Rechnerstrukturen

Hardwareentwurf

Dr. Rainer Buchty
buchty@ira.uka.de

Universität Karlsruhe (TH) – Forschungsuniversität
Institut für Technische Informatik (ITEC)
Lehrstuhl für Rechnerarchitektur

23. April 2009

Entwurfsmethodiken:
Grundlagen
VHDL

Motivation

- **Technologische Entwicklung**
  - Sinkende Strukturgrößen
  - Steigende Chipfläche
  - Erhöhung der Integrationsdichte
  - Komplexere Gesamtfunktionalität bis hin zum SoC
  - aber auch: sinkende Preise
- **Automatisierter Entwurf zwingende Entwicklung**
  - Stetig wachsende Entwurfskomplexität durch steigende Komplexität und technologische Randbedingungen
  - Modularisierung und Wiederverwendbarkeit
  - Time-to-market
- **Logikbausteine** mit traditionellen Entwurfsmethoden nicht handhabbar

Beispiel: IC-Entwurfszeit

- Weit über 10000 Personenjahre Entwicklungszeit bei heutigen Technologien ($> 10^7$ Transistoren)
- Strukturierter Entwurf mit CAD-Werkzeugen nur ca. 10 Personenjahre

Motivation (forts.)

- Programmierbare Logikbausteine bieten **Logik- und Routingressourcen**
- Abbildung des gewünschen Systemverhaltens durch **Konfiguration**
- **Beschreibung des Verhaltens**
  - Vorzugsweise auf abstrakter und unabhängiger Ebene
  - Keine Vorwegnahme technologisch-implementativer Aspekte
  - Lesbarkeit und Wiederverwertbarkeit
  - Simulation und Synthese, d.h. Abbildung und Verifikation
- Konfiguration individueller Bausteine gemäß **Aufteilung des Systems**
  - Digitale vs. analoge Bestandteile
  - Bausteinvorgaben erzwingen ggf. **Verteilung**
    - Sonderfunktionen
    - Kapazitätsprobleme
    - Verteilung

Zielsetzungen des automatisierten Entwurfs

- **Schnellerer Entwurf** (Time-to-Market) und **Kostensenkung**
  - Vereinfachung durch Abstrahierung
  - Wiederverwertbarkeit durch Modularisierung
- **Vereinfachung** durch Abstrahierung
  - Konzentration auf algorithmische Ebene
  - Problem-, nicht Schaltungsbeschreibung
  - Ziel/technologieunabhängige Beschreibung
  - Implementation/Synthese automatisiert auf Basis der höchsten, abstraktesten Entwurfseben
- Wiederverwertbarkeit durch **Modularisierung**
  - Modularisierung: Kapselung von Funktionsgruppen
  - Wiederverwertung der Funktionsgruppen
  - Beispiel: Interfacebausteine

Zielsetzungen des automatisierten Entwurfs (forts.)

Problemstellungen

- **Wahl der abstrakten Beschreibungsform**
  - Architekturunabhängigkeit
  - Problem-Unabhängigkeit
  - Eignung der formalen Spezifikationssprache für Beschreibung auf Architektur- und Systemebene
- **Auswirkungen von Randbedingungen**
  - Mächtigkeit der Sprache (Simulation vs. Synthese)
  - Laufzeit (Extraktion und Generierung)
- **Integration verschiedener Werkzeuge**
  - Übersetzer
  - Minimierer
  - Place&Route
  - Simulation
  - Visualisierung
- **Qualität** von Syntheseergebnis und Simulation

Hardwareentwurf: Entwurfsphasen

1. **Spezifikation** (specification)
   - Beschreibung des Systemverhaltens
   - Definition von Schnittstellen / Interaktion
2. **Systementwurf** (system design)
   - Modellierung des spezifizierten Verhaltens (abstrakt)
   - Black-Box-Sicht: Aufteilung in Komponenten
3. **Logikentwurf** (logic design)
   - Erstellung der Strukturbeschreibung
   - Ableitung der logischen Funktionen aus spezifiziertem Verhalten
4. **Schaltungsentwurf** (circuit design)
   - Konvertierung der Strukturbeschreibung
   - Abbildung auf zur Verfügung stehende Zieltechnologie
5. **Validierung** (validation)
   - Simulation durch Stimulation
   - Vergleich mit erwartetem Verhalten
   - Überprüfung auf einzuhaltende Vorgaben

