerät noch über USB mit dem PC verbunden ist und das CPLD-Board noch mit 9 V versorgt wird!

Wie im Schaltplan wählen Sie das jed-File aus, das soeben erzeugt worden ist und mit *Program* wird der CPLD programmiert.

Verbinden Sie mit Flachbandkabeln die Pinreihe SV1 mit der Pinreihe SW und SV2 mit der Pinreihe LED. Achten Sie darauf, dass die rote Kante des Flachbandkabels an allen 4 Buchsen zur Oberkante des Boards weist (wo die LEDs sind).

Sehen Sie auf den 8 LEDs das Binärmuster aus dem hochlaufenden Zähler?

4.3. Änderung 1: Ausgabe des Zählers an die Siebensegmentanzeigen

Wir wollen nun den Zählerstand auf den Siebensegmentanzeigen sichtbar machen. Dazu haben wir in der Verilog-Datei weitere Programmzeilen vorbereitet, die aber noch aktiviert werden müssen. Entfernen Sie bei den folgenden Zeilen die Kommentarzeichen. Die neuen Zeilen beinhalten folgendes zusätzlich:

- Die Outputs segment1 und segment2 waren schon aktiviert (nicht mehr auskommentiert):
- Es müssen aber noch Register hierzu aktiviert werden:

```
reg [7:0] segments1;
reg [7:0] segments2;
```

- Am Ende des always-Blocks ist folgender Befehl:

```
 segments1 = segment7(counter[3:0]);
```

  Mit diesem Befehl werden die unteren 4 Bits von *counter* auf die 8 Bites des Outputs *segment1* gegeben, der die rechte Siebensegmentanzeige ansteuert. Dies geschieht über eine Funktion *segment7*.

- Diese Funktion ist einige Zeilen tiefer wie folgt definiert (beachten Sie die erklärenden Kommentare!):

```
 function [7:0] segment7;       // Eine Funktion mit dem Namen und
                                // 8 Bit großen Ausgabewert segment7

  input [3:0] decodeinput;      // Was an die Funktion übergeben wird,
                                // wird innerhalb der Funktion unter dem
                                // Namen decodeinput geführt (4 Bit).

  case (decodeinput)            // Für den Fall, daß decodeinput ...
    4'h0 : segment7 = 8'b00111111;  // ... den Wert 0 hat: segment7 = Muster "0"
//                       Dgfedcba <<< Stellen für Dezimalpunkt (D) und 7 Segmente
//                                    als 8-Bit-Binärzahl (daher 8'b)
//      decodeinput schreiben wir als 4-Bit-Hexadezimalzahl, also 0-9,a-f
    4'h1 : segment7 = 8'b00000110;   // das Strichmuster "1"
    4'h2 : segment7 = 8'b00000001;   // diese Strichmuster müssen Sie noch ändern
    4'h3 : segment7 = 8'b00000010;
    4'h4 : segment7 = 8'b00000100;
    4'h5 : segment7 = 8'b00001000;
    4'h6 : segment7 = 8'b00010000;
    4'h7 : segment7 = 8'b00100000;
    4'h8 : segment7 = 8'b01000000;
    4'h9 : segment7 = 8'b10000000;
    4'ha : segment7 = 8'b01000000;
    4'hb : segment7 = 8'b00100000;
    4'hc : segment7 = 8'b00010000;
    4'hd : segment7 = 8'b00001000;
    4'he : segment7 = 8'b00000100;
    4'hf : segment7 = 8'b00000010;
 endcase                              // Ende der case-Anweisung
endfunction                           // Ende der Funktion
```

Kompilieren Sie diese Datei und programmieren Sie sie in den CPLD! Verbinden Sie mit Flachbandkabeln die Pinreihe SV3 mit der Pinreihe DIS1. Achten Sie darauf, daß die rote Kante des Flachbandkabels an allen 4 Buchsen zur Oberkante des Boards weist (wo die LEDs sind).

Fragen: Wie können Sie die Geschwindigkeit erhöhen oder erniedrigen, mit der die Zählerzustände durchlaufen werden, ohne die Eingangsfrequenz (ca. 50 kHz) zu verändern?

Das Signal CLR dient zum Rücksetzen des Zählers. Welcher Taster erzeugt es?

Ein weiteres Signal wurde bisher gar nicht benutzt: CE (clock enable). Wie können Sie es erreichen, daß der Zähler nur läuft, wenn CE auf 1 ist? Wie kann man den Zähler stoppen, wenn CE auf 1 ist? Welcher Taster erzeugt CE? (SW5 ist es nicht, dieser schließt nur den Clock C kurz, weil er an die selbe Leitung geht).

