Laborversuch 2

VHDL-Strukturbeschreibungen / Logiksynthese

1 Ziel des Laborversuchs

Neben der Einarbeitung in die Entwurfssoftware Vivado stand im letzten Laborversuch die Modellierung des Verhaltens digitaler Schaltungen mit Hilfe von RTL-Blockschaltbildern sowie deren Umsetzung in VHDL im Vordergrund. In diesem Laborversuch werden Sie eine Möglichkeit zur Modellierung der Struktur – also des hierarchischen Aufbaus von VHDL-Modellen – kennenlernen.

Ein wesentlicher Teil des Entwurfsprozesses bei der Entwicklung digitaler Schaltungen für programmierbare Logikbausteine ist die Logiksynthese und die Implementierung, also die Abbildung des VHDL-Modells auf die Hardwarekomponenten, die der jeweilige Baustein in seinem Inneren zur Verfügung stellt.

Die Synthese läuft in Vivado weitgehend automatisiert ab: Sie geben den VHDL-Code ein, definieren die Randbedingungen (*Constraints*) für Timing und Anschlusspins, starten Synthese und Implementierung, und kurze Zeit später wird eine Programmierdatei (*Bitfile*, bzw. *Bitstream*) erzeugt, mit der Sie den Baustein konfigurieren können. War im Vorfeld die Simulation und nun auch noch die Synthese erfolgreich (d.h. ohne Fehlermeldungen), verfällt man leicht der Annahme, die programmierte Hardware funktioniere ebenfalls wunschgemäß. Dass dies nicht immer der Fall ist, werden Sie in diesem Laborversuch anhand eines Beispiels feststellen und dabei auch Werkzeuge kennenlernen, mit deren Hilfe Sie die Syntheseergebnisse interpretieren und zur Beseitigung von Fehlern im VHDL-Code nutzen können.

2 Einleitung

2.1 VHDL-Strukturbeschreibungen

Mit VHDL lässt sich nicht nur das funktionale und zeitliche Verhalten digitaler Schaltungen modellieren, sondern auch deren Struktur. Aus der Vorlesung ist Ihnen bereits bekannt, wie durch Deklaration und Instanziierung von VHDL-Komponenten (Stichwort: *component / port map*) hierarchische Strukturen aufgebaut werden können. Eine praktische Anleitung dazu finden Sie im zweiten Teil des Vivado-Einführungskurses.

Die grafische Darstellung einer Strukturbeschreibung in Form eines Blockschaltbilds sieht einem RTL-Blockschaltbild sehr ähnlich:

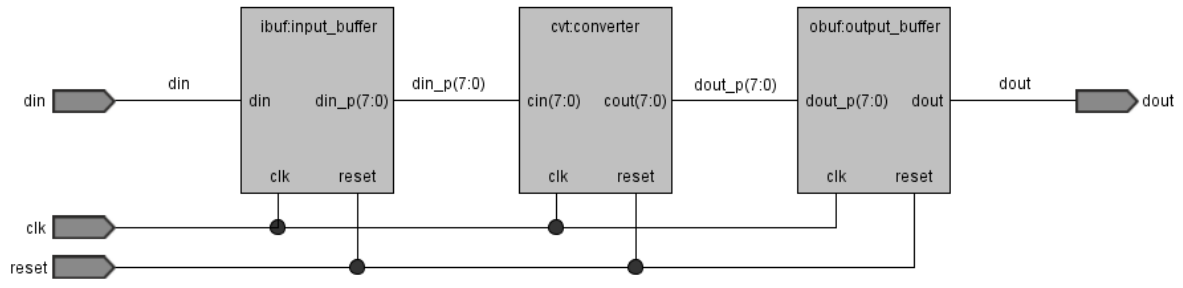

Abbildung 1: Struktur-Blockschaltbild

Hier noch einmal zum Vergleich das RTL-Blockschaltbild zur Verhaltensbeschreibung des Zählers aus dem ersten Teil des Vivado-Einführungskurses:

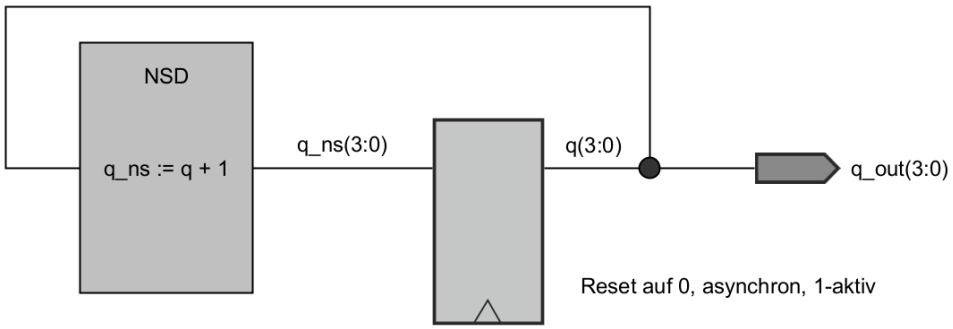

Abbildung 2: RTL-Blockschaltbild eines 4-Bit-Zählers

Der wesentliche Unterschied besteht darin, dass in einer **Strukturbeschreibung** mehrere Komponenten über ihre Schnittstellen (*entity*) miteinander verbunden werden, während die Blöcke in einem **RTL-Blockschaltbild** Prozesse repräsentieren, hinter denen sich eine **Verhaltensbeschreibung** verbirgt. Mit Strukturbeschreibungen wird ein VHDL-Modell also hierarchisch untergliedert, was insbesondere bei komplexen Schaltungen die Übersichtlichkeit gewährleistet.

Es ist wichtig, dass Sie diese beiden Beschreibungsmethoden unterscheiden können (Unterschiede sind kursiv dargestellt):

Strukturbeschreibung	**Verhaltensbeschreibung**
Beschreibt die *Struktur* einer Schaltung, die *Verbindung* von Komponenten	Beschreibt das *Verhalten* einer einzelnen Komponente
Enthält: • *Komponentendeklarationen* • *Instanziierungen von Komponenten* • Signale • Evtl. Typ- und Konstantendeklarationen	Enthält: • *Prozesse* • Signale • Evtl. Typ- und Konstantendeklarationen
Grafische Darstellung mit *Struktur-Blockschaltbild*, bestehend aus: • Blöcken • Signalen mit Bezeichner • Ein- und Ausgangsports mit Bezeichner • evtl. Legende für zusätzliche Informationen	Grafische Darstellung mit *RTL-Blockschaltbild*, bestehend aus: • Blöcken • Signalen mit Bezeichner • Ein- und Ausgangsports mit Bezeichner • evtl. Legende für zusätzliche Informationen

Strukturbeschreibung	**Verhaltensbeschreibung**
Blöcke im Struktur-Blockschaltbild repräsentieren *Instanzen* von VHDL-Komponenten	Blöcke im RTL-Blockschaltbild repräsentieren *Prozesse:* • Sequentiell: Register/Flip-Flops • Kombinatorisch: Schaltnetze • Gemischt sequentiell/kombinatorisch: z.B. Schaltwerke, Zähler, Schieberegister
Blöcke im Struktur-Blockschaltbild enthalten • *Bezeichner (Instanzname:Komponentenname)* • *Ports*	Blöcke im RTL-Blockschaltbild enthalten • bei Registern - ein *Dreieck*, das die Polarität des Taktes kennzeichnet - evtl. ein ‚*R*' für den asynchronen Reset-Eingang • bei Schaltnetzen - einen *Bezeichner*, z.B. *NSD* = Next State Decoder / Überführungsschaltnetz, *OD* = Output Decoder / Ausgangs-Dekoder, u.U. auch das *Prozesslabel* - eine *vollständige Beschreibung der Schaltfunktion*, z.B. Pseudocode, Wahrheitstabelle, evtl. aus Platzgründen gesondert dokumentiert • bei Schaltwerken, Zählern, Schieberegistern - einen *Bezeichner* z.B. *FSM* - *gesondertes Zustandsdiagramm* bei Schaltwerken