Hardwareentwurf: Vorgehensweise

- **Spezifikation**
  - Funktion & Verhalten: „Was passiert?“
  - Pflichtenheft
  - Definition grundsätzlicher Eigenschaften
  - Schnittstellen und Verhalten
- **Systementwurf / Modellierung**
  - Struktur: „Wie passiert es?“
  - Beschreibung des gewünschten Verhaltens
  - Modellierung **ggf. bereits implementativ**
  - Abbildung auf Zieltechnologie bereits zum Entwurfszeitpunkt durch Entwickler
    - Diskrete Einzelbausteine: Schaltplan
    - Modellierung basierend auf Bausteineigenschaften
  - **Besser: Abstrakte Modellierung** und Abbildung auf Zieltechnologie durch Werkzeuge
    - Beschreibungssprachen
    - Modellierungswerkzeuge

Hardwareentwurf: Vorgehensweise (forts.)

- **Synthese**
  - Topologie/Geometrie: „Wo passiert es?“
  - Implizit: Schaltungsaufbau anhand von Schaltplan
  - Automatisierte Abbildung auf Zieltechnologie durch Synthesewerkzeug
- **Simulation**
  - Realmodell: Prototyp, Test
  - Virtueller Aufbau: Simulationswerkzeuge
    - Simulation auf unterschiedlichen Ebenen
      Verhaltenssimulation, Logiksimulation, Pre- und Post-Fit-Simulation
- **Programmierung bzw. Fertigung**
  - Konfiguration von Logikbausteinen anhand erzeugter Bitstreams
    - In-Circuit Programming vs. externer (Vor-)Programmierung
  - Fertigung von ASICs
    - Transition: Fertigung von ASICs auf Basis von FPGA-Technologie (feste Konfiguration)

Abstraktionsebenen

- **Aufteilung des Entwurfsprozesses** in Abstraktionsebenen
- **Abstraktion** bedeutet
  - Unterscheidung zwischen wichtiger und (aktuell) unwichtiger Information
  - Entwicklung einer Hierarchieebene
  - Weglassen der auf einer Ebene nicht benötigten Information
  - Zunehmende Spezialisierung nach unten

Beispiel

- Funktionsbeschreibung
- Blockschaltbild
- Schaltplan auf Bausteinebene
- Schaltplan auf Gatterebene
- Schaltplan auf Transistorebene
- Chipmaske

- Wichtig: **Reinhaltung des Abstraktionsgrades**
- Nicht Module unterschiedlichen Abstraktionsgrads mischen

Abstraktionsebenen (forts.)

- **Verhalten** (behavioral)
  - Algorithmische Ebene
  - Modellierung von Bussystemen
  - Stimuli
    → **Systemspezifikation, Standardmodelle**
- **Register-Transfer-Level** (RTL)
  - Ziel-unabhängige Hardwarebeschreibung
  - Register
  - Logik
    → **Synthetisierbare Modelle**
- **Gatterebene** (gate/logic level)
  - Netzliste
  - Gatterstrukturen
    → **PLD-Entwicklung**
- **Layout**
  - Technologieaspekte (Fertigungstechnologie)
    → **Full-custom Design**

Abstraktionsebenen (forts.)

