


WP ECE SS24

INF-WP Einführung Computer Engineering

Jochen Rust
Michael Schäfers
Tim Tiedemann

VHDL-Teil

Prof.: Michael Schäfers
email: ECE.S24S@Hamburg-UAS.eu

Raum: 780
(spezifisch für diese Veranstaltung) : "MS-Teams"

data: https://users.informatik.haw-hamburg.de/~schafers/LOCAL/S24S_ECE

Wer steht gerade da vorn?

Michael Schäfers

Informatikstudium

+

Promotion

an der TU Braunschweig

Berufstätigkeit (3.1995-3.2002)

Nokia Telecommunications

→ Nokia Networks

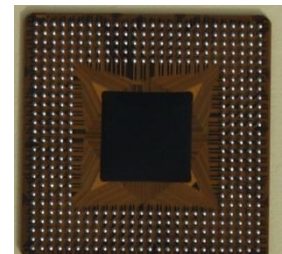
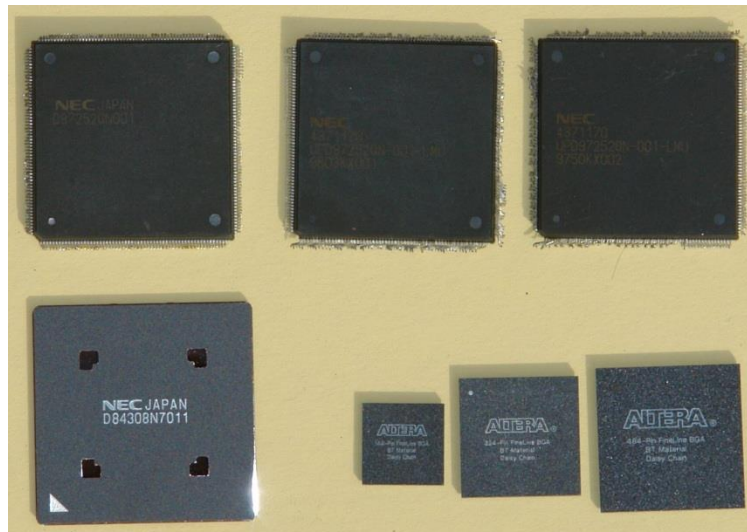
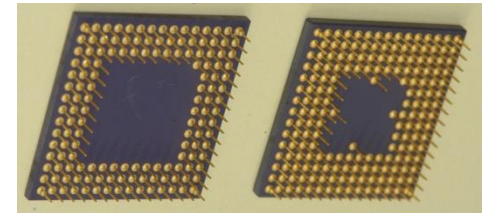
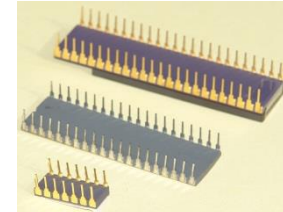
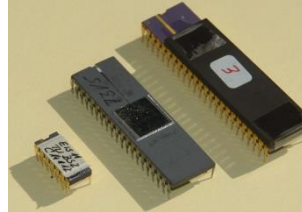
→ NSN (*Nokia Siemens Networks*)

→ NSN (*Nokia Solutions and Networks*)

Düsseldorf

R&D → **ASIC Entwicklung**

HAW Hamburg seit April 2002



<BREAK>

Folien mit **skipped**-Markierung

- Folien, die rechts unten mit

skipped

markiert sind,
wurden übersprungen.

Sie müssen also zum Vorlesungszeitpunkt nicht beunruhigt sein, wenn eine derartige Folie Dinge enthält, die Sie nicht verstehen.

Sofern das "Thema" **nicht** später wieder aufgegriffen wird,
ist es nicht Prüfungs-relevant.

Das kennen Sie schon von mir bzw. P1 ;-)

Wdh. Aus P1
bekannt.

Folien mit **touched**-Markierung

- Folien, die rechts unten mit

touched

markiert sind,
wurden nur angerissen.

Sie müssen also zum Vorlesungszeitpunkt nicht beunruhigt sein, wenn eine derartige Folie Dinge enthält, die Sie nicht verstehen.

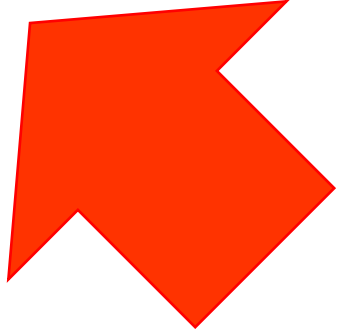
Sofern das "Thema" nicht später wieder aufgegriffen wird,
ist es nicht Prüfungs-relevant.

Das kennen Sie schon von mir bzw. P1 ;-)

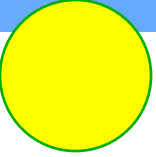
Wdh. Aus P1
bekannt.



