Friedrich-Schiller-Universität Jena
Lehrstuhl für Rechnerarchitektur und -kommunikation

Praktische Übung Computertechnik

Versuch A-1:
CAE-Werkzeuge

Volker Dörsing 19. November 2008

Inhaltsverzeichnis

Einführung

Am Beispiel einer einfachen und einer komplexeren logischen Funktion führen Sie die Arbeitsschritte von der Aufgabenstellung bis hin zu einem getesteten programmierbaren Schaltkreis durch.
Die Automatisierung solcher Entwurfsprozesse schreitet immer weiter fort und besteht aus dem Entwurf, der Synthese, der Implementierung sowie der Simulation. Dieser Vorgang wird als Electronic Design Automation (EDA) zusammengefasst. Der Schwerpunkt der Aufgabe liegt auf dem Kennenlernen und dem Umgang mit Electronic Computer-Aided Engineering (CAE-) Werkzeugen. In der praktischen Übung kommen die kommerziellen Werkzeuge Synopsys Entwicklung Umgebung Version 2003.6 und die Integrated Synthesis Environment Version 6.1i von Xilinx (ISE) zum Einsatz.
Der Entwurf der Schaltungen wird manuell ausgeführt und Ihnen aus Zeitgründen vorgeschlagen. In der ersten Aufgabe werden die Arbeitsschritte im Detail durchgeführt und zur Automatisierung in Kommandodateien aufbereitet. Für die zweite Aufgaben sind die Algorithmen der komplexeren Funktion ebenfalls bereits in einer Hardware Beschreibungssprache (VHDL) für 8 Bit Datenbreite formuliert. VHDL Kenntnisse sind nur in geringem Umfang erforderlich [2,3].

Aufgabenstellung

1. Einfache logische Funktion

Simulieren, synthetisieren und implementieren Sie den vom Betreuer vorgegebenen VHDL-Entwurf für 4 Bit eines Zählers `cnt*.vhd` bzw. eines synchronen Schieberegisters `shift*.vhd`.

1. Simulieren Sie die Eingaben aus der Testumgebung. Drucken und erläutern Sie detailliert das Ergebnis sowie die Funktion der einzelnen Ein- und Ausgangssignale. Geben Sie eine eineindeutige Bezeichnung für diese Funktion an.
2. Synthetisieren Sie den Entwurf. Betrachten Sie das Syntheseergebnis auf den verschiedenen (wenn vorhanden) Hierarchieebnen. Nennen Sie die verschieden Logikzellen der Synthese (z.B. or2).
3. Implementieren Sie Ihren synthetisierten Schaltungsentwurf. Welche Komponenten (z.B. IOB) und wie viele von jeder Sorte wurden belegt. Wie hoch ist die Zahl der Gatteräquivalente. Nennen Sie von jeder Art der benutzen Komponenten beispielhaft eine Ortsbezeichnung (z.B. CLB_R23C19.S1). Welches sind die drei wichtigsten Ergebnisdateien und wofür werden sie benötigt.

Simulieren und erproben Sie den Schaltkreis.

4. Simulieren Sie die Eingaben aus der Testumgebung. Nennen sie zwei Unterschiede zur funktionalen Simulation. Drucken und erläutern Sie detailliert den Signalverlauf der zeitbehafteten Simulation. Messen Sie mit den Markierungslinien die Verzögerung zwischen der steigenden Taktflanke und der Änderung der Ausgangssignale. Nennen sie im zeitlichen Verlauf zwei Unterschiede zur funktionalen Simulation.
5. Stellen Sie kleine Befehlsfolgen in Dateien für die Teilschritte auf Betriebsystemebene und wo nötig auf der Befehlskonsole zusammen. Führen Sie die Kommandodateien aus, protokollieren, kommentieren und drucken Sie diese.
6. Konfigurieren Sie das FPGA-Experimentalsystem. Beobachten und Protokollieren Sie die Anzeige genau. Legen Sie dieselben Signalwerte, wie sie in der Testumgebung benutzt werden, an den Schaltkreis und weisen Sie die korrekte Funktion nach.

2. Binär BCD Wandlung

Modifizieren Sie die Kommandodateien aus der Aufgabe 1.5 für die komplexere Funktion: Konvertierung einer 8 Bit binär Zahl in eine BCD-Zahl. Führen Sie alle Kommandodateien bis zur Anzeige der zeitbehafteten Simulation entsprechend aus, protokollieren und diskutieren Sie sinngemäß die Aufgaben 1.1. – 1.6. Welche Zeit benötigt die Konvertierung der 8 Bit binär Zahlen.

Vorbereitung

1. Schaltungsentwicklung für einen programmierbaren Schaltkreis

Im folgenden Abschnitt werden die Arbeitsschritte von der Aufgabenstellung bis hin zu einem getesteten programmierbaren Schaltkreis dargestellt. Diese Arbeitsschritte sind der Schaltungsentwurf, die funktionale Simulation, die Synthese, die Implementierung, die zeitbehaftete Simulation und die Erprobung.

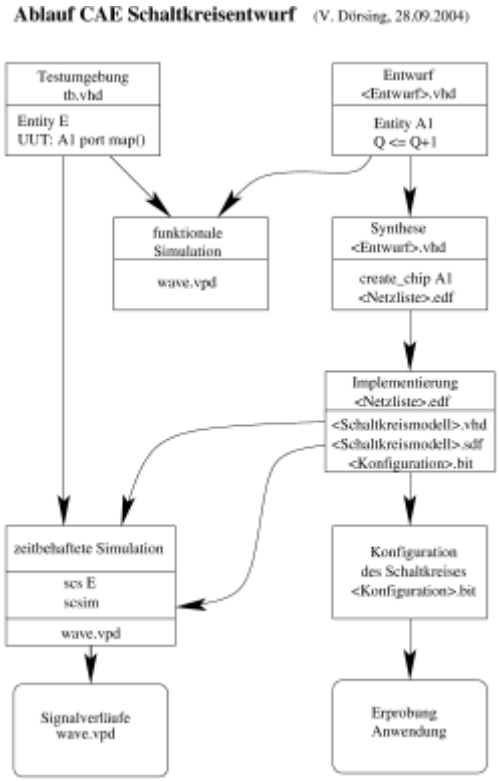
1.1. Schaltungsentwurf

Zuerst erfolgt die Umsetzung der Aufgabenstellung in einen Algorithmus und die Formulierung des gewählten Algorithmus in einer Hardwarebeschreibung, einer Schaltung oder einer Mischung aus einem Schaltplan und einer Hardwarebeschreibung.
Wir gehen von der Hardwarebeschreibung mit der Sprache VHDL aus. Diese ist der Programmiersprache ADA verwandt und ermöglicht die textuelle Beschreibung von gleichzeitig (d.h.