Ebenen und Elemente

Ebene		Typische Elemente
Systemebene	→	Speicher, Prozessoren
Architekturebene oder Algorithmische Ebene	→	Datenpfade, Steuerungen
Register-Transfer-Ebene	→	Register, Zähler, ALU
Logikebene	→	Flipflops, Gatter
Elektrische Ebene	→	Kondensatoren, Transistoren
Physikalische Ebene	→	Halbleiterdotierungen, Flächen

Darstellungsformen

- Gesamtdarstellung (z.B. Blockdiagramm mit Erklärung)
- Programmiersprache (z.B. VHDL)
- RTL-Sprache (z.B. Netzliste)
- Logisches Verhalten (Logikterme)
- Elektrisches Verhalten (Transistor-Darstellung)
- Physikalisches Verhalten (Chipmaske)

Begriffsbildung: Phasen des Entwurfsprozesses

- **Entwurfsprozess** durchläuft mehrere Phasen
  - Analog zu Ebenenmodell
  - Schrittweise Spezialisierung der Beschreibung

- **Spezifikation** und **Modellierung**
  - Abstrakte Beschreibung des Verhaltens, z.B. VHDL-Source
- **Extraktion**
  - Erzeugung einer detaillierten Beschreibung des Verhaltens aus einer abstrakten Beschreibung
  - Beispiel: Logikterme aus abstrakter VHDL-Beschreibung
- **Synthese**
  - Erzeugung einer detaillierten topologischen Struktur (Schaltungsstruktur) aus einer abstrakteren Beschreibung des Entwurfsmodells
  - Abbildung von Logiktermen auf Netzliste (Logikblöcke)
  - Abbildung der Netzliste auf Zieltechnologie $\rightarrow$ Generierung

Begriffe des Entwurfsprozesses (forts.)

Generierung

- Umsetzung einer Schaltungsstruktur in eine detailliertere Form
- Generierung einer **topologischen Struktur**, d.h. Struktur der Schaltung
  - Verfeinerung (De-Abstrahierung) der Darstellung
  - Teilsysteme → Architektur → Funktionseinheiten → Makrozellen → Gatterebene → Transistorebene
- Generierung einer **topographischen Struktur**, d.h. Struktur auf dem Chip
  - Verfeinerung des Maßstabs
  - Wafer → Chip-Plan → Funktionseinheiten → Makrozellen → Funktionselemente → geometrische Elemente

Simulation

- Überprüfung bestimmer Eigenschaften anhand einer geeigneten Verhaltensbeschreibung

Begriffe des Entwurfsprozesses (forts.)

	Sicht		
Ebene	**Funktion/Verhalten**	**Struktur**	**Geometrie**
System	Leistungs-anforderungen	Netzwerk	System-partitionierung
Architektur	Algorithmen	Blockschaltbild	Floorplan
RT	Daten-/Steuerfluss	RT-Diagramm	Standardzellen
Logik	Boolesche Gleichungen	Logiknetzliste, trad. Schaltbild	Makrozellen
Elektrische	Differential-gleichungen	Elektr. Schaltbild	Symb. Layout
Physikalische	Partielle Differential-gleichungen	Dotierungsprofile	Layout

Entwurfsproblematik

- **Abbildbarkeit**: Nicht jede Beschreibung läßt sich auf den gewünschten Baustein abbilden
- **Interne Ressourcen** müssen berücksichtigt werden, d.h.
  - Anzahl von Ein- und Ausgängen
  - Geforderte Funktionalität (Register, Zusatzfunktionen, ...)
  - Mächtigkeit der Verbindungsressourcen
- Beispiel:
  - Anzahl Produktterme beschränkt durch Mächtigkeit der Auswahlmatrix oder implizit vorgegeben (ROM)
  - Bei CPLDs „stehlen“ benachbarter Ressourcen möglich (Product Term Stealing)
- **Aufgabe der Werkzeuge** nicht nur Übersetzung, sondern auch Minimierung und gezielte Abbildung auf vorhandene Ressourcen
- Vergleichbar optimierendem Compiler