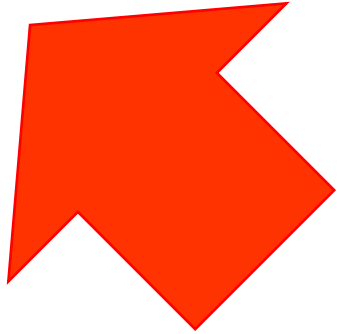
Folien mit grünen Punkt



- Folien mit grünen Punkt sind besonders wichtig



Folien mit gelben Punkt



- Folien mit gelben Punkt sind wichtig

- Die nachfolgenden Folien sind ein Auszug aus der ECE-Veranstaltung des WS20/21
- Damals waren viele Randbedingungen anders - z.B. andere HW
- "Jetzt" in dieser Veranstaltung (SS24) ist vieles überarbeitet & verbessert wurden

VHDL

Vorweg

- VHDL wird nur "vorgestellt"
- VHDL wird **nicht** (vergleichbar zu Java in P1+P2) im Detail erklärt
- Sie werden sich selbstständig in die Sprache einarbeiten müssen
- Informatiker müssen befähigt sein sich in kürzester Zeit in neue Sprachen einzuarbeiten

Grundsätzliches

- VHDL ist eine HW-Beschreibungssprache
(*HDL: **H**ardware **D**escription **L**anguage*)
- VHDL steht für VHSIC Hardware Description Language
- VHSIC steht für Very High Speed Integrated Circuit
- VHDL ist standardisiert im Standard IEEE1076.
Jedoch inzwischen fünf Versionen:
 - IEEE 1076-1987 Standard VHDL kurz **VHDL'87**
 - IEEE 1076-1993 Standard VHDL kurz **VHDL'93**
 - IEEE 1076-2002 Standard VHDL kurz **VHDL2002**
 - IEEE 1076-2008 Standard VHDL kurz **VHDL2008**
 - IEEE 1076-2019 Standard VHDL kurz **VHDL2019**
- VHDL'98 - der Versuch VHDL objektorientiert zu machen - scheiterte

Was ist VHDL?

- eine weit verbreitete HDL
- eine Programmiersprache
- eine Simulationssprache
- eine Spezifikationssprache
- eine Dokumentationssprache
- eine standardisierte (also nicht proprietäre) Sprache
- brauchbar als Input für Synthese

HDLs und Programmiersprachen

- viele Forderungen werden von "normalen" Programmiersprachen erfüllt
- hohe Grad an Parallelität ist Problem für "normale" Programmiersprachen
- Echtzeit-Sprachen sind ausgerichtet auf das Einhalten realer Zeiten (Echtzeit) und nicht auf exakte Auflösung der zeitlichen Diskretisierung (z.B. Femtosekunden) ohne Echtzeitanforderungen genügen zu müssen
- HDLs sind gewöhnlich an Programmiersprachen angelehnt
 - VHDL an ADA (*und ADA an PASCAL*)*
 - Verilog an C
 - *SystemC an C*
- * *Wer PASCAL oder Modula-2, Oberon, ADA kann, dem kommt sehr viel in VHDL bekannt vor*

gängige HDLs

- 2 HDLs haben sich durchgesetzt und werden noch lange Zeit nebeneinander her existieren
 - VHDL
stärker vertreten in Europa
stärker auf den höheren Ebenen und "mehr" normale Programmiersprache
wurde als HDL entworfen
 - Verilog
stärker vertreten in USA und Japan
(verstärkt gegenwärtig durch internationale Firmen seine "Verbreitung")
stärker auf den niederen Ebenen – sehr guter Gate-Level-Simulator
hat sich erst zur HDL entwickelt
- es wird immer wieder versucht auf höherer Ebene eine Sprache zu etablieren
 - SystemC ist ein Beispiel hierfür
- Traum einer Sprache sowohl für HW als auch SW
Idee:
Nachfolgende Tools generieren daraus automatisch die SW- und die HW-"Teile"

Geschichte von VHDL

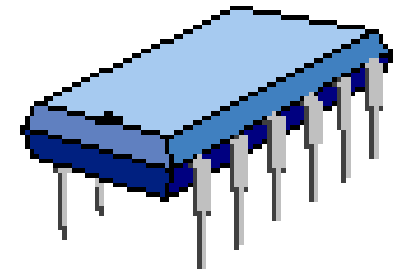
- US DoD forcierte Entwicklung und Einsatz von VHDL, die HDL sollte
 - standardisiert sein und geeignet für
 - Spezifikation,
 - Simulation (also vom Computer lesbar und ausführbar),
 - Dokumentation,
 - und damit insbesondere Austausch, Wiederverwendung und Wartung von Schaltungen gewährleisten
- VHDL basiert bewusst auf ADA
- 1981 formaler Start der VHDL-Entwicklung
- 1985 VHDL Version 7.2
- 1986 IEEE Standard 1076 ("1. VHDL" – VHDL'87)
- 1993 überarbeiteter IEEE Standard 1076 ("2. VHDL" – VHDL'93)
im wesentlichen ein Bug-Fix
- objektorientiertes VHDL immer wieder in der Diskussion
(z.B. als "VHDL'98" – kam aber nicht)
- es folgten VHDL2002, VHDL2008 und schließlich 2019