parallel) ablaufenden Vorgängen. Jede Funktionseinheit besteht aus einer Schnittstelle (Entity) und der eigentlichen Beschreibung (Architektur). In der Architektur kann sowohl das Verhalten als auch die Verbindung von Funktionseinheiten beschrieben werden. Verbundene Funktionseinheiten werden wiederum durch ihr Verhalten oder ihre Struktur beschrieben und so fort. Jede Strukturebene wird als Hierarchieebene bezeichnet.
Zu Beginn des Versuches erhalten Sie Ihre funktionsfähigen VHDL-Beschreibungen vom Betreuer.

1.2. Funktionale Simulation

Nach dem Entwurf einer Beschreibung, was dem Entwurf eines Schaltplans entspricht, wird die VHDL Beschreibung (Design) mit einem Simulator analysiert, was im wesentlichen eine syntaktische Prüfung beinhaltet.



Darauf folgt die Simulation der Schaltung. Damit die Aufgabenstellung überprüft werden kann, müssen Signale an die Schaltung angelegt werden, die die Schaltung entsprechend der gewünschten Funktion stimulieren. Dazu dient eine Testumgebung (testbench). Das ist ebenfalls eine VHDL-Beschreibung, welche die gewünschten Eingangssignale der Schaltung erzeugt. Sie muss ebenfalls analysiert werden und bindet die zu testende Schaltung als Funktionseinheit ein.
Die Testumgebung mit der formalen Schnittstelle (Entity) E stellt die oberste Hierarchieebene (/E) dar. In der Architektur der Testumgebung wird die Schnittstelle der zu testenden Schaltung den Signalen der Testumgebung zugeordnet. Diese Zuordnung hat den Bezeichner UUT (unit under test) und wird auch Instantiierung unserer Funktionseinheit (Komponente) genannt. Damit ist unsere Funktionseinheit als Instanz mit dem Namen UUT mit der Testumgebung verbunden. Die UUT stellt die nächst niedrigere Hierarchieebene (/E/UUT) dar. Da dieser Vorgang rekursiv erfolgen kann, entsteht eine Baumstruktur.
Im nächsten Schritt wird eine ausführbare Datei `scsim` erzeugt, die das Simulationsmodell enthält und sich auf die oberste Hierarchieebene als Ausgangspunkt und aktuellen Arbeitsbereich (/) bezieht. Mit dem Ausführen dieses Simulationsmodells wird die Simulationskonsole gestartet, welche die Testumgebung simuliert, was die Simulation der zu testenden Schaltung nach sich zieht.
Die simulierten Signalverläufe werden anschließend mit einem Visualisierungsprogramm angesehen und auf Korrektheit überprüft. Falls die Signalverläufe nicht der Aufgabe entsprechen, muss die Schaltungsbeschreibung geändert werden und der gesamte beschriebene Ablauf wird wieder durchlaufen.
Eine reale Schaltung hat Signallauf- und Verzögerungszeiten. Wenn eine funktionale Beschreibung ohne Angabe von Verzögerungszeiten gemacht wurde, können diese auch in der Simulation nicht berücksichtigt sein. Auf den Fall der zeitbehafteten Simulation wird später eingegangen.

1.3. FPGA Xilinx Virtex E

In dieser Aufgabe wird ein unbegrenzt oft programmierbarer Logikschaltkreis, ein FPGA aus der Virtex E Familie von Xilinx eingesetzt [8]. Er besteht aus der Speicher- und der Logik- Ebene. In der Speicherebene, die als SRAM ausgebildet ist, wird die Konfiguration des FPGA gespeichert. Sie steuert die Logikebene des Schaltkreises, der ein gleichförmig angeordnetes Feld von konfigurierbaren Logikblöcken (CLB - configurable logic block) enthält. Diese sind von programmierbaren Eingangs-/Ausgangsblöcken (Input/Output Blocks) umgeben. Weiterhin

grammierbaren Eingangs-/Ausgangsblöcken (Input/Output Blocks) umgeben. Weiterhin enthält der Schaltkreis 4 Takt Puffer (GBUF) für den ganzen Schaltkreis durchziehende Taktnetzwerke. Die Komponenten des Schaltkreises können durch lokale und globale Verdrahtungsnetze miteinander verbunden werden.
Jeder CLB besteht aus zwei Teilen (Slice). Ein „Slice“ besteht aus einer F und G Logikzelle, aus zwei D-Flip-Flop sowie einer Carry-Logik. Die Logikzellen sind Wahrheitswertetabellen (look-up table – LUT). Sie realisieren die kombinatorische Logik NOT, AND, OR, XOR.
Der oben beschriebene Schaltungsentwurf wird durch die Synthese und Implementierung auf diese inneren Strukturen des Schaltkreises umgesetzt, so dass die Logikzellen, Flip-Flops und anderen Ressourcen die logischen Funktionen darstellen.