Entwurfsablauf

- **Modellierung** in Entwurfssprache
- **Übersetzung** der Entwurfssprache in flache Logikterme
- **Optimierung** der Logikterme
- Simulation auf Logikterm-Ebene (**Pre-fit Simulation**)
- **Aufbereitung** der Logikterme für Zielarchitektur (z.B. Anzahl von Produkttermen)
- **Abbilden** der Logikterme auf Zielarchitektur (Fitting)
- **Bitstrom-Generierung** zur Programmierung
- **Programmierung** des Bausteins
- **Hardware-Test** anhand von Test-Vektoren
- Grundsätzlich kein Unterschied zu heutiger Entwurfssoftware
  - Aufwendigere Übersetzungs-, Optimierungs- und Simulationsmethoden.
  - Größerer Freiraum in der Abbildung durch komplexere Bausteine
  - Ggf. Unterstützung von Teil-Bitströmen für partiell-dynamische Rekonfiguration

Frühe Entwurfssprachen und -werkzeuge

- Ziel: **Vereinfachung des Entwurfs** mit SPLDs und CPLDs
- **Herstellerspezifische Entwurfswerkzeuge** und -sprachen
  - MMI/AMD: PALASM, MachXL (PALASM)
  - Altera: MaxPlus, Quartus (AHDL)
  - Cypress: Warp (VHDL)
- **Unabhängige Entwurfswerkzeuge** und -sprachen
  - Data I/O: ABEL (Another Boolean Equation Language)
    heute: Xilinx
  - Logical Devices, Inc.: CUPL
    (Compiler Universal Programming Language)
    heute: Altium
  - MINC: DSL (Design Synthesis Language)
  - MicroSIM: Schaltplaneingabe basierend auf Logikbibliothek
- **Zielarchitekturen**: SPLDs und CPLDs
- Fitting (Synthese) typischerweise vom Bausteinhersteller lizenziert

Frühe Entwurfssprachen

- Sehr **tiefe Modellierung**
- **Unzureichende Modularisierung**
- Mangelnde Unterstützung von **abstrakten Datentypen**
  - PALASM: Ausformulierung von Zählern statt Addition
  - ABEL: Keine vollständige Unterstützung von Vektoren
- **Vorwegnahme implementativer Gesichtspunkte**
  - Explizite Angabe von Flipfloptyp
  - Explizite Angabe von Setz- und Rücksetzeingängen
  - Keine automatische Ummodelung durch Werkzeug, falls im Baustein nicht vorhanden
- Für Entwürfe jenseits typischer PLD-Größen nicht praktikabel
- **Abstrahierende Entwurfssprachen**
  - VHDL: Systemsimulation
  - Verilog: ASIC-Entwurf
  - System-C: Systemsimulation

Literatur (für Interessierte)

- PALASM2 Fortran Source

  `http://www.brouhaha.com/~eric/retrocomputing/mmi/palasm/`
- PALASM4 V1.5 Software und Sprachreferenz

  `http://www.engr.uky.edu/~melham01/ee481/software.htm`
- ABEL-HDL Primer
  Univ. of Pennsylvania

  `http://www.ese.upenn.edu/rca/software/abel/abel.primer.html`
- Converting ABEL Design Files to CUPL
  Atmel Corp.

  `http://www.atmel.com/dyn/resources/prod_documents/doc3303.pdf`
- Documentation for MINC PLDesigner-XL and PLSynthesizer

  `http://jason.sdsu.edu/minc/`

Die HW-Beschreibungssprache VHDL

- **VHDL** = VHSIC HDL
- **VHSIC** = Very High Speed Integrated Circuits
- VHDL hervorgegangen aus dem VHSIC-Programm der USA
- Standardisierte Hardware-Beschreibungssprache
- Seit 1987 IEEE-Standard, mittlerweile überarbeitet
- Kann **verschiedene Beschreibungen des gesamten Entwurfsablaufs** darstellen
  - Algorithmische Spezifikation (behavioral)
  - Realisierungsnahe Strukturen (RTL)
  - Simulation
- **Vorsicht:**
  Nicht alle Sprachkonstrukte in jedem Entwurfsschritt nutzbar!

