vhdl2cpld: Designflow unter Linux für CPLDs mit den Xilinx-WebPack

V0.41 (c) 2005-2011 Joerg Wolfram

1 Allgemeines

Das Programm unterliegt der GPL (GNU General Public Licence) Version 3 oder höher, jede Nutzung der Software/Informationen nonkonform zur GPL oder ausserhalb des Geltungsbereiches der GPL ist untersagt! Die Veröffentlichung dieses Programms erfolgt in der Hoffnung, daß es Ihnen von Nutzen sein wird, aber OHNE IRGENDEINE GARANTIE, auch ohne die implizite Garantie der MARKTREIFE oder der VERWENDBARKEIT FÜR EINEN BESTIMMTEN ZWECK.

2 Warum?

Nachdem ich schon längere Zeit nach einer Möglichkeit, VHDL zu lernen und auch praktisch anzuwenden, bin ich auf die freie Linux-Version des Xilinx-Paketes gestossen. Nach etwa 10 Stunden Download (ISDN) war die erste Hürde genommen. Die Installation hat dann auch geklappt, aber mit dem Hinweis, dass die Kabeltreiber als root installiert werden müssen.
Nach dem oder besser beim Start kam dann schnell die Ernüchterung. Auf meinem Laptop (IBM Thinkpad 600 mit Pentium II-300, Linux auf Basis von SuSE 9.0) dauerte dieser schon ein paar Minuten. Und auch jede Änderung dauerte ewig, der Schematic Editor liess sich gar nicht nutzen. Mit dem VHDL-Editor ging es schon etwas besser, wegen dem fehlenden Kabeltreiber kam ich nur bis zur .jed Datei, aber das war schon OK. Wesentlich ärgerlicher war, daß ich beim nächsten Start mein Projekt nicht mehr öffnen konnte. Das Internet war auch keine große Hilfe, bis ich dann von Kommandozeilentools gelesen und mir die verschiedenen Dokumentationen im Xilinx-Verzeichnis durchgelesen habe. Ein paar Versuche mit der schonmal erfolgreich übersetzten VHDL-Datei haben mich dann zum Shell-Script **vhdl2cpld** gebracht, welches das Ganze weitestgehend automatisiert. Inzwischen ist daraus daraus ein eigenes Projekt.

Warum CPLD's und keine FPGA's?
Das hie beschriebene Vorgehen kann bei kleinen Änderungen am Script auch für FPGA's verwendet werden, aber gerade mit den 44-Pin-CPLD's lassen sich auch noch Projekte mit einlagigen Leiterplattenlayouts (oder auch Lochrasterplatten) realisieren, was die Gestaltung und den Nachbau von Projekten erheblich erleichtert.
Das Projekt befindet sich noch im Fluss und erhebt keinen Anspruch auf Vollständigkeit. Für Hinweise und Anregungen bin ich jederzeit dankbar.

3 Installation der Software

Zuallererst brauchen wir die Linux-Version von ISE WebPack Diese kann unter http://www.xilinx.com nach einer einfachen Registrationsprozedur heruntergeladen werden. Ich benutze noch immer die "alte" Version 7.1i, die für den vorgegebenen Zweck völlig ausreicht und auch vom Speicherverbrauch her noch recht moderat ist. Außerdem lässt sich das Programmverzeichnis problemlos und Installationsprozedere von einem Rechner auf den nächsten kopieren. Die Datei ist etwa 242 Megabytes groß, ausgepackt belegt das Ganze auf der Festplatte etwa 670MB. Damit die Programme auch ihre Librarys finden können, muss der genaue Pfad mit den Binarys (.../Xilinx/bin/lin) in die Datei /etc/ld.so.conf eingetragen und ldconfig aufgerufen werden.
Desweiteren werden noch die Kabeltreiber benötigt. Da es unter Kernel 2.6.x Glückssche ist, den Jungo-Driver kompiliert zu bekommen und dieser auch noch propietärer Art ist, habe ich das Ganze auf den USB-DRIVER.HEAD umgestellt. Dieser wird einfach nach Anleitung kompiliert und installiert und funktioniert sowohl mit den USB- als auch mit den Parallelportkabeln.