1.4. Synthese

Der Syntheseprozess unterteilt sich in die Phasen Analyse (analog zur Simulation), Synthese und Export einer Netzliste für die Implementierung.
Die abstrakte textuelle Beschreibung einer Schaltung wird im Laufe der Synthese in logische Funktionen überführt. Die Grundelemente für die kombinatorische Logik sind AND, OR und NOT. Das Grundelement der sequentiellen Logik ist das Flip-Flop (SEQ). Die Konstrukte der mächtigen Programmiersprache VHDL müssen isoliert und in semantisch richtige logische Konstruktionen übersetzt werden. Dies erfordert den Einsatz von Methoden der künstlichen Intelligenz mit Datenbanken, um die komplexen Gebilde der Programmiersprachen semantisch richtig zu interpretieren.. Für einige logische und arithmetische Operatoren gibt es eine spezielle „Design Ware“ Bibliothek, die diese Operatoren in entsprechende Konstruktionen (z.B. DW01_dec_8_1) umsetzt. Weiterhin gibt es auch programmtechnische Formulierungen, die nicht sinnvoll synthetisiert werden können und deshalb von der Synthese ausgeschlossen sind.
Durch das Einbinden von technologieabhängigen Bibliotheken können die Synthesewerkzeuge für die verschiedensten Technologien eingesetzt werden. Beispiele sind ASIC Technologien oder Technologien für programmierbare Bauelemente (PLD, CPLD, FPGA),. Jede Technologie besitzt ihre eigenen besonderen Strukturelemente, Synthesezellen genannt. Die Abbildung des Schaltungsentwurfs muss auf diese speziellen Synthesezellen erfolgen. Es ist bereits in dieser Phase möglich, Vorgaben hinsichtlich Platzbedarf, Verbindungsstruktur und Zeitbedingungen für diesen Syntheseschritt zu machen.
Am Ende des Prozesses steht der Export einer Netzliste für eine bestimmte Technologie.

1.5. Implementierung

Die bei der Synthese erzeugte Netzliste mit technologiespezifischen Synthesezellen wird von dem Implementierungswerkzeug, das im Allgemeinen vom Schaltkreishersteller bereitgestellt wird, auf den konkreten ausgewählten Schaltkreis (z.B. Xilinx Virtex E 200 mit Gehäuse PQ240 und Geschwindigkeitsgrad 0.6 ns) abgebildet. Dabei müssen die synthetisierten Logikzellen den CLB des Schaltkreises zugeordnet werden. Weiterhin wird der zugeordnete Logikblock in die Zeilen und Spalten Anordnung im Schaltkreis eingefügt. Die Verbindungen zu anderen Logikblöcken und zu den Anschlüssen des Schaltkreises müssen hergestellt werden.
Dieser Vorgang ist durch Implementierungsvorgaben beeinflussbar. In einem „constraints file“ machen wir Vorgaben über die Lage der Schaltkreisanschlüsse. Vorgaben für Signallaufzeiten, Taktfrequenzen und Platzierung von Logikblöcken sind ebenfalls möglich.
Die Implementierung erfolgt in den Phasen Übersetzen, Umwandlung der Netzliste von Synthesezellen in Logikblöcke, Platzierung der Logikblöcke, Verbinden der Anschlüsse, Berechnen der Verzögerungszeiten sowie Erzeugen eines Simulationsmodells mit Zeitinformationen und einer Konfigurationsdatei.
Der Schaltkreis kann mit den grafischen Werkzeugen „Constraints Editor“, „FPGA Editor“ und „Floorplaner“ betrachtet und weiter bearbeitet werden. Wir benutzen zur Visualisierung den FPGA Editor.

Mit dem Simulationsmodell und den Zeitinformationen kann noch vor dem Einsatz in einem größeren System die Funktion des Schaltkreises in der zeitbehafteten Simulation überprüft werden. Das eigentliche Ergebnis des gesamten Prozesses ist jedoch die Konfigurationsdatei. Durch die Konfiguration, speichern der Datei in dem FPGA, erhält der Schaltkreis seine durch den Entwurf bestimmte Funktion.

1.6. Zeitbehaftete Simulation

Die zeitbehaftete Simulation erfolgt wie die funktionale Simulation und gibt Auskunft über das letztendlich korrekte zeitliche Verhalten der zu lösenden Aufgabe. In diesem Abschnitt wird nur auf Unterschiede zur funktionalen Simulation hingewiesen.
Das Simulationsmodell ist jetzt ein Modell des realen Schaltkreises. Dieses Modell setzt sich wiederum aus Simulationsmodellen der elektrischen Einheiten des Schaltkreises, u.a. den Logikblöcken, und den Verbindungen zusammen (vgl. VHDL- Hierarchieebenen).
Die Schaltkreiskomponenten-Bibliothek `SIMPRIM` beinhaltet diese Einheiten und muss in der Initialisierungsdatei `synopsys_sim.setup` für die zeitbehaftete Simulation angegeben werden. Die Elemente dieser Bibliothek werden von der funktionalen Simulation nicht benutzt. Die Zeitinformationen befinden sich nicht in dem VHDL Simulationsmodell, sondern in einer separaten Datei im „Standard Delay Format“ (SDF). Diese Datei wird dem Simulator zusammen mit der Hierarchieebenen, die sie beschreibt, angegeben.
Wie die funktionale Simulation wird die zeitbehaftete in einer Testumgebung durchgeführt, die eine größere Schaltung oder auch Anzeige- und Eingabeelemente sein kann.. Der Vorteil ist u.a., dass gezielt interessierende und kritische Zustände für die Schaltung erzeugt werden können, was in realen Schaltungen nicht immer einfach ist.
Weil wir es hier mit dem Modell eines realen Schaltkreises zu tun haben, muss die Testumgebung an diesen angepasst werden. Zum einen dürfen an einen realen Schaltkreis keine variablen Größen, wie Generics übergeben werden. Zum anderen benötigt ein Schaltkreis eine gewisse Zeit zur Initialisierung seiner inneren Zustände. Das bedeutet, dass 100 ns vergehen, bevor die angelegten Signale richtig verarbeitet werden.
Da die Verbindungen durch die Synthese künstlich erzeugt wurden, ist es meist schwierig, die Namen interessierender interner Signale zu bestimmen. Häufig bleiben nur die Signalnamen aus den Portdefinitionen erhalten, welche die Schnittstellen zwischen verschiedenen Hierarchieebenen darstellen. Solche Veränderungen in den Signalnamen müssen bei der zeitbehafteten Simulation berücksichtigt werden und erschweren sie manchmal erheblich. Gleichzeitig stellt die Simulation auch von internen Signalen ein bedeutender Vorteil gegenüber der realen Erprobung dar, denn bei der realen Testschaltung kann man nicht in den Schaltkreis „hinein sehen“.