Die Entwurfssprache VHDL

- Ursprünglich als **Modellierungssprache nur für die Simulation** konzipiert
- Zunehmender **Einsatz als Sprache für Synthese und Verifikation**
- Einsatz zum **ASIC- und FPGA-Entwurf**
  - Mit Erweiterungen auch Analog-Entwurf möglich
  - Aktuelle Entwurfssprachen: VHDL, Verilog, SystemC
- VHDL umfasst alle **Elemente einer klassischen Programmiersprache**
  - Abgeleitet aus ADA
  - Erweitert um **Konstrukte für Schaltungsentwurf**

VHDL–Fallstricke

- **Mächtigkeit der Sprache** problematisch:
  - Nicht alle Sprachkonstrukte können in Hardware umgesetzt werden
- **Trennung zwischen simulierbar und synthetisierbar**
- Abstraktion erzeugt einen – im Vergleich zu niederen Modellierungssprachen (ABEL, PALASM) – gewissen **Mehraufwand in der Beschreibung**
- Abstraktionsmethoden für Ein- und Umsteiger ggf. gewöhnungsbedürftig
  - Unterschiedliches zeitliches Verhalten von Signalen und Variablen
  - Registerbeschreibung
  - Resolving Functions

Grundlage: Spezifikation der Schaltung

- **Schnittstellen** (Zahl und Art der Ein-/Ausgänge)
- **Gewünschtes Verhalten**
- Eventuelle **weitere Vorgaben** bezüglich Geschwindigkeit, Kosten, Fläche, Leistungsverbrauch etc.

- Nur **Schnittstellen und gewünschtes Verhalten** werden typischerweise in VHDL formuliert
- **Geschwindigkeitsvorgaben nur simulierbar**
  - Einhalten von Zeitfenstern
  - Abgleich mit Fitting-Report (Post-Fit-Simulation)
- **Weitere Parameter** Domäne des Synthesewerkzeugs
  - Design Constraints
  - Optimierung hinsichtlich Fläche/Geschwindigkeit
  - Vorgabe eines Pinouts

Entwurfsschritte

- **Verhaltensverfeinerung**
  - Detailliertes Ausarbeiten der gewünschten Funktionalität
  - Ersetzen von Black-Boxes
- **Strukturverfeinerung**
  - Realisierung einer spezifizierten Funktion durch Verschaltung von Komponenten mit einfacherer Funktionalität
- **Datenverfeinerung**
  - Realisierung abstrakter Datentypen durch einfachere Datentypen
  - Synthese: Binärer Datentyp bzw. erweiterter binärer Datentyp (incl. Tri-state-Zustand, `std_logic`)

→ **Schaltungsbeschreibung**

Erstellen einer Beschreibung

Schnittstellendefinition (entity)

- **Entity** beschreibt Ein-/Ausgabeschnittstelle
- Pro Modul nur eine Entity erlaubt

Verhaltensbeschreibung (architecture)

- Mehrere **Architectures** pro Modul möglich, z.B. zur Unterscheidung von Verhaltens- oder Synthesebeschreibung
- Keine begriffliche Verwandtschaft zu (Mikro)Architektur bei Prozessoren

Konfiguration und Zuordnung (configuration)

- Festlegung verwendeter Architecture
- Konfiguration, z.B. von Signalbreiten

Signalmodi

Signalmodus bestimmt Datenflussrichtung

- **in** kann nur gelesen werden
- **out** kann nur geschrieben werden
- **inout** bezeichnet bidirektionales Signal
- **buffer** ist ebenfalls bidirektional mit eingeschränkter Zuweisungsmöglichkeit
  (VHDL-FAQ: „*The use of buffer ports is discouraged*“)
- **linkage** dient zur Verbindung mit externen, nicht-VHDL Modulen; Semantik werkzeug- bzw. herstellerspezifisch
- Typischerweise nur **in, out, inout** in Verwendung
- Modi werden nur in Entity deklariert, nicht bei internen (d.h. in der Architecture spezifizierten) Signalen

VHDL-Datentypen

- **boolean**: True, False
- **bit**: 0, 1
- **std_logic**
  - Erweiterung von bit mit zusätzlich
  - **Z**: tristate / hochohmig; Signal mit diesem Wert kann von anderen Signalen mit 0 oder 1 überschrieben werden
  - **X**: unbekannt; Datenwert ist durch unvollständige oder fehlerhafte Berechung / Zusammenschaltung entstanden
  - **U**: undefiniert, d.h. uninitialisiert oder unbekannt
- Skalare Datentypen, Arrays (**_vector**), Integer, Characters
- Fließkomma, Datei, Records, eigene Typdefinitionen
- Synthetisierbarkeit beachten!