4 Installation und Konfiguration des Scripts

Jetzt muss noch das Shell-Script vhdl2cpld z.B. nach /usr/local/bin kopiert und konfiguriert werden. Dazu muss es mit einem Texteditor geöffnet werden. Zuerst muss der Pfad zur USB-driver library festgelegt werden. Wenn der Driver selbst

kompiliert wurde, sollte der angegebene Pfad bereits stimmen. Danach wird die Variable $xilinx eingerichtet. Hier muss der Pfad angegeben werden, wohin das WebPack installiert wurde. Es folgen die Einträge $errview und $logview. hier muss angegeben werden, mit welchem Programm die Fehlermeldungen angezeigt werden sollen. Arbeitet man lieber nur auf der Konsole, dürfte ein **cat** genügen, ansonsten z.B. xemacs, bei Bedarf auch mit Parametern.

5 Eine VHDL-Datei

VHDL-Tutorials und Referenzen gibt es im Internet zuhauf, deshalb will ich nur die für die CPLD-Entwicklung notwendige Syntax und ein paar Eigenheiten des User Constraint Files (ucf) erklären. Doch zuerst einmal ein VHDL-Quellfile:

```
---Company:
---Engineer:    Joerg Wolfram
---Create Date:    10:19:45 06/09/05
---Design Name:    testlogik
---Module Name:    main1 - Behavioral
---Project Name:   Demoprojrkt
---Target Device:  XC9536-5-44PC
---Tool versions:  XILINX Webpack 7.1 (linux) + vhdl2cpld
---Description:
---Dependencies:
---Revision:
---Comments:

IEEE;
IEEE.STD_LOGIC_1164.ALL;
IEEE.STD_LOGIC_ARITH.ALL;
IEEE.STD_LOGIC_UNSIGNED.ALL;

entity testlogik is
   Port (   in1 : in std_logic_vector(1 downto 0);
         result : out std_logic);
end testlogik;

architecture program of testlogik is
begin
 process (in1)
 begin
  if (in1 = &quot;00&quot;) then
   result &lt;= '1';
  else
   result &lt;= '0';
  end if;
 end process;

end program;

configuration main of testlogik is
 for program
 end for;
end configuration main;
```

Kommentare in VHDL-Dateien beginnen immer mit zwei Minuszeichen (–) und gehen bis zum Ende der Zeile. Eine detaillierte Beschreibung findet sich im VHDL-Projekte Verzeichnis.

Eine weitere wichtige Komponente ist das User-Constraints-File mit der Endung .ucf mit dem zum Beispiel die IN und OUT-Signale zu physikalischen Pins fest zugeordnet werden können. Existiert ein derartiges File nicht, so werden die

Pins vom CPLD-Fitter willkürlich bestimmt. Das gilt auch für Signale, die nicht explizit an Pins gebunden werden.

Der Bereich zur Pin-Zuordnung steht im Beispiel zwischen den Kommentaren **#PINLOCK_BEGIN** und **#PINLOCK_END**. dazwischen stehen die Pinzuweisungen in folgender Form:

- NET ”Signalname” LOC=”PINxx”;

Nach NET steht der Signalname, wie er auch in der VHDL-Datei steht. Im zweiten Teil steht nach LOC= die Pinnummer. Um Strom zu sparen, kann man für bestimmte Ausgänge ein langsameres Timing wählen:

- NET ”Signalname” pwr_mode=low;

Oder, für hohe Geschwindigkeit (und hohen Stromverbrauch):

- NET ”Signalname” pwr_mode=std;

Diese Einstellungen haben natürlich nur für Ausgänge Sinn. Bei Register-Ausgängen kann der Zustand nach der Initialisierung (Einschalten) festgelegt werden:

- NET ”Signalname” INIT=’0’;
- NET ”Signalname” INIT=’1’;

setzt das Register auf den entsprechenden Wert.
 Mittels des ”Pipe”-Zeichens können auch mehrere Festlegungen zu einem Signal getroffen werden:

- NET ”Signalname” LOC=”PINxx” | INIT=’0’;

Die Datei **m.ucf** für unser Demo hat folgenden Inhalt:

```
#PINLOCK_BEGIN
NET ''in1<0>'' LOC=''PIN1'';
NET ''in1<1>'' LOC=''PIN2'';
NET ''result'' LOC=''PIN3'';
#PINLOCK_END
```

Jetzt geht’s ans übersetzen. Wenn als CPLD ein XC9536 im PLCC-44 Gehäuse benutzt wird, brauchen keine Optionen eingegeben werden.
vhdl2cpld -noburn
Mit der Option ”-noburn” wird verhindert, dass Impact zum Brennen des CPLD’s gestartet wird, weitere Optionen können mit **vhdl2cpld –help** angezeigt werden. Die Bedeutung der Optionen ist im Einzelnen:

–help	Anzeige der möglichen Optionen
-nogen	Die Datei ”project.jed” wird ohne vorherige Compilierung gebrannt.
-noburn	Das Projekt wird nur compiliert (ohne Impact zum Brennen)
-detail	Es wird ein detaillierter Timingreport erzeugt.
-top name	Festlegung des toplevel-Elements (optional).
-type Typ	Auswahl eines anderen CPLD-Types anstelle XC9536-5-PC44
-position num	Position des zu programmierenden CPLD's in der Kette (default 1)
-speed	XST und der CPLD-Fitter werden mit Optimierung für Geschwindigkeit aufgerufen. Für zeitkritische Designs, ohne diese Option lässt sich aber oft mehr Logik im Chip unterbringen.
-highpower	Ausgänge werden per default im Standard-Mode betrieben. Dadurch werden sie zwar schneller, aber auch dei Stromaufnahme steigt an.
-startconfig	Programmlauf ohne XST (Compiler), unm z.B. bei fehlgeschlagenem Fitten die Pins anders zu verteilen, ohne das Design neu übersetzen zu müssen.
-inputs num	Begrenzung der Anzahl der Funktionsblock-eingänge für Logikgleichungen (CPLD-Fitter-Option). Diese Option kann benutzt werden, falls der Fitt-Vorgang fehlgeschlagen ist.
-pterms num	Begrenzung der Anzahl der Produkt-Terme für Logikgleichungen (CPLD-Fitter-Option). Diese Option kann benutzt werden, falls der Fitt-Vorgang fehlgeschlagen ist.
-optlevel 1/2	Einstellung für den Optimierungs-Level. Standardmäßig wird 1 für Optimierung nach Geschwindigkeit (-speed) und 2 für Optimierung nach Fläche voreingestellt.
-ucf dateiname	Benutzt die angegebene UCF-Datei anstelle der zuerst gefundenen.
-svf	Impact erzeugt eine SVF-Datei mit dem Namen **project.svf** diese kann z.B. mit anderen JTAG-Programmern benutzt werden.
-xsvf	Impact erzeugt eine XSVF-Datei mit dem Namen **project.xsvf** diese kann z.B. mit anderen JTAG bzw. XILINX-Programmern benutzt werden.
-sim	Es werden Dateien für eine Post-Fit-Simulation erzeugt.

Für Hinweise empfehle ich xapp444.pdf ”CPLD-Fitting, Tips and Tricks”, das Dokument kann bei Xilinx heruntergeladen werden.

6 To be continued...

Der Beitrag soll zuerst einmal dazu dienen, die ersten Hürden bei der CPLD-Entwicklung mit Linux etwas kleiner zu machen. Einzelne VHDL-Projekte und ein kleines Intro in VHDL gibts bei den VHDL-Projekten.