CPLD: Erste Schritte mit dem Xilinx XC9572XL

1. Einleitung

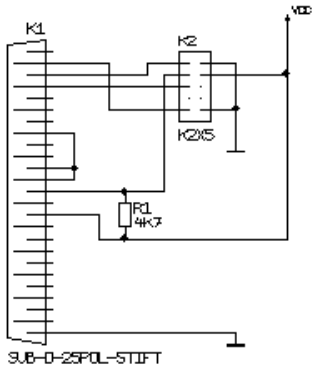
Statt einzelner Logikgatter werden in heutigen Schaltung immer häufiger komplexe programmierbare Logikbausteine, sogenannte CPLDs, verwendet. Hauptvorteile sind der verminderte Flächenbedarf und die höhere Flexibilität; CPLDs können nämlich auch in der fertigen Schaltung umprogrammiert und damit ihr logisches Verhalten erweitert oder angepasst werden.
Da mir eine derartige Anleitung fehlte, möchte ich auf dieser Seite die nötigen Hilfsmittel und die ersten Schritte beschreiben, die zur Programmierung eines Xilinx XC9572XL nötig sind. Dabei wird auf teure Hardware verzichtet und nur Eigenbau-Schaltungen mit günstigen Komponenten (insgesamt weniger als 15 EUR) verwendet.

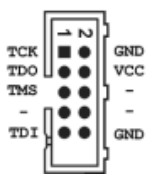
2. Xilinx XC9572XL

Der Xilinx XC9572XL in PLCC-Form kostet weniger als 4 EUR und ist ausserdem im Versandhandel leicht beschaffbar (z.B. bei Reichelt). Die Beschreibung sollte auch für andere ICs der XC9500er Familie gelten bzw. leicht übertragbar sein.
Ein Blick ins Datenblatt (XC9572XL) verrät, dass der XC9572XL mit **3,3 Volt versorgt** werden muss und **5V-tolerante Ein-und Ausgänge** besitzt. Die Programmierung erfolgt über die **JTAG-Schnittstelle**.

3. JTAG-Programmieradapter

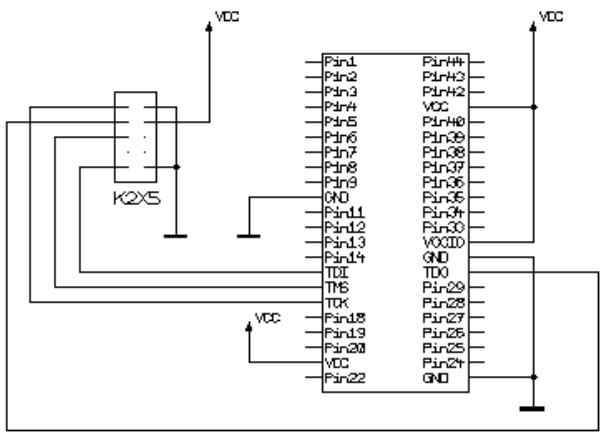
Um den Logikbaustein über den PC programmieren zu können, wird ein passender Adapter gebraucht. Die Verbindung erfolgt meist über den Druckerport. Dabei gibt es grundsätzlich zwei unterschiedliche Aufbauten:
Die **sichere Variante mit Pufferbaustein**, welche vom Hersteller vorgeschlagen wird: JTAG/Parallel Download Cable
Die **Einfachstvariante**, die nur aus den zwei Steckern, einem Widerstand und ein paar Drähten besteht, dafür aber bei einigen Druckerports nicht funktioniert. Ich habe mich für diese Variante entschieden und bisher keine Probleme damit gehabt.
Einfachstvariante:



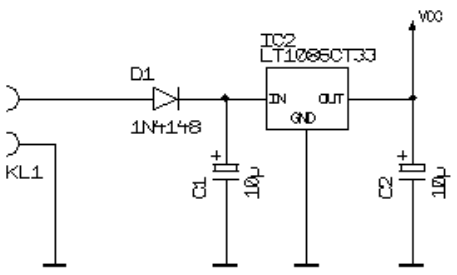


4. Experimentierboard

Statt teuren Entwicklungsboards oder Selberätzen habe ich für meine ersten Versuche einfach eine 160x100mm große Lochraster-Platine verwendet, darauf mittig einen PLCC-Sockel angelötet und mit Schaltlitze nach folgender Grundschaltung verdrahtet:

