

Rechnerstrukturen

Vorlesung im Sommersemester 2002

Wolfgang Karl

Universität Karlsruhe

Fakultät für Informatik

Vorlesung Rechnerarchitektur

□ Kapitel 1: Grundlagen

Bewertung der Leistungsfähigkeit

Bewertung der Leistungsfähigkeit

□ Verfahren

- ◆ Auswertung von Hardwaremaßen und –parametern
 - ◆ Mixe
 - ◆ Kernprogramme
- ◆ Laufzeitmessungen bestehender Programme
 - ◆ Benchmarks
- ◆ Messungen während des Betriebs von Anlagen
- ◆ Modelltheoretische Verfahren

Benchmarks

□ Standardisierte Benchmarks

- ◆ Portierbarkeit
- ◆ Vergleichbarkeit von Rechnern
- ◆ Sammlung von Benchmark-Programmen (Benchmark Suites)
 - ◆ Ausgleich durch die unterschiedlichen Eigenschaften der Programme

Benchmarks

□ Standardisierte Benchmarks

- ◆ Standardisierungsorganisationen
 - ◆ SPEC (Standard Performance Evaluation Corporation)
 - ◆ Gegründet 1988, <http://www.spec.org>
 - ◆ Zusammenschluss von mehr als 40 Firmen (Rechnerhersteller)
 - ◆ Festlegung von Richtlinien für eine gemeinsame Rechnerbewertung
 - ◆ TPC (Transaction Processing Performance Council)
 - ◆ Mitte der 80'er Jahre, <http://www.tpc.org>
 - ◆ Zusammenschluss von Herstellern
 - ◆ Datenbankbereich
 - ◆ EEMBC Benchmarks
 - ◆ Anwendungen aus dem Bereich Eingebettete Systeme
 - ◆ PARKBENCH
 - ◆ Parallelverarbeitung, <http://www.netlib.org/parkbench>

Benchmarks

□ SPEC

◆ Philosophie von SPEC

„The goal of SPEC is to ensure that the marketplace has a fair and useful set of metrics to differentiate candidate systems. The path chosen is an attempt to balance between requiring strict compliance and allowing vendors to demonstrate their advantages. The belief is that a good test that is reasonable to utilize will lead to a greater availability of results in the marketplace.

The basic SPEC methodology is to provide the benchmarker with a standardized suite of source code based upon existing applications that has already been ported to a wide variety of platforms by its membership. The benchmarker then takes this source code, compiles it for the system in question and then can tune the system for the best results. The use of already accepted and ported source code greatly reduces the problem of making apples-to-oranges comparisons.“

Benchmarks

□ Verschiedene Untergruppen in SPEC

- ◆ Open Systems Group (OSG)
 - ◆ **SPEC CPU 2000**: Prozessorleistung
 - ◆ **SPEC JBB2000**: JAVA Business Benchmark
 - ◆ **SPEC JVM98**: Vergleich von Java Virtual Machine Client Plattformen
 - ◆ **SPEC MAIL2001**: Mail Server Benchmark
 - ◆ **SPEC SFS 2.0**: System: File Server
 - ◆ **SPEC WEB99**: Test für WWW Server
- ◆ High Performance Group (HPG)
 - ◆ **SPEChpc96**: Industrielle große Anwendungen
- ◆ Graphics Performance Characterization Group (GPC)
 - ◆ Zusammenschluss mit SPECopc: OpenGL Leistungsbewertung

Beispiel SPEC CPU2000

□ SPEC CINT2000

Benchmark	Referenzzeit	Bemerkung
164.gzip	1400	Datenkompression
175.vpr	1400	FPGA Layout und Platzierung
176.gcc	1100	C Compiler
181.mcf	1800	Lineare Optimierung
186.crafty	1000	Schachprogramm
197.parser	1800	Sprachverarbeitung
252.eon	1300	Ray Tracing
253.perlbmk	1800	Perl
256.gap	1100	Gruppentheorie
255.vortex	1900	Datenbank
256.bzip2	1500	Datenkompression
300.twolf	3000	Platzierung und Routing Simulator

Beispiel SPEC CPU2000

□ SPEC CFP2000

Benchmark	Referenzzeit	Bemerkung
168.wupwize	1600	Quantenphysik
171.swim	3100	Wettervorhersage
172.mgrid	1800	3D-Mehrgitterverfahren
173.applu	2100	Lösung parabolischer u. elliptischer part. DG
177.mesa	1400	3D Graphikbibliothek
178.galgel	2900	Strömungsmechanik
179.art	2600	Neuronales Netzwerk Simulation
183.quake	1300	Finite Elemente Simulation: Erdbeben
187.facerec	1900	Bilderkennung
188.ammp	2200	Chemie
189.lucas	2000	Zahlentheorie
191.fma3d	2100	Finite Elemente: Crash-Simulation
200.sixtrack	1100	Teilchenbeschleunigermodell
300.apsi	2600	Schadstoffverteilung

Beispiel SPEC CPU2000

□ Mögliche Kategorien

	Geschwindigkeit	Durchsatz
Agressive Optimierung	SPECint2000	SPECint_rate2000
	SPECfp2000	SPECfp_rate2000
Konservative Optimierung	SPECint_base2000	SPECint_rate_base2000
	SPECfp_base2000	SPECfp_rate_base2000

- Regeln sind genau festgelegt: *SPEC Run and Reporting Rules*
- Referenzmaschine: Sun Ultra10, 300 MHz Ultra-III Prozessor

Beispiel SPEC CPU2000

□ Metriken (Geschwindigkeit):

- ◆ Jeder Benchmark hat seine eigene **SPECratio**:
 $\text{xxx.benchmark Referenzzeit} / \text{xxx.benchmark Laufzeit}$
- ◆ **SPECint2000**: geometrisches Mittel der 12 „Integer“ SPECratios (aggressive Optimierung)
- ◆ **SPECint_base2000**: geometrisches Mittel der 12 „Integer“ SPECratios (konservative Optimierung)
- ◆ **SPECfp2000**: geometrisches Mittel der 14 „FP“ SPECratios (aggressive Optimierung)
- ◆ **SPECfp_base2000**: geometrisches Mittel der 14 „FP“ SPECratios (konservative Optimierung)

