Rechnerstrukturen im SS2006

Hardwareentwurf

Dr. Rainer Buchty
buchty@ira.uka.de

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

31.05.2006

Hardwareentwurf: Inhalte

Teil 1: Grundlagen

- **Schematischer Ablauf**
- **Logikbausteine**

Teil 2: Praktische Anwendung

- *Hardwarebeschreibungssprachen am Beispiel VHDL*

Hardwareentwurf

- **Klassischer Hardwareentwurf**
  - Diskrete Elektronikbauteile
    - Passive Bauteile (Widerstände, Kondensatoren, ...)
    - Aktive Bauteile (Transistoren, Röhren)
  - Integrierte Schaltkreise (ICs)
    - Ab 1970: Niedere Funktionen
    - Boolesche Logik, Zähler, Vergleicher
    - “Standardisierte” Serien: TTL 74xx, CMOS 40xx
  - Komplexe Integrierte Schaltkreise (LSI, VLSI)
    - Komplexe integrierte Funktionen
    - Recheneinheiten (ALUs)
    - Interfacebausteine
    - Mikroprozessoren
- **Programmierbare Logikbausteine**
  - “Freie” Verschaltung der Logik-Ressourcen
  - Steigende Komplexität (PLDs, CPLDs, FPGAs)
  - Heute: Nicht nur Logik und Speicherzellen, auch Zusatzfunktionen
    $\rightarrow$ SoC möglich

Hardwareentwurf (forts.)

- **Modellierung**
  - Beschreibung des gewünschten Verhaltens
  - früher: Schaltplan
  - Abbildung auf Zieltechnologie bereits zum Entwurfszeitpunkt durch Entwickler
  - heute: Beschreibungssprachen, Modellierungswerkzeuge Abstrakte Modellierung, Abbildung auf Zieltechnologie durch Werkzeuge
- **Simulation**
  - früher: Prototyp, Test (Realmodell)
  - heute: Simulationswerkzeuge (Virtueller Aufbau) Verhaltenssimulation, Pre- und Post-Fit-Simulation
- **Synthese**
  - früher: implizit (Schaltungsaufbau anhand von Schaltplan)
  - heute: automatisierte Abbildung auf Zieltechnologie

Entwurfsmethodik

- **Schaltplanentwurf**
  - Historischer Ansatz
  - Forschreibung der bisherigen Arbeitsweise
  - Nachteil: Keine abstrakte Modellierung, sondern bereits implementativ (Bauteileauswahl, Abbildung hierauf)
- **Entwurfssprachen**
  - Einfache Entwurfssprachen (PALASM, ABEL): erfordern Kenntnis über Zielhardware
  - Abstrakte Entwurfssprachen (VHDL, Verilog, SystemC): idealerweise keine Kenntnis über Zielhardware notwendig
- **Simulation**
  - Zeitgenaue vs. funktionelle Simulation
  - Pre/Post-Fit-Simulation
  - Stimuli / Testvektoren

Entwurfsphasen

- **Spezifikation** (specification)
  - Beschreibung des Systemverhaltens
- **Systementwurf** (system design)
  - Modellierung des spezifizierten Verhaltens (abstrakt)
- **Logikentwurf** (logic design)
  - Erstellung der Strukturbeschreibung
- **Schaltungsentwurf** (circuit design)
  - Konvertierung der Strukturbeschreibung
- **Validierung** (validation)
  - Simulation durch Stimulation
  - Vergleich mit erwartetem Verhalten

Zielsetzungen

- **Zielsetzungen**: Schnellerer Entwurf (Time-to-Market) und Kostensenkung

- **Vereinfachung** durch Abstrahierung
  - Synthese auf der höchsten, abstraktesten Entwurfseben
  - Problem-, nicht Schaltungsbeschreibung
  - Ziel/technologieunabhängige Beschreibung
- Wiederverwertbarkeit durch **Modularisierung**
  - Modularisierung
  - Wiederverwertbarkeit

- **Probleme**
  - Eignung der formalen Spezifikationssprache für Beschreibung auf Architektur- und Systemebene
  - Auswirkungen von Randbedingungen (Mächtigkeit, Laufzeit)
  - Integration verschiedener Werkzeuge (Übersetzer, Minimierer, Place&Route, Simulation, Visualisierung)
  - Qualität des Syntheseergebnisses

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

Beispiel

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

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.)

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

Begriffe des Entwurfsprozesses

- **Extraktion**
  - Erzeugung einer abstrakten Beschreibung des Verhaltens aus einer detaillierteren Beschreibung

Hierarchie mit Beispielen

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

- **Synthese**
  - Erzeugung einer detaillierten topologischen Struktur (Schaltungsstruktur) aus einer abstrakteren Beschreibung des Entwurfsmodells

Begriffe des Entwurfsprozesses (forts.)