Zuweisungsoperatoren

- Zuweisung von **skalaren Werten**
  - a<=’1’;
- Zuweisung von **Vektoren**
  - a<=”1010”;
  - a<=(others=>’0’);
- Zuweisung von **Signalen**
  - a<=b;
- Zuweisung von **Termen**
  - a<=not(b);

Zu beachten:

- Zuweisungsoperator unterscheidet zwischen Signalen $<=$ und Variablen $:=$
- VHDL führt strikte Typprüfung durch (auch für Operatoren)
- Ggf. Konvertierung (typecasting) notwendig

Operatoren

VHDL-Operatoren

- Logik: AND, OR, NAND, NOR, XOR, XNOR
- Vergleich: $=$, $/=$, $<$, $<=$, $>$, $>=$
- Schieben: SLL, SRL, SLA, SRA, ROL, ROR
- Arithmetik: +, -, *, /, MOD
- Diverse: **, ABS, NOT
- Wichtig: Operatoren innerhalb einer Gruppe haben **gleiche** Präzedenz
- AND/OR haben gleiche Priorität, d.h. gemischte Ausdrücke passend klammern, um Mehrdeutigkeiten zu vermeiden
- Ordnung der Gruppen nach aufsteigender Präzedenz
- Datentypen und Operatoren werden durch Bibliothek definiert; erst durch Benutzung von entsprechenden Bibliotheken werden Datentypen und Operatoren nutzbar

Auch hier gilt: nicht zwingend in Hardware abbildbar

Datenobjekte

- **Datenobjekte haben Typ und** – je nach Deklarationsort – **Modus**

Signale (signals)

- Häufigstes Datenobjekt
- Datentransport und Datenspeicherung innerhalb von Architectures und über deren Grenzen (Schnittstellen) hinaus.
- Wertezuweisung nebenläufig, wenn nicht explizit serialisiert (Prozesse)

Konstanten (constants)

- Statische Werte

Datenobjekte

Variablen (variables)

- Definition innerhalb des **process**-Headers
- Auf einzelnen Process beschränkt, kein Transport über process-Grenzen hinweg
- Wertezuweisung sequentiell und umittelbar in der Reihenfolge der Abarbeitung
- Unterschied zu Signalen!

Entwurfsbeispiel: NAND-Gatter

$$NAND : f(x, y) = \begin{cases} 0 & wenn\ x = 1 \wedge y = 1 \\ 1 & sonst \end{cases}$$

- Datentyp std_logic soll verwendet werden
- Einladen benötigter Bibliotheken

NAND-Gatter: Einladen benötigter Bibliotheken

```
library IEEE;
use IEEE.std_logic_1164.all;
```

Entwurfsbeispiel (forts.)

$$NAND : f(x,y) = \begin{cases} 0 & wenn\ x = 1 \wedge y = 1 \\ 1 & sonst \end{cases}$$

- Zwei Eingänge x/y, ein Ausgang f(x,y)
- Interfacebeschreibung ergibt Entity

NAND-Gatter: Beschreibung der Schnittstellen

```
entity NAND is
  port(
    -- x,y sind Eingänge vom Typ std_logic
    x,y: in std_logic;
    -- fxy ist Ausgang vom Typ std_logic
    fxy: out std_logic
  );
end entity;
```

Entwurfsbeispiel (forts.)

$$NAND : f(x,y) = \begin{cases} 0 & wenn\ x = 1 \wedge y = 1 \\ 1 & sonst \end{cases}$$