Beispiel SPEC CPU2000

□ Metriken (Durchsatz):

- ♦ Jeder Benchmark hat seine eigene **SPECrate**:
Anzahl der ausgeführten Kopien * (Referenzzeit von xxx.benchmark * Anzahl der Sekunden / längste SPEC CPU2000 Referenzzeit) / Laufzeit aller xxx.benchmarks
- ♦ **SPECint_rate2000**: geometrisches Mittel der 12 „Integer“ SPECrates (aggressive Optimierung)
- ♦ **SPECint_rate_base2000**: geometrisches Mittel der 12 „Integer“ SPECrates (konservative Optimierung)
- ♦ **SPECfp_rate_2000**: geometrisches Mittel der 14 „FP“ SPECrates (aggressive Optimierung)
- ♦ **SPECfp_rate_base2000**: geometrisches Mittel der 14 „FP“ SPECrates (konservative Optimierung)

Beispiel SPEC CPU2000

Processor	Alpha 21264C	AMD Athlon XP	HP PA-8700	IBM Power 4	Intel Itanium	Intel Itanium 2	Intel Xeon	MIPS R14000	Sun UltraSPARC III
System or Motherboard	Alpha ES40 Model 6	EpoX 8KHA+	HP9000 rp7400	pSeries 690 Turbo	HP I2000	HP RX2600	Dell Prec. 340	SGI 3200	Sun Blade 2050
Clock Rate	1,001MHz	1.8GHz	750MHz	1,300MHz	800MHz	1,000MHz	2,530MHz	600MHz	1,050MHz
External Cache	8MB	None	None	128MB	4MB	None	None	8MB	8MB
164.gzip	463	903	495	563	332	583	911	322	433
175.vpr	534	448	550	718	376	704	511	572	460
176.gcc	693	504	710	788	407	1,014	1,058	445	577
181.mcf	550	346	396	1,087	402	834	625	783	659
186.crafty	816	1,059	642	673	356	781	860	502	558
197.parser	418	674	427	445	296	660	883	409	488
252.eon	799	1,452	508	985	370	1,004	1,162	507	527
253.perlbmk	635	1,094	510	694	320	815	1,171	367	540
254.gap	454	844	242	746	256	680	1,130	308	372
255.vortex	844	1,157	930	1,246	459	1,193	1,338	679	738
256.bzip2	656	593	411	868	334	759	709	493	629
300.twolf	795	568	781	1,027	449	880	750	645	570
SPECint_base2000	621	738	520	790	358	810	893	483	537
168.wupside	660	928	333	1,570	591*	1,003	1,234	434	659
171.swlm	1502	840	790	1,493	1,369*	3,205	1,425	529	980
172.mgrid	517	551	469	726	749*	1,720	845	379	487
173.applu	741	575	591	960	1,022*	2,033	939	381	310
177.mesa	800	968	570	665	329*	642	1,043	425	543
178.galgel	1638	578	1,374	2,637	1,019*	2,505	1,133	1,398	1,713
179.art	1782	431	500	1,703	2,369*	4,226	586	1,436	9,389
183.equake	332	574	239	1,696	834*	1,871	903	347	645
187.facerec	1024	757	334	1,310	637*	1,152	1,134	647	958
188.ammp	592	477	472	751	511*	788	563	573	509
189.lucas	891	666	300	1,124	837*	1,206	1,155	442	371
191.fma3d	703	711	273	855	323*	747	806	306	400
200.sixtrack	401	456	407	505	575*	894	459	298	366
301.aspl	641	504	591	940	350*	678	684	406	471
SPECfp_base2000	776	624	464	1,098	703*	1,356	878	499	701

*Dell PowerEdge 7150 with 4MB L3 cache

Quelle: MDR, Microprocessor Report, August 26, 2002

Messungen während des Betriebs

□ Monitore

- ◆ Aufzeichnungselemente, die zum Zweck der Rechnerbewertung die Verkehrsverhältnisse während des normalen Betriebs beobachten und untersuchen.
- ◆ **Hardware-Monitore**
 - ◆ Unabhängige physikalische Geräte
 - ◆ Keine Beeinflussung

Messungen während des Betriebs

□ Monitore

◆ Software-Monitor

- ◆ Einbau in das Betriebssystem
- ◆ Beeinträchtigung der normalen Betriebsverhältnisse

◆ Aufzeichnungstechniken:

- ◆ Kontinuierlich oder sporadisch
- ◆ Gesamtdatenaufzeichnung (Tracing)
- ◆ Realzeitauswertung
- ◆ Unabhängiger Auswertungslauf (Post Processing)

Modelltheoretische Verfahren

□ Analytische Methoden

- ◆ Deterministische Warteschlangenmodelle
 - ◆ Systemparameter: feste Werte
- ◆ Stochastische Warteschlangenmodelle
 - ◆ Systemparameter: Mittelwerte
- ◆ Operationelle Warteschlangenmodelle
 - ◆ Systemparameter: gemessene Werte

□ Simulationen

- ◆ Deterministische Simulation
- ◆ Stochastische Simulation
- ◆ Aufzeichnungsgesteuerte Simulation

Lautzeitmessungen bestehender Programme

□ **Maßzahlen für die Operationsgeschwindigkeit**

- ◆ **MIPS: Millions of Instructions Per Second**
 - ◆ $\text{MIPS} = \text{Anzahl der Befehle} / \text{Ausführungszeit} \times 10^6$
 - ◆ $\text{Ausführungszeit} = \text{Anzahl der Befehle} \times 10^6$
 - ◆ MIPS ist abhängig vom Befehlssatz
 - ◆ Schwierig: Rechner mit unterschiedlichen Befehlssätzen zu vergleichen
 - ◆ Unterschiedliche MIPS-Zahl auf einem Rechner bei verschiedenen Programmen
 - ◆ MIPS kann umgekehrt zur Leistung variieren
- ◆ **MFLOPS: Millions of Floating Point Instructions Per Second**