2.2 Entwurfsablauf

In den ersten beiden Laborversuchen werden Sie durch die Vivado-Einführungskurse Schritt für Schritt von der Erstellung eines Projekts bis zum Test auf der Laborhardware geführt. Vivado ist jedoch nur eines von vielen Werkzeugen am Markt. Wichtig ist daher, dass Sie die grundlegenden Schritte des Entwurfs digitaler Schaltungen mit Hardwarebeschreibungssprachen verinnerlichen. Die nachfolgende Tabelle verschafft Ihnen einen Überblick über alle wesentlichen Punkte:

	Entwurfsschritte	**…und was Sie dabei beachten sollten**
1	Erstellung eines Konzepts • Funktionalität beschreiben • Schnittstellen definieren • Umsetzung in Hardware klären • Testbarkeit prüfen / Teststrategie festlegen • Entwurf partitionieren • Dokumentation erstellen (z.B. Blockschaltbilder)	• An dieser Stelle sind die fachliche Kompetenz und die Kreativität des Ingenieurs / der Ingenieurin besonders gefordert – der Rest ist größtenteils Handwerk • Eignen Sie sich eine effiziente Methodik an (Entwurfsschritte / Werkzeuge / Dokumente / Absprachen…) • Beachten Sie den Kontext (System / Schnittstellen / Betroffene anderer Bereiche…) • Halten Sie Standards ein • Seien Sie selbstkritisch • Erweitern Sie Ihren Horizont

2	VHDL-Eingabe, Syntaxprüfung	• Eindeutige Bezeichner verwenden, die einen Bezug zur Funktion des Signals oder Prozesses herstellen, z.B. `data_in, select, state, state_next`
		• Code sinnvoll einrücken, z.B.:
		`if select = ´0´ then`
		`  d_out <= d_in0;`
		`else`
		`  d_out <= d_in1;`
		`end if;`
		• Kombinatorische und sequentielle VHDL-Prozesse trennen, z.B. **je einen Prozess** für Register/Flip-Flop, Überführungsschaltnetz und Ausgabeschaltnetz
		• Möglichst alle Register mit einem einzigen globalen Systemtakt betreiben; Ausgangssignale mit niedrigerer Taktfrequenz über *Enable*-Signale realisieren (siehe Laborversuch 3)

	Entwurfsschritte	**…und was Sie dabei beachten sollten**
3	Testbench erstellen, Simulation	• Steuerung des zeitlichen Ablaufs in der Testbench über `wait`-Statements, nicht über Sensitivitätslisten • Testbench sollte Realität möglichst gut nachbilden, z.B. Taktfrequenz, zeitliche Abläufe, Initialwerte von Signalen, Kombinationen von Eingangssignalen, … • Darstellung im Simulator auf die wesentlichen Signale beschränken, Reihenfolge der Signale dem Datenfluss entsprechend wählen, ggf. interne Signale hinzufügen, bei Bussen evtl. Darstellungsformat ändern (z.B. hexadezimal) • Erst die Simulation, dann die Realisierung in Hardware!
4	Randbedingungen (*Constraints*) für Timing und Platzierung der Ein- und Ausgangspins festlegen	• Timing Constraints: Vorab klären, welche internen und externen Takte existieren. Welche Frequenz/Periode besitzen die Takte? Sind die Takte synchron? • Platzierung der I/O-Pins: Vorab klären, an welchen Pin welches Signal angeschlossen werden kann bzw. werden soll. Welche globalen Steuersignale (z.B. Takte, Reset-Signale und Enable-Signale) gibt es?
5	RTL-Analyse / Synthese, Auswertung der Syntheseergebnisse	• Blockschaltbild der RTL-Analyse auf Übereinstimmung mit VHDL-Code prüfen • *Synthesis Messages* lesen, Code ggf. korrigieren • *Synthesis Report* lesen, Übereinstimmung zwischen synthetisierter Hardware und VHDL-Code prüfen
6	Implementierung, Bitstream erstellen	• Fehler- und Warnmeldungen auswerten • *Utilization Report* lesen, Auslastung des Bausteins überprüfen • *Timing Report* lesen, Einhaltung der Timingvorgaben überprüfen
7	Baustein konfigurieren und Schaltung testen	• Erst kleinere Schaltungsteile in Betrieb nehmen und testen, dann die gesamte Schaltung • Überprüfung der Funktionalität möglichst unter realen Bedingungen (Ein-/Ausgangssignale, Taktfrequenz, Umgebungsbedingungen)