2. Algorithmen

Als einfache logischen Funktionen werden ladbare links und rechts Schieberegister mit einer Schrittweite von 1 Bit, Zähler mit und ohne Carry sowie kaskadierbare Zähler eingesetzt.
Für die Konvertierung einer Binärzahl in eine BCD-Zahl gibt es Algorithmen, die eine unterschiedliche Anzahl von N Takten benötigen: $O(2^N)$, O(N) und O(1) [1].

2.1. Algorithmus $O(2^N)$

Diese einfachste Methode besteht darin, einen Binärzähler und einen Dezimalzähler parallel laufen zu lassen. Wenn man beide Zähler startet und die Zähler dann anhält, wenn die umzuwandelnde Dualzahl erreicht ist, steht am Ausgang des Dezimalzählers die BCD-Zahl zur Verfügung. Eine N-stellige Binärzahl benötigt zur Umwandlung 2^N Zählschritte. Für die Realisierung in Hardware ist es einfacher, den Binärzähler erst mit der Binärzahl zu laden und Umwandlung beim Zählwert 1 anzuhalten.

2.2. Algorithmus O(N)

Die effektiveren Algorithmen beruhen auf der Korrektur der Pseudotetraden. Um sie zu korrigieren, muss man die Zahl $6 = 0110_2$ addieren. Binärzahlen mit mehr als 4 Bit können von rechts nach links in einen „BCD-Rahmen“ geschoben werden. Überschreitet eine Eins die Grenze zwischen der Einer- und der Zehnergruppe, entsteht ein Fehler. Denn bei der Binärzahl nimmt der Stellenwert durch die Verschiebung von 8 auf 16 zu, während bei der BCD-Zahl nur eine Erhöhung von 8 auf 10 entsteht. Zur Korrektur muss eine 6 addiert werden. Weiterhin sind noch die Pseudotetraden zu korrigieren.

Dasselbe erreicht man auch durch die Addition einer $3 = 0011_2$ vor dem Schieben. Ob eine Korrektur notwendig ist, lässt sich bereits vor dem Schieben erkennen: Bei Werten einer Tetrade kleiner gleich $4 = 0100_2$ ist keine Korrektur nötig. Bei Werten von 5, 6 oder 7 entstehen Pseudotetraden aber keine Grenzüberschreitung. Bei Werten von 8 oder 9 muss die Grenzüberschreitung korrigiert werden. Werte größer 9 können wegen der Pseudotetraden-Korrektur nicht auftreten. Daraus resultiert die Korrekturtabelle.

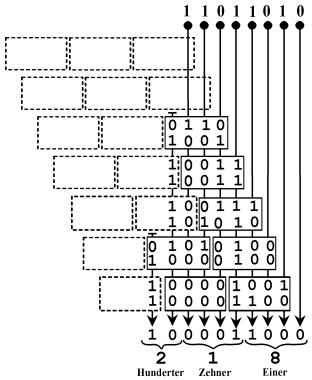
Dezimal	Eingang				Ausgang				Funktion
X	x3	x2	x1	x0	y3	y2	y1	y0	Y
0	0	0	0	0	0	0	0	0	X
1	0	0	0	1	0	0	0	1	X
2	0	0	1	0	0	0	1	0	X
3	0	0	1	1	0	0	1	1	X
4	0	1	0	0	0	1	0	0	X
5	0	1	0	1	1	0	0	0	X+3
6	0	1	1	0	1	0	0	1	X+3
7	0	1	1	1	1	0	1	0	X+3
8	1	0	0	0	1	0	1	1	X+3
9	1	0	0	1	1	1	0	0	X+3

Schaltungstechnisch ist dieser Algorithmus dadurch realisierbar, dass die Dualzahl von rechts in ein Schieberegister geschoben wird, dass in 4 Bit-Blöcke (Dekaden) unterteilt ist. An jeder Dekade wird ein Korrekturnetzwerk angeschlossen, das den Registerinhalt entsprechend der obigen Wahrheitstabelle verändert.

Für eine Umwandlung sind N Takte für das Schieberegister notwendig.

2.3. Algorithmus O(1)

Der sequentielle O(N) Algorithmus kann weiter beschleunigt werden, indem die Schiebeoperation durch eine feste kombinatorische Verdrahtung ersetzt wird. Statt die Zahlen von rechts nach links zu schieben, wird der dekadische Rahmen (4 Bit) von links nach rechts geschoben und die selbe Korrektur wie oben ausgeführt. Um die Verschiebung der Rahmen feste verdrahten zu können, benötigt man pro Dekade und Schiebeschritt je ein Korrekturnetzwerk. Korrekturnetzwerke, an die weniger als 3 Bit angeschlossen werden, können (logischer Weise) weggelassen werden.



Da alle Operationen kombinatorisch sind, ist die Zeitdauer für die Umwandlung gleich der Summe aller Verzögerungszeiten auf dem längsten kombinatorischen Pfad. Wenn diese Funktion in einen größeren getakteten Schaltungsentwurf eingebunden ist, ergibt sich die Taktperiode des getakteten Schaltungsentwurfs aus dieser Verzögerungszeit.

Die Komplexität des Algorithmus ist mit O(1) nicht korrekt angegeben, weil weiterhin N Operationen notwendig sind. Die Angabe O(1) ist in dem Sinne gemeint, dass das Ergebnis, wie oben erläutert, schon nach einem Takt vorliegt.