Bewertung der Leistungsfähigkeit

□ Gleichung der CPU-Leistung

- ◆ Rechner laufen mit einer festen Taktrate
 - ◆ Takt: ticks, clock ticks, clock periods, clock cycles,
- ◆ Takt:
 - ◆ Zeitdauer (z.B. 1 ns)
 - ◆ Taktrate (z.B. 1 GHz)
- ◆ CPU Time für ein Programm:

CPU time =

Anzahl CPU Taktzyklen für ein Programm * Taktdauer =

Anzahl CPU Taktzyklen für ein Programm / Taktrate

Bewertung der Leistungsfähigkeit

□ Gleichung der CPU-Leistung

- ◆ Anzahl der ausgeführten Instruktionen (instruction count, IC)
- ◆ Anzahl der Zyklen pro Instruktion (Cycles Per Instruction, CPI)

$$\text{CPI} = \text{Anzahl CPU Taktzyklen für ein Programm} / \text{IC}$$

- ◆ Aufschluss über verschiedene Entwürfe von Befehlssätzen und Implementierungen

Bewertung der Leistungsfähigkeit

□ Gleichung der CPU-Leistung

- ◆ Umformung der Gleichung liefert:

$$\begin{aligned}\text{CPU time} &= \text{IC} * \text{Taktdauer} * \text{CPI} \\ &= (\text{IC} * \text{Taktdauer}) / \text{Taktrate}\end{aligned}$$

- ◆ Erweiterung:

$$\begin{aligned}\text{CPU time} &= \text{Instruktionen pro Programm} * \\ &\quad \text{Anzahl Zyklen pro Befehl} * \\ &\quad \text{Sekunden / Anzahl Zyklen}\end{aligned}$$

Bewertung der Leistungsfähigkeit

□ Gleichung der CPU-Leistung

- ◆ CPU Leistung ist abhängig von
 - ◆ Taktzyklus: Technologie und Organisation
 - ◆ CPI: Organisation und Befehlssatz (Architektur, ISA)
 - ◆ IC: ISA und Compiler Technologie

Vorlesung Rechnerarchitektur

□ Kapitel 1: Grundlagen

Allgemeine Grundlagen des Entwurfs

Rechnerentwurf: Ziele

□ Ziele beim Entwurf eines Rechners

- ◆ Verarbeitungsgeschwindigkeit (Performance)
- ◆ (Objektcode-)Kompatibilität
- ◆ Leistungsaufnahme/Stromverbrauch
- ◆ Benutzerfreundlichkeit
- ◆ Verlässlichkeit/Robustheit bzw. Ausfalltoleranz (Fehlertoleranz): Gewährleistung einer gewissen Minimalverfügbarkeit des Systems

Rechnerentwurf: Ziele

□ Ziele beim Entwurf eines Rechners

- ◆ Erweiterbarkeit (Scalability)
- ◆ Flexibilität (Universalrechner statt Spezialrechner)
- 👍 Entwurfsprozess besteht aus dem Abwägen zwischen vielen verschiedenen Kriterien (*Trade-Off*).

Rechnerentwurf

□ **Gestaltungsgrundsätze beim Entwurf eines Rechners**

- ◆ **Konsistenz:** folgerichtiger, schlüssiger Entwurf.
- ◆ **Orthogonalität:** Komponenten sind unabhängig voneinander spezifizierbar.
- ◆ **Symmetrie:** mathematisch symmetrische Dinge im System auch symmetrisch verarbeiten.
- ◆ **Angemessenheit:** die Funktionalität der benutzten Elemente wird bei der Lösung der vorgesehenen Problemstellung ausgeschöpft.

Rechnerentwurf

□ Gestaltungsgrundsätze beim Entwurf eines Rechners

- ◆ **Sparsamkeit (Ökonomie):** Kosten eines Systems möglichst gering halten.
- ◆ **Wiederverwendbarkeit:** Komponenten
- ◆ **Transparenz:** Überschaubarkeit des Systems, Verstecken von Details.
- ◆ **Virtualität:** Begrenzung der Implementierung vor dem Benutzer verbergen.

Rechnerentwurf: Vorgehensweise

□ **Der Entwurf eines Rechnersystems**

- ◆ legt die grobe Gliederung des Rechners (die Rechnerstruktur) in
 - ◆ Prozessoren,
 - ◆ Bussysteme,
 - ◆ Hauptspeicher,
 - ◆ Cachespeicher,
 - ◆ Plattenspeicher,
 - ◆ Grafikinterface und sonstige Schnittstellen fest.
- ◆ Zu berücksichtigen sind Fragen
 - ◆ der Zuverlässigkeit, des Durchsatzes, der Zugriffszeiten, der Bandbreiten, der maximalen Kapazitäten, der Erweiterbarkeit usw.

Rechnerentwurf: Vorgehensweise

□ Der Entwurf einer Rechner- / Prozessorarchitektur

- ◆ legt den Maschinenbefehlssatz und
- ◆ die durch ihn angesprochenen Objekte (wie Register, Speicher, Ein-/Ausgabe-Schnittstelle, Interruptsystem) fest.