2.3 RTL-Analyse, Synthese und Implementierung – Auswertung der Entwurfsschritte –

Jeder Entwurfsschritt erzeugt Reportdateien, ggf. Warn- oder auch Fehlermeldungen, die Ihnen wertvolle Hinweise zur Korrektur bzw. Optimierung Ihres VDHL-Entwurfs geben. Von besonderer Bedeutung sind folgende Rückmeldungen:

2.3.1 RTL-Analyse

Das von der RTL-Analyse (Elaboration) erzeugte RTL-Blockschaltbild eignet sich zur Kontrolle der hierarchischen Struktur der Schaltung und zum Abgleich mit dem selbst erstellten RTL-Entwurf.

2.3.2 Synthese

Bei einem Syntheselauf wird der VHDL-Code auf Konstrukte überprüft, die zu Unterschieden zwischen der Simulation und der Realisierung in Hardware führen können, z.B.:

- unvollständige Sensitivitätslisten (*signal … is **not** in the sensitivity list*),
- Latches infolge unvollständig auskodierter Bedingungen (*inferring **latch** …*),
- wegoptimierte Ausgangssignale (*unused … was **removed***).

Sollten die Synthesemeldungen derartige Warnungen enthalten, müssen Sie unbedingt Ihren VHDL-Code korrigieren!

Dem *Project Summary* können Sie Informationen darüber entnehmen, wie Ihr VHDL-Code später in Hardware umgesetzt wird. Prüfen Sie

- die Anzahl der Flip-Flops,
- die Anzahl der Zähler,
- die Anzahl der Addierer,
- Anzahl und Typ weiterer Hardwarekomponenten.

Voraussetzung für eine solche Prüfung ist natürlich, dass Sie eine Vorstellung davon haben, wie Ihr Entwurf später in Hardware aussehen wird. Auch hier gilt: Bei gravierenden Unterschieden zwischen Ihren Erwartungen und der synthetisierten Hardware muss nach der Ursache gesucht und der VHDL-Code ggf. korrigiert werden.

2.3.3 Implementierung

Bei komplexeren Schaltungen kann es vorkommen, dass nicht alle Pins verdrahtet werden können. In diesem Fall sind entweder Änderungen am Entwurf notwendig, oder es muss zu einem größeren Baustein gewechselt werden.

3 Vorbereitungsaufgaben

V1 Bearbeiten Sie das Dokument *Vivado-Einführungskurs, Teil 2* bis zum Kapitel *Erzeugen des Bitstreams*.

V2 Der Siebensegment-Dekoder aus dem *Vivado-Einführungskurs, Teil 2* ist bisher als ein Netzwerk von Logikgattern realisiert. Er soll nun mit Hilfe der `case`-Anweisung modelliert werden.