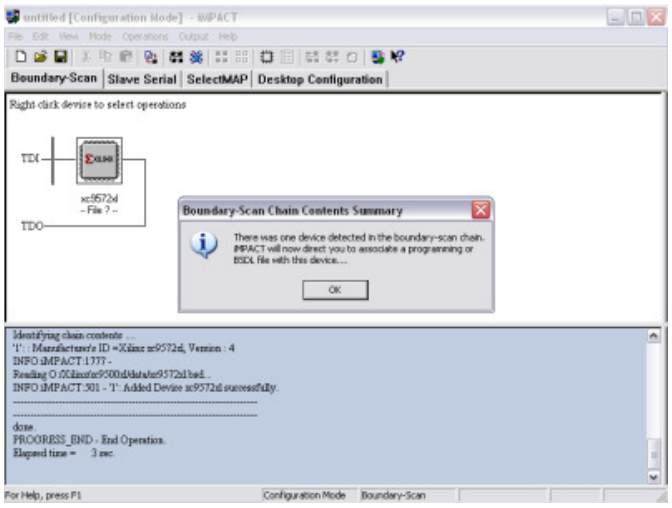


Für den Aufbau ist es ganz hilfreich, die Belegung der PLCC-Fassung von unten gesehen, vor sich zu haben. Deswegen gibt's hier die Untenansicht als PDF-File: plcc44_unten.pdf. In der Datei sind auch die JTAG-Anschlüsse aufgeführt und die Pins für die Spannungsversorgung markiert. Die Versorgungsspannung (VCC) beträgt 3,3 Volt; sie kann entweder mit einem (Labor-)Netzteil oder mit folgender Schaltung bereitgestellt werden:



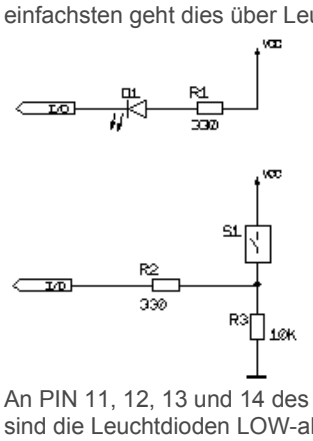
5. Erster Test

Ist die Grundschaltung aufgebaut und ein JTAG-Programmieradapter vorhanden, kann der Aufbau durch einen ersten Kontaktversuch zwischen PC und CPLD getestet werden. Man benötigt dafür natürlich die entsprechende Software für den PC. Dazu gibt es **ISE WebPACK**, das nach einer (kostenlosen) Anmeldung von der Herstellerseite (XILINX) heruntergeladen werden kann. Das Softwarepaket enthält alle Tools, die für den Entwurf und die Programmierung der Logikbausteine nötig sind.
Zur Programmierung dient das Tool **iMPACT**. Beim Starten öffnet sich ein Wizzard, in dem man "Configure Devices" wählt, dann "Boundary-Scan-Mode" und "Automatically connect to cable and identify...". Klickt man anschließend auf "Finish", versucht das Programm das JTAG-Kabel und angeschlossene Chips zu erkennen. Wenn alles passt, wird der erkannte Baustein angezeigt:



6. Ein- und Ausgabe

Wenn der Grundaufbau den ersten Funktionstest bestanden hat, kann die Schaltung um Ein- und Ausgabeeinheiten erweitert werden. Am einfachsten geht dies über Leuchtdioden bzw. DIP-Schalter:

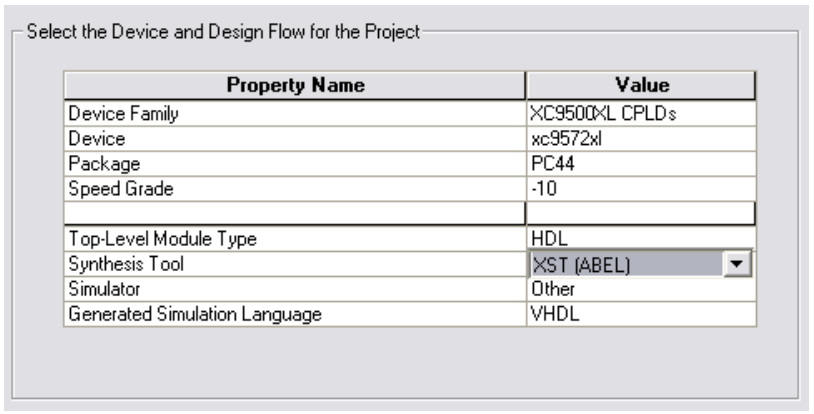


An PIN 11, 12, 13 und 14 des Bausteins habe ich Schalter angeschlossen und an PIN 33, 34, 35 und 36 Leuchtdioden. So wie hier gezeichnet, sind die Leuchtdioden LOW-aktiv, d.h. sie leuchten, wenn der Ausgang auf 0 gesetzt ist und erlöschen bei einer 1. Das muss bei der Programmierung natürlich beachtet werden!

7. Einfache logische Funktionen: UND und ODER

Jetzt sind hardwaremäßig die Voraussetzungen geschaffen, um eine erste Funktion in den Logikbaustein zu "brennen" und das Ergebnis zu überprüfen.

Der Entwurf soll vorerst in der Hardwarebeschreibungssprache (HDL) ABEL erfolgen. Dazu wird der "Project Navigator" vom WebPack aufgerufen und ein **neues Projekt angelegt**: "File -> New Project", Projektname und Speicherort eingeben, "HDL" wählen und im nächsten Dialog den verwendeten CPLD einstellen und die Synthesetools wählen:


Select the Device and Design Flow for the Project

Property Name	Value
Device Family	XC9500XL CPLDs
Device	xc9572xl
Package	PC44
Speed Grade	-10
Top-Level Module Type	HDL
Synthesis Tool	XST (ABEL)
Simulator	Other
Generated Simulation Language	VHDL

Im nächsten Dialog wird über "New Source..." eine neue Quelldatei angelegt. Dazu "ABEL-HDL-Module" wählen und Dateinamen eingeben. Alle weiteren Dialoge mit "Next" bzw. "Finish" ohne spezielle Eingabe übergehen.
Nun ist das neue Projekt angelegt und im Hauptfenster wird die **ABEL-Datei zum Editieren** angezeigt. Diese kann nun erweitert werden zu (Kommentare werden mit Anführungszeichen eingeleitet):

```
MODULE test

    S0 pin 11;
    S1 pin 12;
    !L0 pin 33;    "negiert wegen low-aktiver LED"
    !L1 pin 34;    "negiert wegen low-aktiver LED"

  EQUATIONS
    L0 = S0 & S1;  "UND-Verknüpfung"
    L1 = S0 # S1;  "ODER-Verknüpfung"

END
```

Jetzt kann das ABEL-"Programm" **übersetzt** werden. Links gibt es dazu ein Fenster "Process View". Darin nacheinander die Unterpunkte "Compile Design", "Translate", "Fit" und "Generate Programming File" doppelklicken und jedesmal prüfen, ob Fehler bzw. Warnungen ausgegeben wurden.

Zum **Brennen** muss iMPACT aufgerufen werden. Dies kann direkt aus der Entwicklungsumgebung durch Doppelklick auf "Configure Device (iMPACT)" erfolgen. Nun wie oben beschrieben den Boundary-Scan durchführen. Der Baustein sollte nun im Hauptfenster angezeigt werden. Über einen Doppelklick kann ihm eine JEDEC-Datei (die vorhin durch "Generate Programming File" erzeugt wurde) zugeordnet und anschließend durch einen Rechtsklick -> "Program..." mit den Optionen "Erase Before Programming" und "Verify" gebrannt werden.

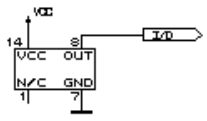
Jetzt besitzt der CPLD die zuvor definierten Funktionen!
Das kann natürlich gleich getestet werden! Wenn einer der Schalter an PIN 11 oder 12 geschlossen wird, leuchtet die LED an PIN 34. Sind beide Schalter geschlossen, leuchten beide LEDs.