Rechnerentwurf: Vorgehensweise

□ Der Logik-Entwurf

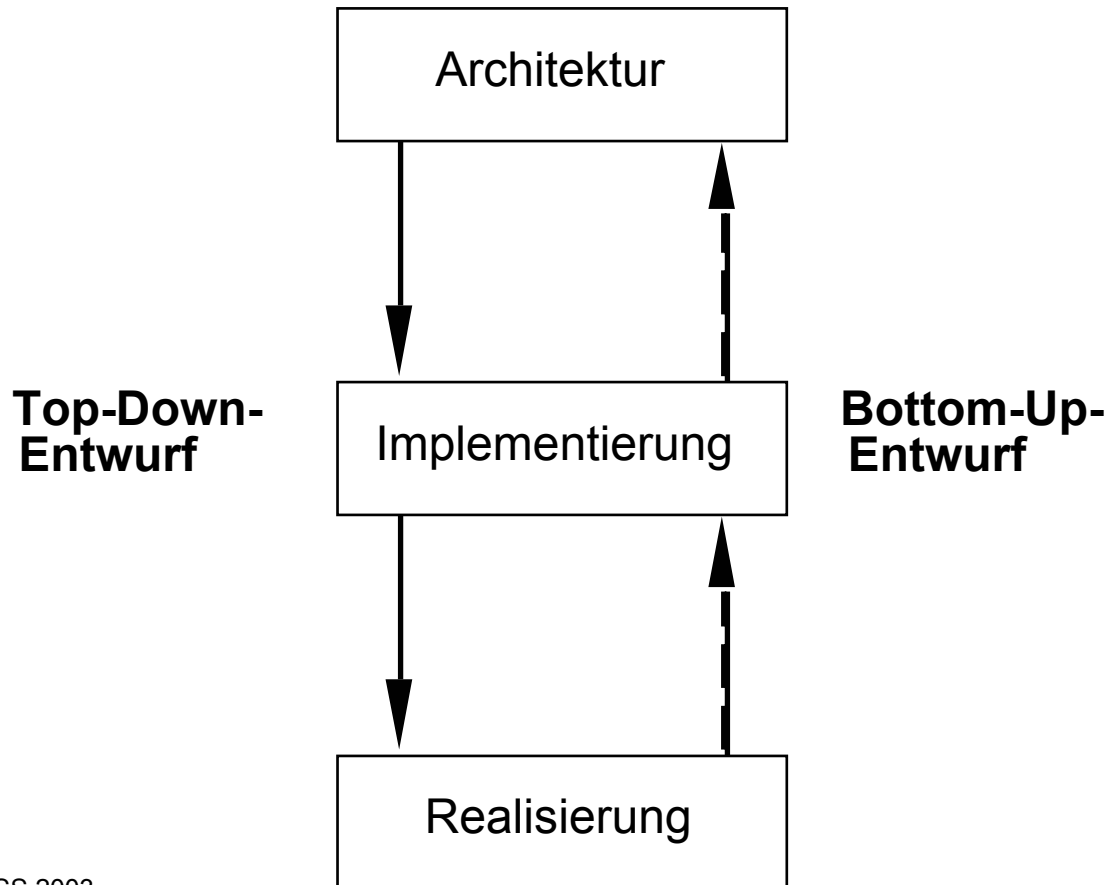
- ◆ entwickelt mit Hilfe der Design-Tools die entsprechenden integrierten Schaltkreise.
- ◆ Synthese-Programme, die aus einer algorithmischen oder Register-Transfer-Beschreibung eine Logik-Beschreibung generieren, werden eingesetzt.

□ Der Rechnerhersteller

- ◆ fügt die Komponenten entsprechend der definierten Rechnerstruktur zu einem Ganzen zusammen.