- Erstellen Sie ein neues Vivado-Projekt. Fügen Sie die VHDL-Dateien des Zählers `counter4.vhd` und der Strukturbeschreibung `counter4_dec7seg.vhd` mit *Project Manager > Add Sources > Add or Create Design Sources > Add Files* zum Projekt hinzu. Damit Sie die Korrekturen am Zähler nicht noch einmal durchführen und die Strukturbeschreibung nicht neu erstellen müssen, importieren Sie die Dateien aus dem alten Projektverzeichnis (s. Vorbereitungsaufgabe V1). Die Dateien liegen im Unterordner …`<projektname>\<projektname>.srcs\sources_1\`.
- Erzeugen Sie mit *Project Manager > Add Sources > Add or Create Design Sources > Create File* eine neue VHDL-Datei `dec7seg_c.vhd`. Die Schnittstellen (Ports) sollen identisch zum Siebensegment-Dekoder aus dem Einführungskurs sein.
- Modellieren Sie das Verhalten des neuen Siebensegment-Dekoders mit Hilfe der `case`-Anweisung. Beim Anliegen einer 12_{10} am Eingang des Siebensegment-Dekoders soll ein kleines „c“ ausgegeben werden. Sie können die Wahrheitstabelle aus dem Einführungskurs also nicht unverändert übernehmen.
- Ersetzen Sie in der Strukturbeschreibung `counter4_dec7seg.vhd` die Komponente `dec7sec` durch die neu erstellte Komponente `dec7seg_c`.
- Fügen Sie mit *Project Manager > Add Sources > Add or Create Simulation Sources* die vorgegebene selbsttestende Testbench `sttb_counter4_dec7seg.vhd` zum Projekt hinzu und überprüfen Sie damit die Funktion der Gesamtschaltung.
- Übernehmen Sie die Randbedingungen des Projekts aus dem *Vivado-Einführungskurs, Teil 2*. Fügen Sie dazu die bestehende Constraints-Datei mit *Project Manager > Add Sources > Add or Create Constraints > Add Files* zum Projekt hinzu (siehe …`<projektname>\<projektname>.srcs\constrs_1\`).
- Synthetisieren und implementieren Sie das Projekt und erzeugen Sie den Bitstream.

  Kontrollieren Sie die Warnmeldungen der Synthese. Korrigieren Sie ggf. Ihren VHDL-Code.

V3 Zusätzlich zum Siebensegment-Dekoder soll ein 4-Bit-Gray-Kodierer `bin2gray` mit dem Ausgang des 4-Bit-Binärzählers verbunden werden.

- Informieren Sie sich an geeigneter Stelle über den Gray-Code. Welche Funktion hat ein Gray-Kodierer? Wie viele Bits benötigen die Ein- und Ausgangssignale eines 4-Bit-Gray-Kodierers?
- Erstellen Sie das Blockschaltbild einer Strukturbeschreibung aus den Komponenten `counter4`, `dec7seg_c` und `bin2gray`. Die Ausgänge des vierstelligen Binärzählers `counter4` sollen sowohl mit den Eingängen des Siebensegment-Dekoders `dec7seg_c` als auch den Eingängen des Gray-Kodierers `bin2gray` verbunden sein. Die Ausgänge des Zählers sollen zusätzlich über einen Port `cnt(3:0)` ausgegeben werden, die Ausgänge des Gray-Kodierers über einen Port `gray(3:0)`, die Ausgänge des Siebensegment-Dekoders wie gehabt über die Ports `a…g`.

V4 Mit der Datei `count1to6.vhd` wird Ihnen der funktionsfähige VHDL-Code eines Binärzählers vorgegeben, der zyklisch von 1 bis 6 zählt. Der Zähler besitzt 1-aktive Ausgänge. Sein Zählerstand ändert sich mit der steigenden Taktflanke. Der Reset wirkt asynchron, 1-aktiv und setzt den Zähler auf 1 zurück. Nehmen Sie keine Änderungen am VHDL-Code des Zählers vor.