4.4. Änderung 2: Strichmuster für alle 16 Zustände

Überlegen Sie sich, wie sie die Muster 2 bis f in der Funktion ändern müssen, damit auch wirklich die Zahlen 2 bis 9 und Buchstaben A bis F erscheinen!

Kompilieren Sie diese Datei und programmieren Sie sie in den CPLD!

4.5. Änderung 3: Aus dem Binärzähler wird ein Dezimalzähler

Was müssen Sie tun, damit der Zähler auf dem Display statt bis F (15) nur bis 9 zählt? Sie müssen ihn zurücksetzen, sobald er den verbotenen Zustand 10 erreicht.

Aktivieren Sie die folgenden mit NEU markierten Zeilen, indem Sie in Ihrer Verilog-Datei die Kommentazeichen // entfernen:

```
if(prescaler == 0)                    // diese Zeilen kennen Sie schon
 begin
     counter = counter + 1;
       if(counter[3:0] == 10)        // NEU
         begin                       // NEU
           counter[3:0] = 0;         // NEU
         end                         // NEU
 end
```

Kompilieren Sie diese Datei und programmieren Sie sie in den CPLD!

4.6. Änderung 4: Ein zweistelliger Zähler

Überlegen Sie sich, was Sie ändern müssen, um auch das zweite Display zu aktivieren. Es soll aber eine Zehnerstelle sein, d.h. der Zähler darf nur bis 99 zählen. Programmieren Sie es, kompilieren Sie diese Datei und programmieren Sie sie in den CPLD!

Verbinden Sie mit Flachbandkabeln die Pinreihe SV4 mit der Pinreihe DIS2. Achten Sie darauf, dass die rote Kante des Flachbandkabels an allen 4 Buchsen zur Oberkante des Boards weist (wo die LEDs sind).

Hinweis: Es kann beim Kompilieren zu einer Fehlermeldung kommen, weil dieses Programm mehr Resourcen braucht als im CPLD vorhanden sind. Verkleinern Sie dann Ihren prescaler auf [13:0] oder [12:0] und versuchen Sie es erneut.

Frage: Wenn Sie den CLR-Taster drücken, springt das Siebensegmentdisplay nicht sofort auf Null. Warum? Wie können Sie das ändern?

4.7. Wenn noch Zeit ist ...

... versuchen Sie einige der folgenden Funktionen zu realisieren:

- Auf Knopfdruck einen bestimmten Zählerstand erzwingen (verwenden Sie dazu den CE-Taster).
- Zähler rückwärts laufen lassen. Bei 00 soll er stehenbleiben (oder nicht).
- Lauflichter mit den 8 LEDs, z.B. von rechts nach links, von links nach rechts, hin und her oder wie „Knight-Rider“ von außen zur Mitte und zurück.
- Zählerstand mit mehreren Knöpfen einstellen (dazu müssen Sie das ucf-File erweitern - Assistenten fragen!) Ein Knopf für die Zehner, einer für die Einer, einer zum Starten und einer zum Stoppen.

V. Anhang

1. Zusammenfassung der Pinbelegungen

SV1	CPLD
1	2
2	3
3	5
4	4
5	6
6	8
7	7
8	9
9,10	GND

SV2	CPLD
1	12
2	13
3	14
4	18
5	19
6	20
7	22
8	24
9,10	GND

SV3	CPLD
1	1
2	44
3	42
4	43
5	40
6	39
7	38
8	37
9,10	GND

SV4	CPLD
1	35
2	34
3	33
4	29
5	28
6	27
7	26
8	25
9,10	GND

SV5	CPLD
1	5 = GCK0
2	42 = GTS0
3	6 = GCK1
4	40 = GTS1
5	7 = GCK2
6	39 = GSR
7	11
8	36
9,10	GND

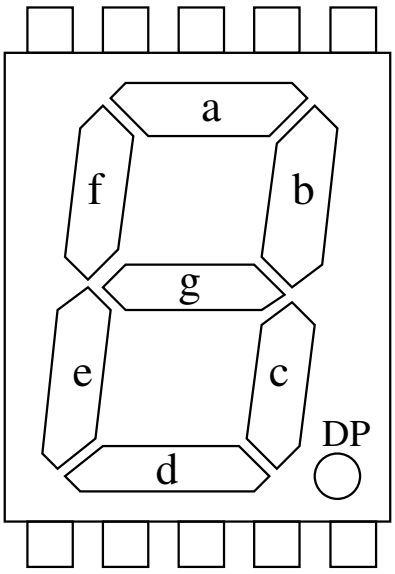
Anordnung der Pins in allen Pinreihen:

1	2
3	4
5	6
7	8
9	10

Die Zuordnung zwischen den LED bzw. Tasten zu ihren Pins ist trivial:

LED1=1, LED2 = 2 usw. ebenso SW1 = 1, SW2 = 2 usw.