Rechnerentwurf: Vorgehensweise

- Prinzipielle Vorgehensweise beim Entwurf eines neuen Rechners

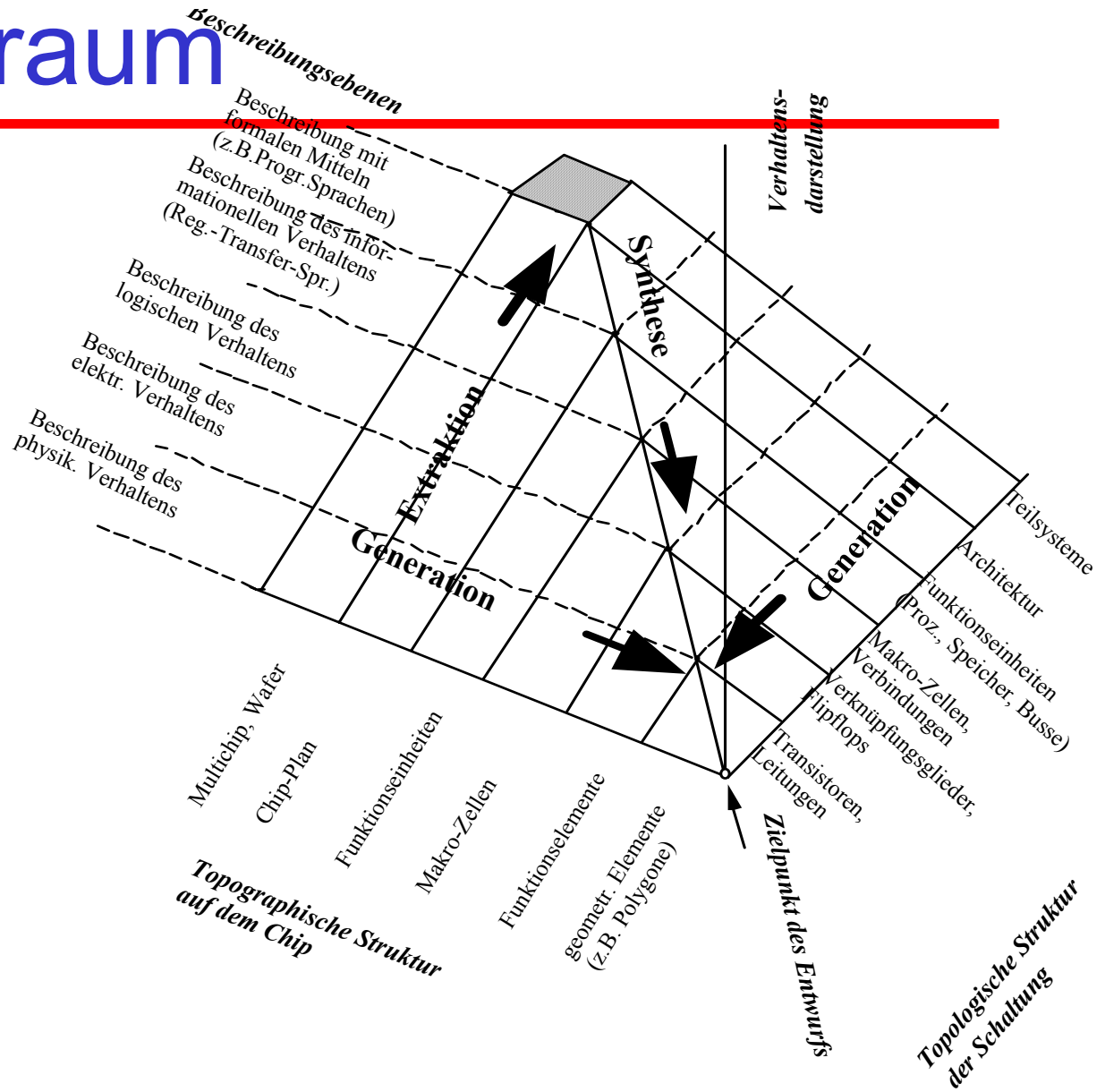


Rechnerentwurf: Vorgehensweise

□ Automatisierung beim Rechnerentwurf

- ◆ Der **Entwurfsprozess** lässt sich in einem **dreidimensionalen Entwurfsraum** mit den folgenden drei Dimensionen darstellen:
 - ◆ Topologische Struktur
 - ◆ Topographische Struktur
 - ◆ Verhaltensdarstellung

Entwurfsraum



Rechnerentwurf

□ **Begriffe des Entwurfsprozesses**

◆ **Synthese:**

- ◆ Erzeugung einer detaillierteren topologischen Struktur aus einer abstrakteren (also weiter oben liegenden) Beschreibung des Entwurfsmodells.

◆ **Generation:**

- ◆ Umsetzung einer Schaltungsstruktur in eine detailliertere Form; sowohl als Generation einer topologischen Struktur wie auch als Generation einer topographischen Struktur.

Rechnerentwurf: Begriffe

- ◆ **Extraktion:**

- ◆ Erzeugung einer abstrakten Beschreibung des Verhaltens aus einer ausführlichen, also tiefer liegenden Beschreibung.

- ◆ **Simulation:**

- ◆ Überprüfung bestimmter (in der Regel durch die Spezifikation geforderter) Eigenschaften anhand einer dafür geeigneten Verhaltensbeschreibung.

Rechnerentwurf: Begriffe

- ◆ Von besonderem Interesse:
 - ◆ Synthese auf der höchsten (abstraktesten) Entwurfsebene

- ◆ Auftretende Probleme:
 - ◆ Auswirkung von Randbedingungen
 - ◆ Formale Spezifikationssprachen für die Architektur- und Systemebene
 - ◆ Qualität des Syntheseergebnisses
 - ◆ Integration verschiedener Werkzeuge

Rechnerentwurf

□ Die Hardware-Beschreibungssprache VHDL

- ◆ VHSIC-Programm der Vereinigten Staaten (*Very High Speed Integrated Circuits*)
- ◆ Standardisierte Hardware-Beschreibungssprache
- ◆ VHDL wurde 1987 IEEE-Standard, mittlerweile in überarbeiteter Form
- ◆ Die verschiedenen Schaltungsbeschreibungen des gesamten Entwurfsablaufs können dargestellt werden – von der algorithmischen Spezifikationen bis hin zu realisierungsnahen Strukturen.

VHDL

- ◆ Ursprünglich als Modellierungssprache nur für die Simulation konzipiert
- ◆ Heute zunehmend auch als Sprache für die Synthese und die Verifikation eingesetzt
- ◆ Eingesetzt zum ASIC- und FPGA-Entwurf
- ◆ Viele Sprachkonstrukte können jedoch nicht in Hardware umgesetzt werden

VHDL

□ VHDL umfasst:

- ♦ alle Elemente einer klassischen Programmiersprache (ADA), erweitert um Konstrukte, die für den Schaltungsentwurf durch Verfeinerung der verschiedenen Domänen entscheidend sind,

□ Chip-Entwurf mit VHDL

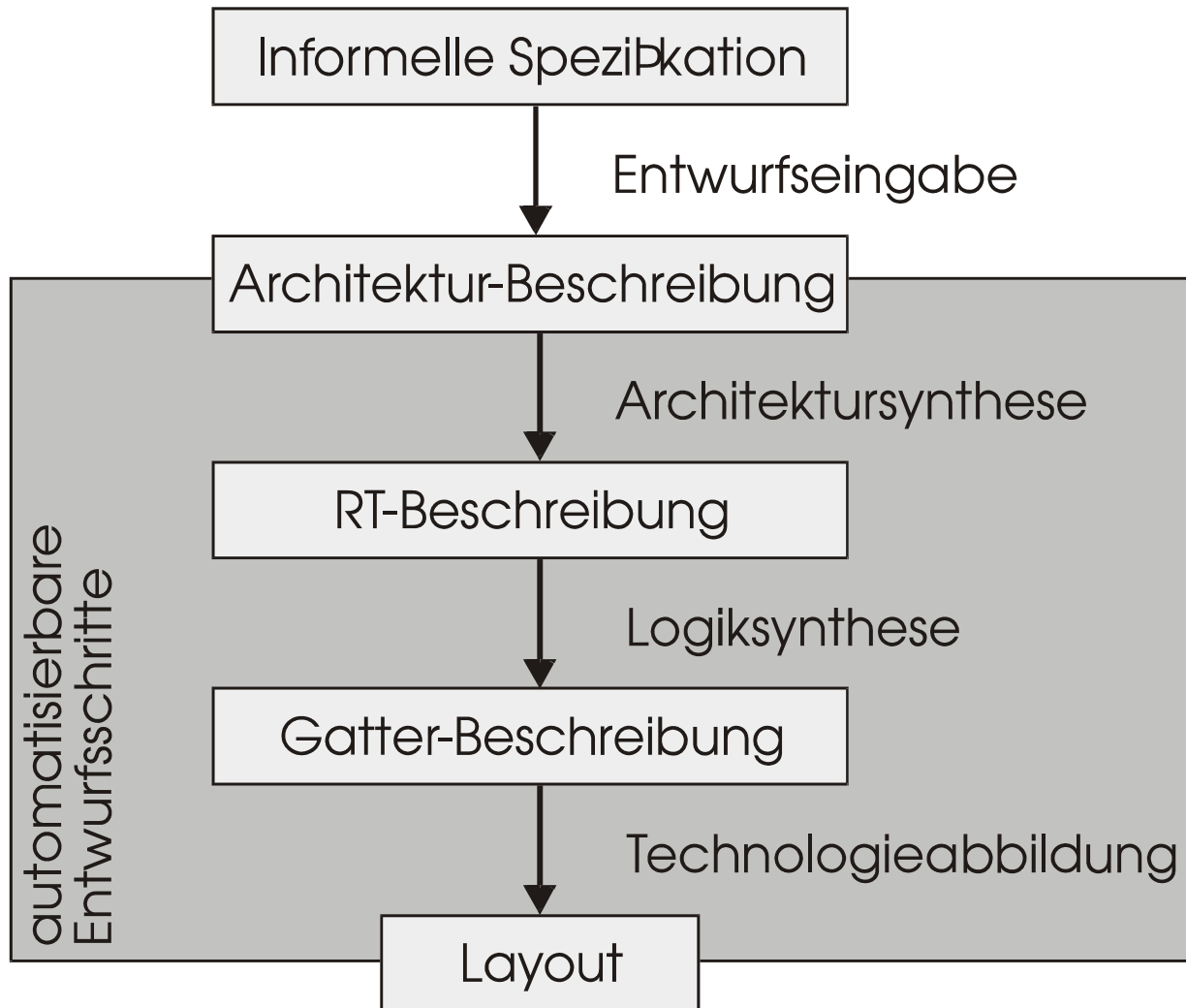
- ◆ Grundlage des Entwurfs ist die **Spezifikation** der Schaltung:
 - ◆ das gewünschte Verhalten,
 - ◆ die Schnittstellen (Zahl und Art der Ein-/Ausgänge)
 - ◆ Vorgaben bezüglich Geschwindigkeit, Kosten, Fläche, Leistungsverbrauch etc.