- Stellen Sie die Spezifikation des Zählers anhand der o.g. Beschreibung tabellarisch dar (s. *Vivado-Einführungskurs, Teil 2*, Kapitel *Spezifikation*). Da der Zähler keinen Ausgangsdekoder besitzt, ist nur das sequentielle Verhalten des Zählers zu beschreiben.
- Erstellen Sie einen Testplan für einen vollständigen Test des Zählers (siehe *Vivado-Einführungskurs, Teil 2,* Kapitel *Testplan*).

Testumfang	…	
Eingangssignale	…	[Hier aussagekräftige Beschreibungen einfügen!]
Ausgangssignale	…	[s.o.]
Testsequenz / Testdauer	1. 2. …	[s.o.]

- Erstellen Sie ein neues Vivado-Projekt. Fügen Sie den Zähler zum Projekt hinzu.
- Erstellen Sie eine Testbench nach den Vorgaben Ihres Testplans.
- Überprüfen Sie die Funktion des Zählers in einer Simulation (Resetverhalten, Taktflanke, Schrittweite und -richtung, Überlauf).

  Erstellen Sie Screenshots von den relevanten Abschnitten der Signalverläufe.[1]
- Definieren Sie die Randbedingungen für die Synthese:
  - Der Zählerstand soll im Binärcode über die Leuchtdioden LD2…LD0 (LSB) auf dem BASYS3-Board angezeigt werden.
  - Der Takt kommt vom Peripherieboard.
  - Als Resettaster wird BTNC auf dem BASYS3-Board verwendet.
- Synthetisieren und implementieren Sie Ihren Entwurf und erzeugen Sie den Bitstream.

[1] Achten Sie darauf, dass im Screenshot alle relevanten Informationen deutlich erkennbar sind. Löschen Sie ggf. unnötige Signale. Zum Erstellen des Screenshots ist es hilfreich, das Fenster mit der Darstellung der Signalverläufe vorher auszudocken und die Fensterdimensionen an den Inhalt anzupassen.

Bitten nehmen Sie trotz der Synthese-Warnungen vorerst keine Änderungen am VHDL-Code Zählers vor!

- Wählen Sie im Fenster *Project Summary* unter *Utilization* die Darstellung *Table* und erstellen Sie einen Screenshot des Fensters *Project Summary*.

V5 Fassen Sie die nachfolgend aufgeführten Informationen und Dokumente in einem PDF-Dokument zusammen.

Bringen Sie das PDF-Dokument zur Laborveranstaltung mit.

Dateiname: `HBS-Laborbericht_V2_Nachname_Vorname.pdf`

- Kopfzeile: *HBS-Labor, V2 – Datum – Nachname, Vorname*
- VHDL-Quellcode der Datei `dev7seg_c.vhd`[2]
- Struktur-Blockschaltbild der Gesamtschaltung aus 4-Bit-Zähler, Siebensegment-Dekoder und Gray-Kodierer
- 1-bis-6-Zähler:
  - Spezifikation
  - Testplan
  - Screenshot der Signalverläufe in der Simulation
  - Constraints-Datei `constr.xdc`
  - Screenshot des Fensters *Project Summary*

4 Freiwillige Vorbereitungsaufgabe

Realisieren Sie die Zusammenschaltung des Zählers, des Siebensegment-Dekoders und des Gray-Kodierers in einer einzigen Verhaltensbeschreibung. Die einzelnen Komponenten werden dabei mit Prozessen modelliert, die Verbindungen der Prozesse mit Signalen. Sie benötigen insgesamt also vier Prozesse: Zwei für den Zähler, einen für den Siebensegment-Dekoder und einen für den Gray-Kodierer. Modellieren Sie den Gray-Kodierer mit Hilfe der `with`-Anweisung.

[2] Formatieren Sie Quellcode mit einer nichtproportionalen Schriftart in relativ kleiner Schriftgröße (z.B. Courier New, 9 Punkt).