- Funktionsbeschreibung ergibt Architecture

NAND-Gatter: Beschreibung des Verhaltens

```
architecture arch_NAND of NAND is
begin
    -- funktionale Beschreibung
    fxy<='0' when x='1' and y='1' else '1';
end architecture;
```

Entwurfsbeispiel (forts.)

$$NAND : f(x,y) = \begin{cases} 0 & wenn\ x = 1 \wedge y = 1 \\ 1 & sonst \end{cases}$$

- Funktionsbeschreibung ergibt Architecture

NAND-Gatter: Beschreibung des Verhaltens

```
architecture arch_NAND of NAND is
-- Hilfssignal
signal f_and: std_logic;
begin
    -- algorithmische Beschreibung
    f_and<=x and y;
    fxy<=not(f_and);
end architecture;
```

Entwurfsbeispiel (forts.)

$$NAND : f(x,y) = \begin{cases} 0 & wenn\ x = 1 \wedge y = 1 \\ 1 & sonst \end{cases}$$

- Weiterverwendung des NAND-Gatters zur Instantiierung in anderen Architectures ($\rightarrow$ **port map**-Statement)
- Komponentenbeschreibung
- Ggf. Zusammenfassung mehrer Komponenten in eigenem **package** zum Aufbau einer Bibliothek ($\rightarrow$ **use**-Statement)

NAND-Gatter: Komponentenbeschreibung

```
component NAND is
  port(
    x,y: in std_logic;
    fxy: out std_logic
  );
end component;
```

Prozesse und Sequentialität

- Zuweisungen in VHDL grundsätzlich nebenläufig
- Modellierung von Schaltwerken explizit in Prozessen (**process**)
- Beschreibung des Verhaltens über einen sequentiellen Algorithmus
- Simulierte Zeit für die gesamte VHDL-Beschreibung schreitet während Ausführung nicht fort

Aufbau eines Prozesses

- Sensitivity List: Signale, bei deren Änderung der Beschreibungsblock ausgeführt wird
- Rückhalten der Zuweisungen bis Blockende
- Variablendeklaration (**:=**) $\rightarrow$ Zuweisung sofort
- Verhaltensbeschreibung

Prozess-Sprachkonstrukte

Warten auf Ereignis

- **wait for**
- **wait until**
- Problematisch bei Synthese!

Bedingte Zuweisung

- **if/then/else** (entspricht **when/then**-Konstrukt bei nebenläufiger Zuweisung)
- **case/when**-Zuweisung

Schleifen

- **for/loop**
- **while/loop**
- Auch Zuweisungen innerhalb einer Schleife erfolgen erst zum Ende des Blocks
- Ebenfalls problematisch bei Synthese

Modellierung von Speicherelementen

- VHDL verfügt über **kein explizites Sprachelement** zur Erzeugung von Speicherelementen

Modellierung von Speicherelementen

- **Erzeugung über if-Konstrukt** in Prozessen
- Asynchrones Speicherelement (Latch, pegelgesteuert)
- Synchrones Speicherelement (Flip-Flop, taktflankengesteuert)
- Spezifikation über Signalattribut (**'event**) und Signalpegel
- Pro **process** und Taktsignal kann nur eine Taktflanke herangezogen werden
- Speicherelementtyp (D, T, RS, JK) mittels Ausformulierung des gewünschten Verhaltens

Speicherlemenente (forts.)

- Beispiel: **D-Latch**
- Einspeichern von Daten während low-Pegel des Kontrollsignals möglich
- Verriegelt während high-Pegel

Modellierung eines D-Latchs

```
D_LATCH:
process(write,data)
begin
  if write='0' then
    dlatch<=data;
  end if;
end process;
```

Speicherlemenente (forts.)

- Beispiel: **D-Flipflop**
- Übernahme von Daten zur steigenden Taktflanke eines Taktsignals

Modellierung eines D-Flipflops

```
D_FF:
process(clk,data)
begin
  if clk'event and clk='1' then
    dff<=data;
  end if;
end process;
```

Speicherlemenente (forts.)