VHDL: Chipentwurf

□ Entwurfsschritte

- ◆ Verhaltensverfeinerung
- ◆ Strukturverfeinerung
 - ◆ wie eine spezifizierte Funktion durch eine Verschaltung von Komponenten mit einfacherer Funktionalität realisiert werden kann.
- ◆ Datenverfeinerung
 - ◆ Realisierung abstrakter Datentypen durch einfachere Typen.

VHDL: Entwurfsverlauf



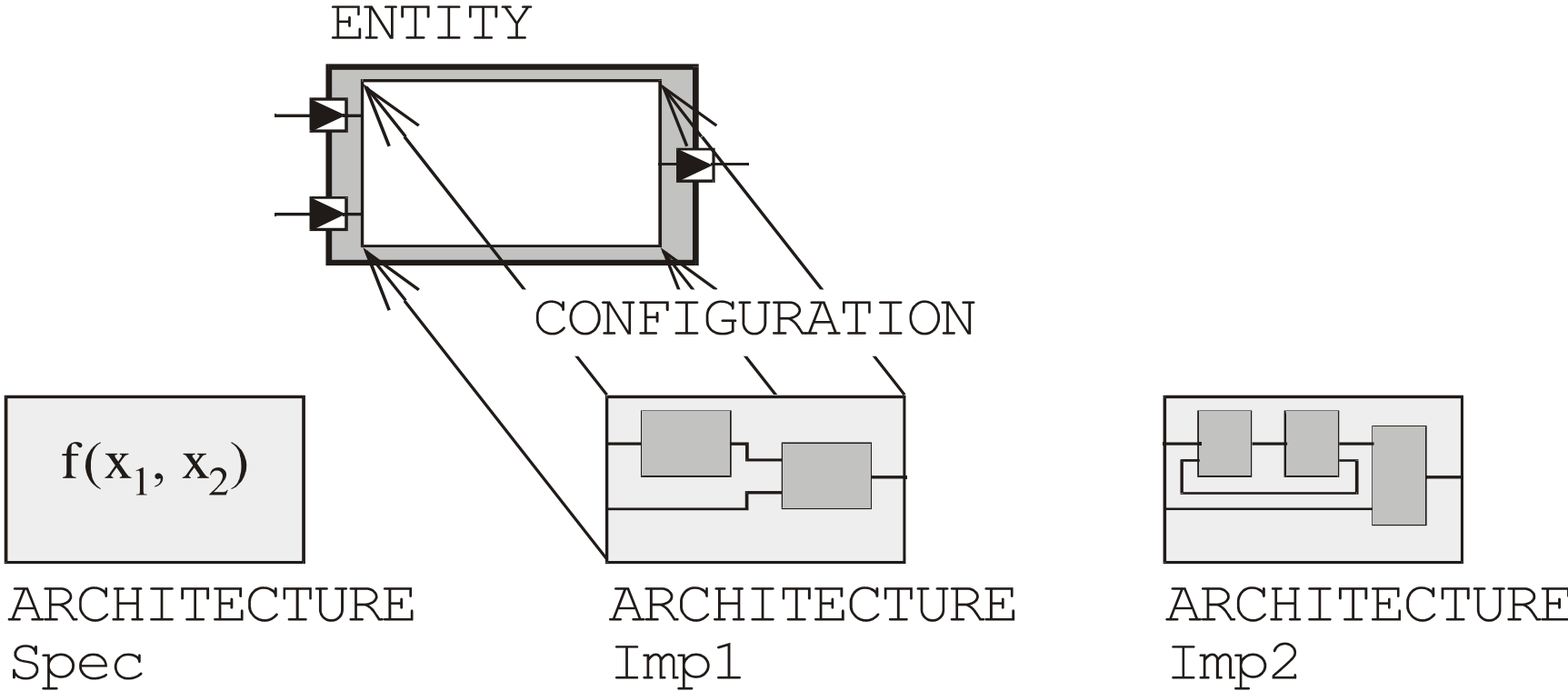
VHDL: Chipentwurf

- ♦ Ein zu entwerfender Chip oder ein Modul ist durch seine Schnittstellen nach außen sowie durch seinen internen Aufbau festgelegt.
- ♦ Der innere Aufbau ist zu Beginn des Entwurfs im allgemeinen nur durch eine funktionale Spezifikation des Verhaltens repräsentiert, die im weiteren Verlauf zu einer strukturellen Implementierung, bestehend aus Submodulen, verfeinert wird.