Durchführung

An einem PC mit einem „X Window System“ erhalten Sie ein Terminal Fenster zu einem Unix-Server in einer Solaris 2.8 Unix-Umgebung. Vor der Benutzung muss die Betriebssystemumgebung auf die CAE-Werkzeuge eingestellt werden. Bitte haben Sie etwas Geduld, wenn ein Fenster nicht sofort auf dem Bildschirm erscheint. Sie arbeiten in einer Client-Server Umgebung mit „Security Shell“, deren Reaktionszeit größer als am heimischen PC ist.

Die Beschreibung verwendet beispielhaft Dateinamen und geht von dem Startverzeichnis der jeweiligen Aufgabe aus. In diesem Verzeichnis befinden sich auch alle Dateien, die Sie zur Verfügung gestellt bekommen. Das sind:

```
<Entwurf>.vhd          # der VHDL Entwurf
tb.vhd                 # die Testumgebung
pin.ucf                # die Anschlusszuordnung für den FPGA (constraints file)
synopsys_dc.setup      # Initialisierungsdatei für Synopsys Synthese
synopsys_sim.setup     # Initialisierungsdatei für Synopsys Simulation
sc.cmd                 # Kommandodatei für Simulator scsim
wave.cfg               # Datei für die Konfiguration des Betrachters für Signalverläufe
```

Diese Dateien sind einfache Textdateien und können mit jedem Texteditor (`emacs` , `joe, pico, nedit (nur Server), vi, vim o.a.`) bearbeitet werden.
Folgende Verzeichnisstruktur sollte im Laufe der Bearbeitung für die Aufgabe 1 entstehen, Aufgabe 2 entsprechend:

~/a1/aufg1:	Quelldateien (vgl. oben)
fsim:	funktionelle Simulation
dc:	Synthese
xilinx:	Implementation
tsim:	Zeitsimulation

Ein lokales Fenster dient zum Test mit dem Experimentalsystem [7]. Hier stehen die Befehle `konf` und `tst` zur Verfügung (vgl 5. Erprobung).
Ihre Druckaufträge werden normalerweise an den Drucker `"npr2"` im Übungsraum geschickt. Textdateien können mit dem Befehl `"lp <Dateiname>"` gedruckt werden.
Am Ende des Versuches melden Sie sich bitte an Ihrem Arbeitsplatz ab.

! Jeder Schritt im Entwurfsfluss wird in einem eigenen Verzeichnis ausgeführt !

1. Funktionale Simulation

Zuerst wird das Verzeichnis für die funktionale Simulation (z.B. `fsim`) angelegt und vorher gelöscht, falls es schon bei einer vorherige Simulation erzeugt wurde:

```
rm –r fsim
mkdir fsim
```

In diesem Verzeichnis werden die weiteren Kommandos ausgeführt. Die Initialisierungsdatei für die Simulationsumbebung `synopsys_sim.setup` muss hierhin kopiert und das Verzeichnis für Arbeitsbibliothek `"work"` angelegt werden:

! Jeder Schritt im Entwurfsfluss wird in einem eigenen Verzeichnis ausgeführt !

```
cd fsim
cp ../synopsys_sim.setup synopsys_sim.setup
mkdir work
```

Der erste Schritt ist die Analyse des VHDL Entwurfs und der Testumgebung mit dem Programm „`vhdlan`“ durch die Kommandos:

```
vhdlan ../<Entwurf>.vhd
vhdlan ../tb.vhd
```

Danach wird eine ausführbare Datei `scsim` erzeugt, die das Simulationsmodell enthält. Argument für diesen Befehl ist der Name der Entity der obersten Hierarchieebene (hier die Testumgebung E):

```
scs E
```

Der Start des Simulators erfolgt durch Aufruf des vorher erzeugten Simulationsmodells `scsim` zusammen mit einer Kommandodatei, die gleich ausgeführt wird:

```
scsim –i[nclude] ../sc.cmd
```

Sie gelangen in den aktuellen Arbeitsbereich der Simulatorkonsole.

! Diese Simulationskonsole nicht verwechseln mit der Konsole des Betriebssystems !

In der Simulationskonsole stehen die sehr mächtigen Kommandos der „Scirocco Control Language“ (SCL) sowie der „Tool Command Language“ (Tcl) zur Verfügung [4]. Die Simulationskonsole ist sehr gut geeignet für die Abarbeitung von Kommandofolgen. Sie ist nicht für die interaktive Arbeit ausgelegt. Deshalb ist eine übliche Arbeitsweise, in einem Editor die Datei mit den Kommandofolgen zu bearbeiten und in dem Fenster des Servers die Kommandodatei in der Konsole auszuführen.
In diesem Versuch werden nur sehr wenige Kommandos benötigt:

- `cd, pwd` Kommandos zum Wechseln und zur Anzeige des aktuellen Arbeitsbereiches
- `ls [-t|-v]` zur Anzeige von Objekten und Arbeitsbereichen sowie deren Typ und Wert, es können Platzhalter (`*,?`) und Selektoren (`'signal, 'port`) verwendet werden
- `dump [-o wave.vpd] <Signalname>` speichert Signalverläufe in einer Datei, der Schalter `–o` kann nur beim ersten Aufruf benutzt werden, auch hier sind Platzhalter und Selektoren möglich
- `run` <Laufzeit> simuliert die angegebene Zeit, ohne Zeitangabe bis zur Tastenkombination Steuerung und c
- `quit` beendet Simulator
- `restart` startet die Simulationskonsole neu
- # Kommentar

Zur Orientierung können Sie die Kommandos `pwd, ls -t, cd` und andere Kommandos (vgl. oben 1.2 Funktionale Simulation und Beispiele weiter unten) verwenden. In diesem Syntax wurden für den Simulator in der Datei `../sc.cmd` einige Kommandos als Alias vereinbart sowie zur besseren Beobachtung der Simulation ein „Monitor“ definiert. Dieser Monitor macht Ausgaben, wenn das angegebene Signal seinen Wert wechselt (event).