Bei der Siebensegmentanzeige gilt ebenfalls eine einfache Zuordnung der Segmente zu den Pins:
a = 1, b = 2, c = 3, d = 4, e = 5, f = 6, g = 7, DP = 8, GND = 9 und 10



2. Datenblätter

Ausführliche Datenblätter zu den CPLDs findet man auf den Internetseiten des Herstellers: `www.xilinx.com`

3. Verilog

3.1. Liste der wichtigsten Befehle und Datentypen in Verilog

Datentypen

```
                        // (hinter dem "//" steht immer Kommentar)
wire wire_name;         // Kabel
reg reg_name;           // Register oder Flip-Flops
input input_name;       // ist per Default ein wire,
                        // muss ggf. noch zusätzlich als reg definiert werden.
output output_name;     // ist per Default ein wire,
                        // muss ggf. noch zusätzlich als reg definiert werden.

wire[3:0], output[1:0], reg[7:0]   // mehrdimensionale Datentypen
```

Wichtige Befehle

- **always**
  - **Syntax:**

    ```
    always @(posedge Clock) // Clock only
    begin
          ... // Synchronous actions
    end
    ```

  - **Beschreibung:**
    Der always-Befehl ist eine prozedurale Zuweisung. Immer wenn die Bedingungen (Flanken) in den Klammern erfüllt werden, wird der Block (zwischen begin und end) ausgeführt. Flanken entsprechen dem Sprung eines Signals von 0 auf 1 (posedge) oder von 1 auf 0 (negedge).
  - **Beispiel:**

    ```
    always @(posedge C or posedge CLR)
    ```

    Der Counter im Block zählt weiter bei posedge C oder wird auf 0 gesetzt bei posedge CLR (siehe Code auf der vorherigen Seite).

- **case**
  - **Syntax:**

    ```
    case (Expression)
       Expression,... : Statement {Expression may be variable}
       Expression,... : Statement
       ... {Any number of cases}
       [default [:] Statement] {Need not be at the end}
    endcase
    ```

  - **Beschreibung:**
    Mit case kann man über den Wert einer Bedingungsvariablen (Expression) über den weiteren Verlauf (Statements) entscheiden. Case ist oft bei sehr umfangreichen if-Bedingungen nützlich.
  - **Beispiel:**
    Dekodierung der Siebensegment-Werte

    ```
    case (decodeinput)    //decodeinput hat 4 Bits
       4'h0: segment7 = 8'b00111111;
       4'h1: segment7 = 8'b00000110;
       4'h2: segment7 = 8'b01011011;
    endcase
    ```

- **function**
  - **Syntax:**

```
function [RangeOrType] FunctionName;
    Declarations ...
    Statement
endfunction
```

  - **Beschreibung:**
    Mit function kann man Statements, die man öfter braucht, auslagern und wiederverwenden. Die Funktion wird im Hauptprogramm mit `FunctionName(Inputs)` aufgerufen. Die Funktionsdefinition steht ganz unten im Code, jedoch vor `endmodule`.

  - **Beispiel:**

```
function [7:0] segment7;
    input[3:0] decodeinput;
    case(decodeinput)
    ...
    endcase
endfunction
```

- **if**
  - **Syntax:**

```
 if (Expression)
 begin
    Statement
 end
 [else
    Statement]
```

  - **Beschreibung:**
    Übliche Bedingungsschleife: Wenn die Expression erfüllt ist, wird das Statement ausgeführt. Ansonsten wird die else-Schleife ausgeführt.
  - **Beispiel:**

```
if (CLR)  //if(CLR == 1)
begin
    prescaler = 0;
end else
begin
    prescaler = prescaler + 1;
end
```

- **module**
  - **Syntax:**

```
module ModuleName [(Port,...)];
        ModuleItems...
endmodule
```

  - **Beschreibung:**
    Modules sind abgeschlossene Programmteile in Verilog. Bei größeren Projekten gibt es ein Topmodul, in dem die Untermodule mit Programmteilen instanziiert (d.h. angelegt und die Übergabe der Werte zwischen den Modulen definiert) werden. Es ist vor allem für die Übersichtlichkeit empfehlenswert, die Module möglichst klein zu halten. In unserem Versuch jedoch arbeiten wir mit nur einem Modul.