VHDL

- VHDL erlaubt die getrennte Definition:
 - ♦ der Schnittstellen eines Moduls (**ENTITY**),
 - ♦ der internen Verhaltens- oder Strukturrealisierungen (**ARCHITECTURE**) sowie
 - ♦ der Zuordnung, die angibt, welche interne Realisierung für das Modul aktiv ist (**CONFIGURATION**) und beispielsweise für eine Simulation oder für eine Synthese verwendet wird.

VHDL: Chipentwurf



VHDL: Chipentwurf

□ Beispiel: Schnittstellendefinition eines NAND-Gatters

```
ENTITY Nand2 IS
    PORT (
        X1, X2: IN Std_Logic;
        Y : OUT Std_Logic);
END Nand2;
```

Für einen Baustein darf es nur eine Schnittstellendefinition, jedoch beliebig viele interne Realisierungen (ARCHITECTURE) geben

Vorsicht: der ARCHITECTURE-Begriff von VHDL hat nichts mit der Definition von „Architektur“ vs. „Mikroarchitektur“ bei Prozessoren zu tun

VHDL: Chipentwurf

Beispiel: Schema einer ARCHITECTURE

```
ARCHITECTURE Architecture-Name OF Entity-  
Name IS  
    <Daten-, Komponenten- und  
    Unterprogrammdeklarationen>  
BEGIN  
    <Realisierung, z.B. durch Prozesse>  
END Spec;
```

VHDL: Chipentwurf

□ Strukturbeschreibung einer ARCHITECTURE

- ♦ Die Strukturbeschreibung einer ARCHITECTURE besteht aus verschiedenen Submodulen und deren Verschaltung.
- ♦ Variable Zuordnung einer ARCHITECTURE („Implementierung“) zu einer ENTITY („Schnittstellenbeschreibung“).
- ♦ Die in einer ARCHITECTURE verwendeten Submodule sind im allgemeinen nicht direkte Kopien einer ENTITY. Man verwendet vielmehr „leere Hülsen“ von Modulen, sogenannte *components* oder **Komponenten**.

VHDL: Chipentwurf

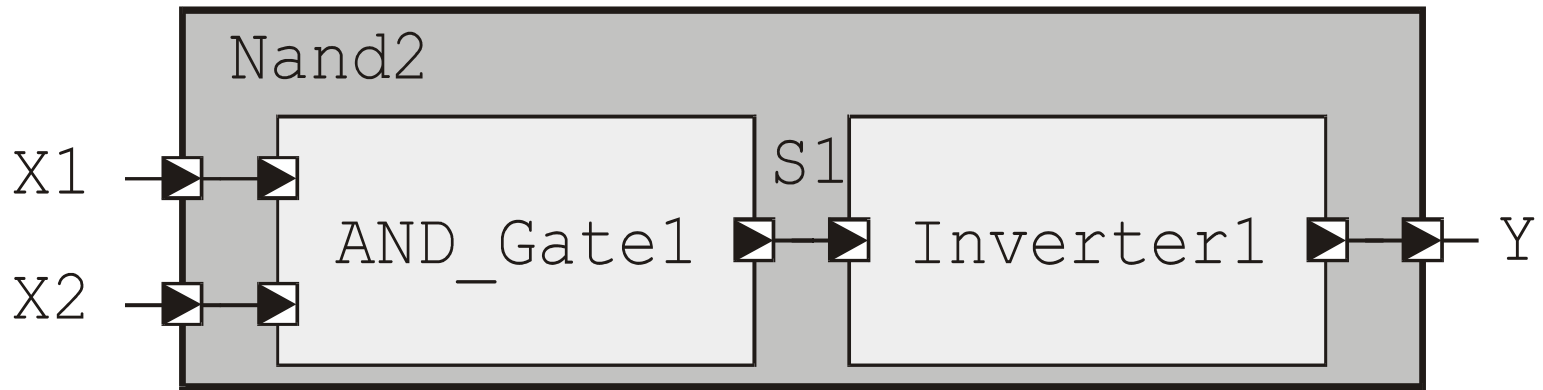
□ Strukturbeschreibung einer ARCHITECTURE

- ◆ Komponenten werden zu Beginn einer ARCHITECTURE bekanntgemacht (***component declaration***).
- ◆ Anschließend werden Kopien (*instances*) der Komponente erzeugt (***component instantiation***) und die Verbindungsstruktur angegeben.
- ◆ Weiterhin Konfigurationen (***component configuration***),
d.h. welche COMPONENT durch welche ENTITY mit welcher ARCHITECTURE realisiert werden soll (separat in einer ***configuration unit***).

VHDL: Chipentwurf

- ❑ **Beispiel:** ein NAND-Gatter soll für die bereits deklarierte ENTITY Nand2 aus einem AND-Gatter und einem Inverter realisiert werden.

- ◆ Modulstruktur von Nand2:



VHDL: Chipentwurf

□ Beispiel: ARCHITECTURE

```
ARCHITECTURE Structure of Nand2 IS
    COMPONENT Inverter
        PORT (
            In1 : IN Std_Logic;
            Out1 : OUT Std_Logic);
    END COMPONENT
    COMPONENT And_Gate
        PORT (
            In1, In2: IN Std_Logic;
            Out1 : OUT Std_Logic);
    END COMPONENT
    SIGNAL S1: Std_Logic;
BEGIN
    And_Gate1 : And_Gate PORT MAP (X1, X2, S1);
    Inverter1 : Inverter PORT MAP (S1, Y);
END Structure;
```

VHDL: Chipentwurf

□ Beispiel: Realisierung der Komponenten Inverter und And_Gate

```
ENTITY An2 IS
    GENERIC Delay : Time;
    PORT (
        X1, X2 : IN Std_Logic;
        Y : OUT Std_Logic);
END An2;
```

```
ENTITY Inv IS
    PORT (
        Y : OUT Std_Logic;
        X1 : IN Std_Logic);
END Inv;
```

VHDL: Chipentwurf

□ CONFIGURATION-Deklaration

- ◆ Möchte man diese Elemente für die Komponenten von Nand2 benutzen, wobei für beide jeweils eine ARCHITECTURE „Behavior“ ausgewählt werden soll, so wird dies durch eine CONFIGURATION vereinbart.
- ◆ In einer CONFIGURATION wird
 - ◆ die Zuordnung von ENTITY und ARCHITECTURE zu konkreten Instanzen jeder COMPONENT gegeben
 - ◆ eventuell eine „Umverdrahtung“ der Signale mit PORT MAP oder eine Verfügung von Parametern mit GENERIC MAP durchgeführt.