In der Simulatorkonsole sind für die funktionale Simulation folgende Arbeiten auszuführen:

1. Wechseln in den uns interessierenden Arbeitsbereich, der „Unit under Test“ (UUT) in der Testumgebung

   ```
   cd /e/uut
   ```

2. Den aktuellen Wert (Value) von Signalen kann man sich anzeigen lassen.

   ```
   ls –v *’sig
   ```

3. Für das spätere Ansehen der Simulationsergebnisse werden die Signalverläufe aller Anschlüsse und interessierenden Signale mit dem Befehl `dump` in eine Datei (`-o wave.vpd`) geleitet. Es ist ratsam diesen Dateinamen zu verwenden, weil die Verbindung zu dem Betrachter für die Signalverläufe (vgl. `wave.cfg`) über diesen Namen hergestellt wird. Die Option `-o` darf nur beim ersten mal angegeben werden.

   ```
   dump –o wave.vpd *’sig
   ```

4. Danach erfolgt der eigentliche Simulationslauf. Die Stimuli für die Anschlüsse stehen in der Testumgebung `tb.vhd` und können bei Bedarf dort geändert werden. Am Ende der in der Testumgebung stimulierten Signalverläufe wird auf der Konsole eine entsprechende Ausgabe gemacht..

```
run 420 ns
ls –v *’sig
...
```

Die Simulationszeit muss der Aufgabenstellung angepasst werden und ist oben nur beispielhaft mit 420 ns angegeben worden. Das Ende der Testumbebung wird durch eine Ausgabe angezeigt.

5. Der Simulator wird mit dem Kommando `quit` verlassen

```
q
```

Diese in der Simulationskonsole ausgeführten Kommandos sollen in der Datei `../sc.cmd` ergänzt werden. Diese Kommandodatei wird in der Betriebsystemkonsole mit dem Befehl

```
scsim –i ../sc.cmd
```

zusammen mit dem Simulator (vgl. oben `scsim`) ausgeführt.
Nun können die Simulationsergebnisse in dem Signalbetrachterfenster angesehen werden.

```
scirocco +cfgfile+../wave.cfg &
```

Die Konfigurationsdatei `wave.cfg` fügt lediglich Signalverläufe ein und beschreibt das Betrachterfenster.
Über den Menüpunkt `File – Print` können Sie aussagekräftige Signalverläufe ausdrucken. Wichtig ist, dass alle Signalwerte im Ausdruck gut zu sehen sind. Folgende Einstellungen dienen zur Orientierung:

- Begin und End Time: 0 und 420,000
- Time Slices: 1
- Standard Sheet Size: A4
- Signal Names: Full Name
- Shrink To Fit

2. Synthese

Für die Synthese wird Design Vision [6] von Synopsys verwendet. Auch hier wird zuerst die Verzeichnisstruktur ähnlich der Verzeichnisstruktur für die Simulation hergestellt. Alte Verzeichnise sollte vor einer neuen Synthese gelöscht werden.

```
rm -rf dc
mkdir dc
cd dc
cp ../synopsys_dc.setup .synopsys_dc.setup
# ! Achtung: Das erste Zeichen der Zieldatei ist ein Punkt !
mkdir work
design_vision
```

In der Konsole von Design Vision wird nun die Synthese ausgeführt. Die Datei **.**synopsys_dc.setup ist die Konfigurationsdatei für Design Vision und enthält u.a. Zuweisungen der Synthesebibliotheken.

! Die design_vision-Konsole ist nicht zu verwechseln mit der Konsole des Betriebssystems !

Sie stellt ein Tcl basiertes Kommandozeileninterface für die Synthesekommandos des Design Compiler [5] zur Verfügung. Die dv-Konsole ist gut geeignet für die Abarbeitung von Kommandofolgen. Dafür werden die Befehlsfolgen in eine Datei geschrieben. Es ist eine übliche Arbeitsweise, in einem Fenster die Datei mit den Kommandofolgen in einem Editor zu bearbeiten und in dem Fenster des Servers die Kommandodatei in der Konsole auszuführen.
Dafür kann diese Datei beim Aufruf von Design Vision angegeben werden.

```
design_vision –f <script>
```

Die Synthese besteht aus folgenden Teilschritten:

1. Analyse der synthesefähigen VHDL Quelldatei `<Entwurf>.vhd` und erzeugen eines technologieunabhängigen Entwurfs

   ```
   analyze -format vhdl <Entwurf>.vhd
   elaborate <Entity Name>
   ```

   Die Synthese geht von dem VHDL Entwurf aus. Deshalb wird der Name der Entity aus der Datei `<Entwurf>.vhd` angegeben. Achten Sie bitte auf Groß- und Kleinschreibung.
2. Vorbereitung des Entwurfs für die Synthese: In dem komplexeren Entwurf für die 2. Aufgabe gibt es Teilentwürfe, die mehrfach benutzt werden (z.B. Korrekturnetzwerk), Eingänge, die zur Anzeige mit Ausgängen verbunden sind, und Ausgänge, denen ein fester Pegel zugewiesen wird. Diese Besonderheiten müssen aufgelöst werden.

   ```
   uniquify
   set_fix_multiple_port_nets –feedthroughs –buffer_constants
   compile
   ```

3. Einsetzen von Anschlüssen, die aus dem Schaltkreis herausgeführt werden

   ```
   set_port_is_pad "*"
   insert_pads
   ```

4. Die eigentliche Synthese

   ```
   compile
   ```

5. Erzeugen der EDIF Netzliste

   ```
   set edifout_design_name <Entity Name>
   write -hierarchy -format edif -output <Netzlisten Name>.edf
   ```

6. Um die synthetisierten Netzlisten betrachten zu können, muss die grafische Oberfläche gestartet werden (dauert einige Zeit). Zum Abschluss wird die dv-Konsole beendet.