- **Generation**
  - Umsetzung einer Schaltungsstruktur in eine detailliertere Form, d.h.
  - Generation einer **topologischen Struktur**
    - Struktur der Schaltung
    - Teilsysteme → Architektur → Funktionseinheiten → Makrozellen → Gatterebene → Transistorebene
  - Generation einer **topographischen Struktur**
    - Struktur auf dem Chip
    - Wafer → Chip-Plan → Funktionseinheiten → Makrozellen → Funktionselemente → geometrische Elemente

- **Simulation**
  - Überprüfung bestimmer Eigenschaften anhand einer geeigneten Verhaltensbeschreibung

Zielarchitekturen

- **Full-custom Logic**
  - Freies Layout im Rahmen der Möglichkeiten der Fertigungstechnologie
- **ASICs**
  - Abbildung auf durch ASIC-Technologie vorgebenes Raster unter Verwendung von Bibliotheken
- **Programmierbare Logikbausteine**
  - Abbildung auf vorgegebene Strukturen unterschiedlicher Granularität und Mächtigkeit
  - früher: einmal programmierbar (PROM)
  - heute: reprogrammierbar (ggf. dynamisch zur Laufzeit)
  - (EE)PROMs
  - SPLDs (PLA, PAL, PALCE, GAL) und CPLDs
  - FPGAs

Programmierbare Logikbausteine (PLDs)

- **Historie**
  - 1975 entwickelt Signetics den PLA
  - 1978 bringt Monolithic Memories Inc. (MMI) den PAL als kostengünstige Alternative zum PLA heraus
  - 1984 führt Lattice den GAL ein, eine elektrisch löschbare PAL-Variante
  - 1985 entwickelt Xilinx den ersten (SRAM-basierten) FPGA

- **Einsatz**
  - Glue-Logic (CPLDs)
  - Komplexe Funktionen und ASIC- oder System-Prototyping (FPGAs)
  - Kleinserien
  - Spezialanwendungen (Rekonfigurierbare Systeme)

Programmierbare Logikbausteine (forts.)

Klassifizierung

- PLDs
  - Oberbegriff
  - Oft eingeschränkt auf SPLDs und CPLDs
- S(imple)PLDs: PLA, PAL, GAL
  - Einfache Logikbausteine basierend auf UND- und ODER-Matrix zur Modellierung von Ein- und Ausgabeverhalten
- C(omplex)PLDs
  - SPLD-Weiterentwicklungen mit mehreren 1000 Gatteräquivalenten
- FPGAs
  - Komplexe Matrixstruktur aus konfigurierbaren Logikblöcken
  - Heute auch: Zusatzfunktionen (DSP, RAM, I/O)

Programmable Logic Devices (forts.)

- PLDs bestehen klassischerweise aus zwei Matrizen beschränkter Größe
- **UND-Matrix** erlaubt die **Konstruktion von Produkttermen aus Eingangssignalen**
- **ODER-Matrix** erlaubt die **Auswahl von Produkttermen** und bestimmt Ausgangsverhalten

Unterschiedliche Freiheitsgrade

- **ROM**s haben feste UND-Matrix (Speicheradresse) und programmierbare ODER-Matrix (Ausgabedatum)
- **PAL**s haben programmierbare UND-Matrix (Produktterm) und feste ODER-Matrix (High oder Low, wenn Produktterm erfüllt)
- Bei **PLA**s sind beide Matrizen programmierbar
- Heute: Unterschied zwischen PAL und PLA verwischt

Programmable Logic Devices (forts.)

- **Klassische PAL/PLA-Struktur**
  - Rein nebenläufig, keine Speicherglieder
  - Zentrale Routing-Instanz (Matrix)
  - Eingeschränkte Möglichkeit zur Rückkopplung (extern)
  - Ursprüngliches Einsatzgebiet: En-/Decoder, komplexere Logikfunktionen

- **Verbesserungen und Erweiterungen**
  - Einführung von Speichergliedern an ODER-Matrix-Ausgang erlaubt sequentielle Logik (Zustandsautomaten)
  - Interne Rückführungsmöglichkeit
  - Programmierung beider Matrizen
  - Bildung sogenannter Makrozellen (CPLDs)
  - Dediziertes I/O-Routing für mehr Flexibilität bei der Pinbelegung
  - Hierarchische Architekturen (z.B. Vantis/Lattice MACH5)

Klassifikation von SPLDs

- **Uniforme Bezeichnung** von PALs und GALs: PLDxxyyzz
- PLD gibt den Bausteintyp an (PAL, PALCE, GAL)
- xx die maximal mögliche Anzahl von Eingängen, welche zur Bildung von Produkttermen herangezogen werden können (inklusive bidirektionaler Ausgangspins, sofern vorhanden)
- zz Anzahl der Ausgänge
- yy Ausgangskonfiguration

Beispiele

- PAL22L10: PAL-Baustein in L-Konfiguration mit 22 Ein- und 10 Ausgängen
- GAL16V8: GAL-Baustein in V-Konfiguration mit 16 Ein- und 8 Ausgängen

Klassifikation von SPLDs (forts.)