Wie der CPLD intern verschaltet wird, kann übrigens über ein spezielles Tool, das sich **ChipViewer** nennt, angesehen werden. Dieser ist über "Implement-Design -> Fit -> View Fitted Design" erreichbar. Der XC9572XL ist in vier Funktionsblöcke unterteilt. Jeder Funktionsblock besteht aus 18 Makrozellen. Jede dieser Makrozellen stellt eine ODER-Verknüpfung von Produkttermen (UND-Verknüpfung von nichtnegierten und negierten Variablen) dar. Im ChipViewer werden belegte Makrozellen schwarz markiert und können durch einen Doppelklick genauer untersucht werden. So kann man mehr über den Aufbau eines CPLDs lernen und sehen, wie die Gleichungen umgesetzt werden. Die Struktur der Logikbausteine wird in einer ApplikationNote näher beschrieben: XAPP073 - Designing with XC9500 CPLDs.

8. Im Takt: ein Dualzähler

Bisher wurde nur ein Schaltnetz umgesetzt - die einzelnen Flipflops der Makrozellen hatten daher noch nichts zu tun. Die Flipflops könnten zwar auch manuell über die Schalter angesprochen werden, jedoch prellen diese beim Umschalten. Das führt zu unerwünschten Schaltvorgängen, was bei der Ansteuerung von Flipflops recht ungünstig ist. Aus diesem Grund und weil es das Ganze etwas spannender macht, wird dem CPLD ein Takt gegeben. Dies geschieht mit einem Quarzoszillator, der an PIN 5 (GCK1) angeschlossen wird (ich habe einen 4MHz-Quarzoszillator

gewählt):



Um von den 4MHz überhaupt etwas sehen zu können, muss der Takt heruntergeteilt werden. Dies geht mit einem Zähler. Hier wird ein 23bit-Zähler definiert und die oberen vier Bits dieses Zählers auf die LEDs gelegt:

```
MODULE test

    CLK pin 5;
    S0 pin 11;
    ![L0..L3] pin 33, 34, 35, 36;
    [Q22..Q0] node istype 'reg';

    Counter = [Q22..Q0];

  EQUATIONS

    Counter.CLK = CLK;
    Counter := Counter + 1;

    [L0..L3] = [Q22..Q19];

END
```

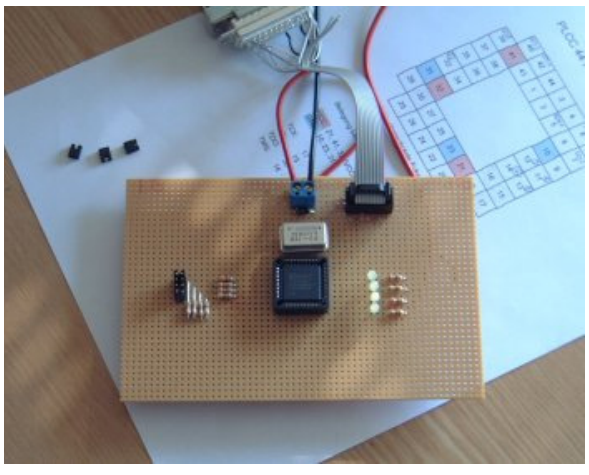
Die Zuweisung mit dem Doppelpunkt ":=" ist eine Registerzuweisung. Die Zählerbits Q0..Q22 müssen daher auch als Register definiert werden: istype 'reg'. Jedem genutzten Register muss ein Taktsignal zugewiesen werden. Hier ist es das Taktsignal vom Quarzoszillator. "node" steht für einen internen Knoten im Gegensatz zu "pin", bei dem das Signal nach aussen gelegt wird. Die eckigen Klammern definieren Mengen, auf die direkt Operationen ausgeführt werden können. So kann man beispielsweise die Ausgabe auf den LEDs abhängig von der Schalterstellung an PIN 11 machen:

```
    [L0..L3] = ([Q22..Q19] & S0) # ([Q20..Q17] & !S0);
```

Ist der Schalter offen, werden die Bits Q20 bis Q17 ausgegeben. Diese Bits ändern sich logischerweise öfter und die Ausgabe auf die LEDs ist daher hektischer.

9. Zum Schluss...

...noch ein Bild von meinem Experimentierboard mit LEDs, Jumpern (statt DIP-Schalter) und JTAG-Programmieradapter:



Thomas Fischl
(02/2005)

Externe Links zum Thema

http://www.xilinx.com/cpld/ Hersteller von CPLDs
http://www.seas.upenn.edu/ese/rca/software/abel/abel.primer.html ABEL-HDL Primer
http://www.mikrocontroller.net Forum mit Bereich "Programmierbare Logik"