   ```
   gui_start
   quit
   ```

   Sie können in der grafischen Oberfläche von Design Vision über das Menü `Schematic – New Design Schematic View` die synthetisierte Schaltung des Schaltkreises ansehen. Durch Vergrößern der Fenster, durch Zoomen und wenn Sie die Maus über den Elementen der Schaltung positionieren, können Sie die Bezeichnungen der Elemente und Verbindungen lesen. Wenn Sie ein zusammengesetztes Element (z.B. B_Zaehler) aktivieren, können Sie über das Menü `Schematic – Move down` bzw. Move `up` in dieses Element „hinein schauen“.
   Der Ausdruck ist über das Menü `File – Print Schematik...` möglich. Folgende Einstellungen sollten Sie machen
   - Print to printer: npr2 isun01
   - Paper format: Landscape; A4 (210x297 mm, 8.26x11.7 inches)

3. Implementierung

Die Erzeugen des Schaltkreises mit einem Werkzeug des Herstellers Xilinx muss wieder in einem eigenen leeren Verzeichnis geschehen.

! Jeder Schritt im Entwurfsfluss wird in einem eigenen Verzeichnis ausgeführt !

```
rm –rf xilinx
mkdir xilinx
```

```
cd xilinx
```

Es gibt folgende Arbeitsschritte:

1. Netzlisten und Vorgaben in das „Xilinx Native Generic Database“ Format überführen

```
ngdbuild -l synopsys -uc ../pin.ucf –p v200epq240-6 \
 ../dc/<Netzlisten Name>.edf <mein Schaltkreis>.ngd
```

2. Zuordnen der Synthese-Elemente zu den FPGA Komponenten. In den Ausgaben auf die Konsole werden auch Angaben über die belegten Komponenten und die Anzahl der Gatteräquivalente (Total equivalent gate count) gemacht. Die Protokolldatei ist `map.mrp`.

```
map -o map.ncd <mein Schaltkreis>.ngd  \
     <mein Schaltkreis>.pcf
```

3. Platzieren der FPGA Komponenten und festlegen der Verbindungswege. Die Protokolldatei ist `<mein Schaltkreis>.par`.

```
par -w map.ncd <mein Schaltkreis>.ncd  \
     <mein Schaltkreis>.pcf
```

4. erzeugen des VHDL Simulationsmodells

```
netgen –w -ofmt vhdl -sim <mein Schaltkreis>.ncd \
     <mein Schaltkreismodell>.vhd
```

5. erzeugen der Konfigurationsdatei

```
bitgen -w <mein Schaltkreis>.ncd <meine Konfiguration>.bit \
     <mein Schaltkreis>.pcf
```

Mit dem FPGA Editor kann man sich die Anordnung und das Innenleben des erzeugten Schaltkreises ansehen.

```
fpga_editor <mein Schaltkreis>.ncd &
```

In dem Fenster “List1” sind alle Komponenten mit Namen, Lage und Typ aufgeführt. Wenn eine Komponente z.B. vom Type „SLICE“ aktiviert wird, kann in dem Fenster „World1“ die Lage dieser Komponente (z.B. CLB_R12C3,S1) als roter Punkt beobachtet werden. Ebenso können Sie sich auch z.B. Verbindungen anzeigen lassen. Mit dem Menübefehl „`View – Zoom Selection`“ und anderen Befehlen zur Vergrößerung kann die ausgewählte Komponente in dem Fenster „Array1“ genauer betrachtet werden.
Mit dem auf der rechten Seite befindlichen Werkzeug „`editblock`“ werden die Struktur und die Verbindungen innerhalb der aktivierten Komponente schematisch dargestellt. Sie können sich die realisierte logische Funktion z.B. des Funktionselementes „LUT“ anzeigen lassen, indem sie das Element aktivieren oder die Maus auf dem Element positionieren.
Ausdrucken der schematischen Darstellungen ist nicht notwendig.

4. Zeitbehaftete Simulation

Die zeitbehaftete ist ähnlich der funktionalen Simulation und hat wenige wesentliche Unterschiede. Der Ablauf ist genauso wie bei der funktionalen Simulation (s. 1.), natürlich in einem eigenen Verzeichnis (z.B. `tsim`). Im Folgenden wird nur auf die Unterschiede hingewiesen.

! Jeder Schritt im Entwurfsfluss wird in einem eigenen Verzeichnis ausgeführt !

Ein wesentlicher Unterschied zur funktionalen Simulation ist, dass bei der zeitbehafteten Simulation nicht mehr der VHDL Entwurf analysiert wird, sondern das VHDL Simulationsmodell des synthetisierten und implementierten Schaltkreises (vgl. Vorbereitung 1.6).

```
vhdlan ../xilinx/<mein Schaltkreismodel>.vhd
vhdlan ../tb.vhd
```

Die Testumgebung bleibt die selbe. An der Stelle des Entwurfs wird das Simulationsmodell des Schaltkreises als „Unit under Test“ (UUT) eingebunden. Wegen der Namensgleichheit braucht die Testumgebung nicht mal geändert werden.
Die Erzeugung der Datei, in der das ausführbare Simulationsmodell enthalten ist, erfolgt wie bei der funktionalen Simulation.
Ein zweiter wesentlicher Unterschied zur funktionalen Simulation ist, dass die Zeitinformationen für den Schaltkreis dem Simulator mitgeteilt werden müssen. Sie wurden während der Implementierung, beim Erzeugen des VHDL Schaltkreismodells (vgl. Vorbereitung 1.6) in die Datei `<mein Schaltkreismodell>.sdf` im Standard Delay Format (SDF) geschrieben. Durch eine Unverträglichkeit der SDF-Versionen des Synthesewerkzeugs von Xilinx und des Simulators ist mit dem Befehl

```
sdf xilinx/<mein Schaltkreismodell>.sdf
```

eine Anpassung vorzunehmen.
Weiterhin muss angegeben werden, auf welche Hierarchieebene sich die Zeitinformationen beziehen und ob die typischen, minimalen oder maximalen Verzögerungszeiten benutzt werden sollen. In diesem Fall beziehen sich die Zeitinformationen auf das Simulationsmodell des Schaltkreises, das als Komponente UUT in die Testumgebung mit der Entity E eingebunden ist.
Die Kommandozeile lautet z.B.:

```
scsim –i ../sc.cmd \
      –sdf typ:/E/UUT:../xilinx/<mein Schaltkreismodell>.sdf
```