- **A**: Arithmetisch, Register im Ausgang
- **C**: komplementäre Ausgänge, d.h. es kann zwischen invertiertem und nicht invertiertem Ergebnis gewählt werden
- **H**: high-aktive Logik (Ausgang=H wenn Eingangsbedingungen erfüllt)
- **L**: low-aktive Logik (Ausgang=L wenn Eingangsbedingungen erfüllt)
- **P**: Programmierbarer Pegel (H oder L)
- **R**: Register im Ausgang
- **S**: Steuerung durch Produktterme
- **V**: Vielseitig (versatile)
- **X**: Exklusiv-Oder, Register im Ausgang
- Klassische Typen: **H**, **L**, später **R**
- Abgelöst durch **V** (kann alle Typen nachempfinden)

Field Programmable Gate Arrays (FPGAs)

- **Verknüpfung der Vorteile** von PLDs (kurze Entwurfszeiten, anwenderprogrammierbar, Standardprodukt) mit denen von Gate Arrays (Flexibilität, Komplexität)
- Realisation von Logikfunktionen in Form von **Lookup-Tables** (LUT) anstatt Produkttermen (Boole'sche Logik)
- Zentrales Element ist der **Configurable Logic Block**
- Xilinx XC4000: 2 4-Bit LUTs mit Auswahl (exklusiv oder kombiniert) und assoziierten Speichergliedern
- **Beliebige Verknüpfung** von Eingabesignalen und Ergebnissen über programmierbare Schaltmatrix
- Heute: **Zusatzfunktionalität** (Multiplizierer, RAM-Blöcke, serielle Interfaces, CPU-Kerne)

Field Programmable Gate Arrays (forts.)

- Konzeptionelle Unterschiede zwischen FPGAs und (C)PLDs

FPGA

Struktur	viele kleine Logikblöcke, aufwendige und flexible Routing-Matrix
Verbindung	dezentral, lokal
Konfiguration	flüchtig (SRAM)
Größe	bis zu mehreren Mio. Gatteräquivalenten

CPLD

Struktur	wenige große Logikblöcke, eingeschränkte, fest vorgegebene Routing-Matrix
Verbindung	zentral, global
Signallaufzeiten	homogen, konstant und kurz, daher vorhersagbar
Konfiguration	nichtflüchtig (EEPROM)
Größe	typ. $<$100.000 Gatteräquivalente

Entwurf mit PLDs

- **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)
  - 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

Rechnerstrukturen im SS2006

Hardwareentwurf

Dr. Rainer Buchty
buchty@ira.uka.de

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

01.06.2006

Hardwareentwurf: Inhalte

Teil 1: Grundlagen

- *Schematischer Ablauf*
- *Logikbausteine*

Teil 2: Praktische Anwendung

- **Hardwarebeschreibungssprachen am Beispiel VHDL**

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 Entwurfsablauf darstellen
  - Algorithmische Spezifikation (behavioral)
  - Realisierungsnahe Strukturen (RTL)
  - Simulation
- Vorsicht: **Nicht alle Sprachkonstrukte in jedem Entwurfsschritt nutzbar!**

VHDL (forts.)

Die Entwurfssprache VHDL

- Ursprünglich als Modellierungssprache nur für die Simulation konzipiert
- Zunehmender Einsatz als Sprache für Synthese und Verifikation
- heute: VHDL, Verilog, SystemC
- Einsatz zum ASIC- und FPGA-Entwurf, mit Erweiterungen auch Analog-Entwurf möglich
- VHDL umfaßt alle Elemente einer klassischen Programmiersprache (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 eine – im Vergleich zu niederen Modellierungssprachen (ABEL, PALASM) – gewissen **Mehraufwand in der Beschreibung**
- Methoden der Abstraktion für Ein- und Umsteiger ggf. gewöhnungsbedürftig (z.B. Registerbeschreibung)

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)
- Weitere Parameter Domäne des Synthesewerkzeugs

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

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 (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 erst 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

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 architeture;
```

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 (**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 (**:=**), 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 $\rightarrow$ 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-Kurzanleitung von Georg Acher und Markus Leberecht

  `http://www.lrr.in.tum.de/~acher/tgi/uebung/VHDL-Buch.pdf`
- 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`