VHDL: Chipentwurf

□ CONFIGURATION-Deklaration

- ♦ Diejenigen Schnittstellensignale und Parameter, die in **COMPONENT** und zu verwendender ENTITY in Zahl und Anordnung übereinstimmen, müssen nicht explizit angegeben werden.

VHDL: Chipentwurf

Beispiel: CONFIGURATION

```
CONFIGURATION Nand_Conf OF Nand2 IS

    -- Angabe der ENTITY
    -- Angabe der ARCHITECTURE
    FOR Structure
        FOR Inverter1 : Inverter
            USE ENTITY Work.Inv1 (Behavior);
            PORT MAP (X1 => In1, Y =>
Out1);
        END FOR;
        FOR And_Gate1: And_Gate
            USE ENTITY Work.An2 (Behavior);
            GENERIC MAP (10 ns)
        END FOR;
    END FOR;
END Nand_Conf;
```

Work ist hierbei die Bibliothek, in der Inverter und AND_Gate abgelegt werden.

Die **FOR**-Anweisung gibt hier nicht eine Iterationsschleife an, sondern legt fest, wie bestimmte Einheiten realisiert werden sollen.

VHDL: Chipentwurf

- Die komplette Beschreibung einer Entwurfseinheit besteht aus:
 - ♦ einer ENTITY,
 - ♦ mindestens einer ARCHITECTURE und
 - ♦ mindestens einer CONFIGURATION.

VHDL: Chipentwurf

□ Beschreibung des nebenläufigen Verhaltens

- ◆ Auftreten einer Taktflanke
 - ◆ die Eingangsregister eines Schaltwerks werden neu geladen und damit eine Flut von Signalwechseln erzeugt.
- ◆ Nach einer gewissen Zeit kommt das Schaltwerk in einem stabilen Zustand zur Ruhe und die Werte können an den Ausgängen abgenommen werden.
- ◆ Bei einem Entwurf aus mehreren synchronen Schaltwerken, müssen diese *nebenläufig* beschrieben werden.
Dies geschieht durch die **PROCESS**-Anweisung

VHDL: Chipentwurf

□ PROCESS-Anweisung

```
[process_name :] PROCESS (signal_1, signal_2, ...)  
process declarations  
BEGIN  
    sequential statements  
END PROCESS [process_name];
```

```
[process_name :] PROCESS  
process declarations  
BEGIN  
    sequential statements  
    WAIT-statement  
END PROCESS [process_name];
```

VHDL: Chipentwurf

□ “Prozesse“ in VHDL

- ♦ Die Anweisungen innerhalb eines Prozesses laufen sequentiell ab.
- ♦ Die Aktivierung eines Prozesses kann durch eine Änderung der angegebenen Signale (*sensitivity list*) oder nach der Erfüllung einer Wartebedingung (WAIT-Anweisung) geschehen.
- ♦ Alle aktiven Prozesse werden nebenläufig zueinander abgearbeitet und können weitere Prozesse aktivieren.

VHDL: Chipentwurf

□ “Prozesse“ in VHDL

◆ **sensitivity list:**

- ◆ der Prozess läuft bis zu seinem Ende und wartet auf eine erneute Änderung seiner sensitiven Signale.

◆ **WAIT:**

- ◆ der Prozess läuft bis er auf ein WAIT trifft, dessen Bedingung nicht erfüllt ist,
- ◆ am Ende wird ein Sprung an den Beginn des Programmumpfes durchgeführt.
- ◆ Für die Synthese einer Schaltung sind jedoch keine WAIT-Anweisungen erlaubt!!

VHDL: Chipentwurf

□ Entwurfsunterstützung

- ◆ Moderne Entwicklungstools von SYNOPSIS oder Xilinx bieten Top-level-Werkzeuge zur Entwurfsunterstützung:
 - ◆ **Schematic Editor**: grafische Oberfläche für die Erstellung von Modulen und deren hierarchischem Aufbau
 - ◆ Module lassen sich in Form von Symbolen mit entsprechenden Ein-/Ausgabe Signalen erstellen und die Verbindungen zwischen den Modulen herstellen.
 - ◆ Aus der grafischen Beschreibung wird dann automatisch VHDL-Code erzeugt, dem *nur* noch die Verhaltensbeschreibung hinzugefügt werden muss.

VHDL: Chipentwurf

□ Lehrbücher: Einführung in VHDL

- ♦ Kurzeinführung: Th. Kropf: VLSI-Entwurf - Vorgehen, Methoden, Automatisierung; Thomson-Verlag, TAT 17, Bonn 1995. - vergriffen
- ♦ Weitere deutschsprachige Einführung: G. Lehmann, B. Wunder, M. Selz: Schaltungsdesign mit VHDL; Synthese, Simulation und Dokumentation digitaler Schaltungen; Franzis-Verlag 1994. - vergriffen
- ♦ Weit verbreitetes Standardlehrbuch: R. Lipsett, C. Schaefer, C. Ussery: VHDL: Hardware Description and Design; Kluwer Academic Publishers, 12te Auflage 1993.
- ♦ P. J. Ashenden: The VHDL Cookbook; University of Adelaide, Australia.
- ♦ J. R. Armstrong, F. G. Gray: *Structured Logic Design with VHDL*; Prentice Hall 1993.
- ♦ Eigene News-group für VHDL: comp.lang.vhdl.