5. Erprobung

Der Experimentalaufbau [7] besteht im wesentlichen aus einem FPGA und einer Anzeige. Er wird über einen Mikrokontroller an den lokalen PC angeschlossen. Die Testprogramme werden auf dem PC an dem Sie arbeiten ausgeführt und nicht in dem Fenster vom Lehrstuhlserver.
Das Programm

```
konf xilinx/<meine Konfiguration>.bit
```

konfiguriert den FPGA des Experimentalsystems mit der Konfigurationsdatei der Schaltung.
Die Konfigurationsdaten befinden sich in dem Verzeichnis für die Implementierung.
Mit dem Programm

```
tst
```

werden Signalwerte an die Eingabeanschlüsse des FPGA gelegt. Die Anschlüsse unterscheiden sich in Abhängigkeit von Ihrer Schaltung. Für die Signale Takt (CLK), Start (S) bzw. Freigabe (EN) und die binäre Zahl (BIN) können Eingaben gemacht werden. Der Wert für das Signal BIN kann durch den Befehl b in hexadezimalem Format mit führendem 0x (z.B. 0xfe), in oktalem mit führender Null (z.B. 0376) oder in dezimalem Format (z.B. 254) eingegeben werden. Nur in der Aufgabe 2 wird Ihnen die Eingabe der binären Zahl in hexadezimalem Format angezeigt und an den FPGA ausgegeben. Folgende Befehle stehen zur Verfügung:

t	- eine Taktperiode
#	- Anzahl der Taktperioden
p	- Anzahl der Taktperioden von 0,5 s
b	- Eingabe von BIN: 0x??
e	- Freigabe bzw. Freigabe beenden
s	- Startsignal an und aus, sowie eine Taktperiode
Eingabetaste	- wiederholt letztes Kommando
h	- Hilfe
q	- beendet Programm

Die Anzeige ist ebenfalls abhängig von Ihrer Schaltung. Von den einfachen Schaltungen aus der Aufgabe 1 werden nur die Ausgabesignale angezeigt. Die Anzeige der komplexeren Funktion aus Aufgabe 2 erfordert für die einzelnen Algorithmen nicht alle Signale und somit kann Ihre Anzeige sinngemäß von der unten beschriebenen abweichen.

Die Ausgabe der Eingabesignale erfolgt nur in der Aufgabe 2 über die untere Anzeige mit zwei Ziffern. Folgende Zuordnung wurde von rechts (1. Ziffer) nach links (2. Ziffer) gemacht:

1. BINo(3..0) – untere binäre Ziffer
   CLKr – Taktanzeige
2. BINo(7..4) – obere binäre Ziffer
   So – Anzeige des Startsignals

Die Anzeige der Ausgabesignale erfolgen über die Anzeige mit acht Ziffern. Welche Ausgabesignale Z zugeordnet werden, hängt von der Funktion ab. Die 1. Ziffer wurde rechts und die 3. Ziffer als dritte von rechts angeordnet:

1. Z(3..0) – untere Ziffer
   ENo – Anzeige Umwandlung aktiv
2. Z(7..4) – mittlere Ziffer
   Z0o – Anzeige Ende der Umwandlung
3. Z(11..8) – obere Ziffer

Literatur

1. Halbleiter-Schaltungstechnik, Ulrich Tietze, Christoph Schenk, Berlin, Springer 1993, S. 595
2. Schaltungsdesign mit VHDL: Synthese, Simulation und Dokumentation digitaler Schaltungen; Gunther Lehman, Bernhard Wunder, Manfred Selz; Poing, Franzis 1994
3. Das VHDL-Informationsmedium der FH Köln: VHDL-easy http://www.nt-nv.fh-koeln.de/Labor/VhdlEasy/index.html
4. VHDL Simulation, User Guide, Version 2002.12, Synopsys. http://www2.informatik.uni-jena.de/~ct_mgr/a1/pdf/VHDLsimUG.pdf
5. Design Compiler, User Guide, Version U-2003.06, Synopsys; http://www2.informatik.uni-jena.de/~ct_mgr/a1/pdf/dcug.pdf
6. Design Vision, User Guide, Version U-2003.03, Synopsys; http://www2.informatik.uni-jena.de/~ct_mgr/a1/pdf/dvug.pdf
7. Reinsch, A.: FPGA-Experimentalsystem - Anwenderdokumentation. Unterlagen zum Praktikum Computertechnik, FSU Jena, Institut für Informatik, LS Rechnerarchitektur, Juli 2002. http://www2.informatik.uni-jena.de/~ct_mgr/fpgasystem/board_docu.pdf
8. Xilinx Virtex-E 1.8V Field Programmable Gate Arrays, v2.2 - Preliminary Product Specification , November 2001. http://www2.informatik.uni-jena.de/~ct_mgr/a1/pdf/ds022-2.pdf

Abkürzungen und Vereinbarungen

- CLB - Configurable Logic Block, Logikzellen der FPGA von Xilinx
- EDIF Electronic Design Interchange Format, entwickelt von Electronic Industries Alliance (EIA), standardisiert als IEC 61690-1+2, European Standards EN 61690-1+2
- FPGA Field Programmable Gate Array, Programmierbarer Schaltkreis
- LUT – Look-up table, Wahrheitswerte Tabelle
- PLD – Programmable Logic Device, CPLD – Complex Programmable Logic Device
- SCL – Scirocco Control Language [4]
- SDF – Standard Delay Format, Zeitinformationen für das Simulationsmodell des implementierten Schaltkreises
- Tcl – Tool Command Language 8.2 [4]
- UUT – unit under test – Bezeichnung für die Zuordnung der Schnittstellen Signale der zu testende Schaltung zu den lokalen Signalen (Instanz)
- VHDL – VHSIC (Very High Speed Integrated Circuit) Hardware Description Language, Standardisiert durch IEEE 1076
- `cmd` Befehl auf der Befehlszeile; die Befehlszeile gehört entsprechend der Situation zur Betriebsystemkonsole, zum Simulator oder zum Synthesewerkzeug