- **assign**
  - **Syntax:**

```
assign RegisterLValue = Expression ;
```

- **Beschreibung:**
  Mit assign ordnet man einem wire permanent ein Register oder eine logische Kombination aus wires zu. Sobald sich einen Komponente verändert, ändert sich auch das zugeordnete wire. Praktisch für Signale und Outputs!
- **Beispiel:**

```
reg [11:0] prescaler;
wire signal;
assign signal = prescaler[5];
```

4. Beispiel für ucf-Datei

```
#PACE: Start of Constraints generated by PACE

#PACE: Start of PACE I/O Pin Assignments
NET "C"  LOC = "p6"  ;
NET "CE"  LOC = "P2"  ;
NET "CLR"  LOC = "P9"  ;
NET "LED_OUT<0>"  LOC = "p12"  ;
NET "LED_OUT<1>"  LOC = "p13"  ;
NET "LED_OUT<2>"  LOC = "p14"  ;
NET "LED_OUT<3>"  LOC = "p18"  ;
NET "LED_OUT<4>"  LOC = "p19"  ;
NET "LED_OUT<5>"  LOC = "p20"  ;
NET "LED_OUT<6>"  LOC = "p22"  ;
NET "LED_OUT<7>"  LOC = "p24"  ;
NET "segments1<0>"  LOC = "p1"  ;
NET "segments1<1>"  LOC = "p44"  ;
NET "segments1<2>"  LOC = "p42"  ;
NET "segments1<3>"  LOC = "p43"  ;
NET "segments1<4>"  LOC = "p40"  ;
NET "segments1<5>"  LOC = "p39"  ;
NET "segments1<6>"  LOC = "p38"  ;
NET "segments1<7>"  LOC = "p37"  ;
NET "segments2<0>"  LOC = "p35"  ;
NET "segments2<1>"  LOC = "p34"  ;
NET "segments2<2>"  LOC = "p33"  ;
NET "segments2<3>"  LOC = "p29"  ;
NET "segments2<4>"  LOC = "p28"  ;
NET "segments2<5>"  LOC = "p27"  ;
NET "segments2<6>"  LOC = "p26"  ;
NET "segments2<7>"  LOC = "p25"  ;

#PACE: Start of PACE Area Constraints

#PACE: Start of PACE Prohibit Constraints

#PACE: End of Constraints generated by PACE
```

5. Auslesen des Bausteins mit ImPACT

Wenn Sie CPLDs prüfen wollen oder vorhandene Programme aus ihnen auslesen wollen, können Sie folgendermaßen vorgehen:

- Öffnen Sie ImPACT: Start → Alle Programme → Xilinx ISE Design Suite 10.1 → ISE → Accessories → ImPACT

- Wählen Sie *create a new project...* und klicken Sie *OK*
- Bestätigen Sie das folgende Fenster mit *Finish*
- Jetzt verbindet sich das Programm mit dem Baustein (target, Ziel). Warten Sie bis dies beendet wurde.
- Da Sie nun nichts programmieren möchten, sondern auslesen möchten, beenden Sie das nächste Popup-Fenster *Assign New Configuration File* mit *Cancel* und das folgende zur Bestimmung der Programmiereinstellungen ebenfalls mit *Cancel*.
- An den unterstrichenen Stellen der folgenden Abbildung sehen Sie nun Firma und Baustein ID. Sollte dort nichts angezeigt werden, klicken Sie *Get Device ID*.
- Mit *Erase* können Sie den Baustein (den CPLD) löschen.
- Mit *Blank Check* prüfen Sie, ob der Baustein gelöscht ist (leer, blank).
- Mit *Readback* lesen Sie das Programm aus dem Baustein wieder aus und speichern es als jed-Datei ab. Sie könnten damit weitere CPLDs programmieren. Diese jed-Datei ist aber nicht der Schaltplan oder die Verilog-Datei! Sie können die jed-Datei zwar mit einem Texteditor öffnen, es ist aber nahezu unmöglich, aus den Zahlenkolonnen auf die programmierten Funktionen zu schließen.