- Beispiel: **D-Flipflop mit asynchronem Rücksetzeingang**
- Übernahme von Daten zur steigenden Taktflanke eines Taktsignals
- Löschen des Inhaltes asynchron, d.h. unabhängig vom Taktsigal, bei low-Pegel des Rücksetzsignals

Modellierung eines D-Flipflops

```
D_FFR:
process(rst,clk,data)
begin
  if rst='0' then
    dffr<=(others=>'0');
  elsif clk'event and clk='1' then
    dffr<=data;
  end if;
end process;
```

Entwurfsbeispiel: Register-File

- **16 Register zu je 8 Bit**
- Schreib- und lesbar
- Steuersignale: Takt, Rücksetzeingang, Chip-Select
- Zugriff via Adress/Datenbus

Register-File: Interface

```
entity register is
  port(
    clk, rst, cs, rw: in std_logic;
    addr: in std_logic_vector(3 downto 0);
    data: inout std_logic_vector(7 downto 0)
  );
end entity;
```

Entwurfsbeispiel (forts.)

- 16 8-Bit Register → zweidimensionales Array
- **In VHDL nicht direkt abbildbar**, d.h. zweistufiges Verfahren
  - Definition eines Subtypen für 8-Bit Arrays
  - Signaldeklaration
- Ausgabe nicht direkt auf Datenbus sondern in internes Signal

Register-File: Deklarationen

```
architecture arch_reg of register is
subtype sdlv8 is std_logic_vector(7 downto 0);
signal register: sdlv8_vector(15 downto 0);
signal data_out: std_logic_vector(7 downto 0);

begin
  ...
end architecture;
```

Entwurfsbeispiel (forts.)

- **Prozessdeklaration:**
  - Steuersignale sind Takt, Reset, Chip-Select und R/W
  - Auswahl mittels Adressleitungen

Register-File: Deklarationen und Reset)

```
process(clk,rst,cs,wr,addr)
begin
 if rst='0' then
   register(0)<=(others=>'0');
   register(1)<=(others=>'0');
   ...
   -- alternativ: generate-Statement
```

Entwurfsbeispiel (forts.)

- Zugriff synchron zu Takt
- Zugriffsart bestimmt durch R/W
- Gültigkeit des Zugriffs durch Chip-Select
- Schreiben synchron, Lesen asynchron

Register-File: Zugriffe

```
    elsif clk'event and clk='1' then
      if rw='0' and cs='0' then
        -- Datum in Register schreiben
      end if;
    end if;
  end process;
  -- Datum aus Register lesen
end architecture;
```

Entwurfsbeispiel (forts.)

- Register lesen: nebenläufig, nicht taktflankengesteuert
- Ausgabe auf Datenbus gesteuert durch R/W und Chip-Select
- Nebenläufige Zuweisung außerhalb des Prozesses

Register-File: Asynchroner Lesezugriff

```
data<=data_out when rw='1' and cs='0'
 else (others=>'Z');
data_out<=register(0) when addr="0000"
     else register(1) when addr="0001"
     ...
```

Entwurfsbeispiel (forts.)

- Register schreiben: synchron zu Taktflanke
- Plazierung innerhalb des Prozesses

Register-File: Synchroner Schreibzugriff

```
case addr is
  when "0000" => register(0)<=data;
  when "0001" => register(1)<=data;
  ...
  when others => null;
end case;
```

Weiterführende Literatur

Literatur

- VHDL-Cookbook von Peter J. Ashenden mit detaillierter Einführung und sehr ausführlichem Entwurfsbeispiel (DP32-Prozessor)
  `http://tech-www.informatik.uni-hamburg.de/vhdl/doc/cookbook/VHDL-Cookbook.pdf`

Für Interessierte

- The Hamburg VHDL Archive
  `http://tams-www.informatik.uni-hamburg.de/vhdl/`
- Freie IP-Cores in VHDL und Verilog
  `http://www.opencores.org`