ein kurzer Ausflug

**Abstraktionsebenen
(für Schaltungen)**

**ein Überblick zur
Hintergrund-Information**

Begriffsklärung



ASIC

- “**A**pplication **S**pecific **I**ntegrated **C**ircuit”
- keine klare Definition / einheitliche akzeptierte Auffassung von "ASIC"
- *(grundsätzlich)* ein speziell für einen Kunden oder eine Anwendung gefertigter integrierter Schaltkreis

Abstraktionsebene

- **wichtiges** Hilfsmittel für die Entwicklung **komplexer** Systeme
- ist definiert durch eine **reduzierte Sicht** auf diejenigen Elemente, die für diese spezielle Ebene von **Bedeutung** sind. Auf jeder Abstraktionsebene werden daher nur die hier wichtigen Teilaspekte betrachtet
- je abstrakter die Ebene, desto “höher” die Ebene
je mehr Details, desto “niedriger” die Ebene
- wichtig bei der Diskussion von Abstraktionsebenen: Für welchen **Zweck** diese Ebenen eingesetzt werden. Hier ist es der Entwurf von ASICs
- in der Praxis “Vermischung der Dimensionen” Verhalten, Struktur und Physik/Geometrie



Funktionale Ebene (Behavioral Level)

- die Funktionale Ebene ist wichtig für die **Spezifikation**
- es wird nur das **Verhalten** beschrieben und keine Struktur
- Konzentration auf das “WAS” unter Vernachlässigung des “WIE”
- Funktion wird beschrieben durch **charakteristische** Variablen und deren Werteverläufen über die Zeit
- Zeitmodell: Kausalität
- beobachtbare Werte: beliebige Werte im frei definierbarem Wertebereich
- Arbeitsmittel: Dokumente in natürlicher Sprache, Ein-/Ausgabetabellen, Skizzen, (VHDL)
- Beispiele: Spezifikation, Pflichtenheft

touched

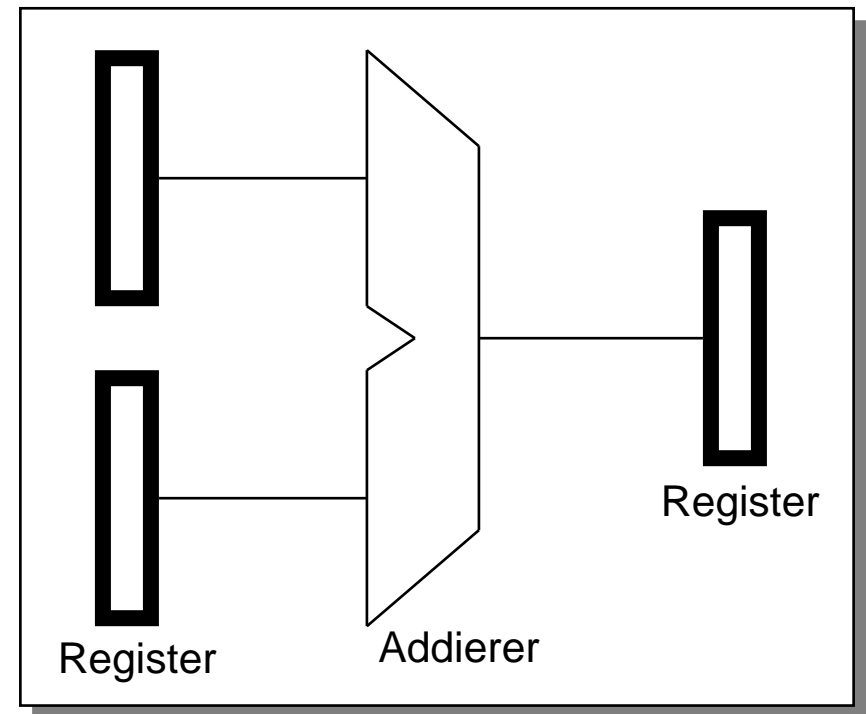
System-Ebene (System Level)

- erster Bezug zur späteren Struktur der Realisierung (Hardwarepartitionierung); Funktionale Einheiten (Blöcke, ASICs) werden bestimmt und beschrieben
- erste Überlegungen zur relativen Lage der Komponenten auf dem Chip (Floorplan)
- Zeitmodell: Kausalität oder diskrete Realzeit
- beobachtbare Werte: Vektoren von Bits mit Interpretation
- Arbeitsmittel: Dokumente, Ein-/Ausgabetabellen, **VHDL**
- Beispiele: Architekturplan, Blockdiagramm

touched

Register-Transfer-Ebene (RTL)

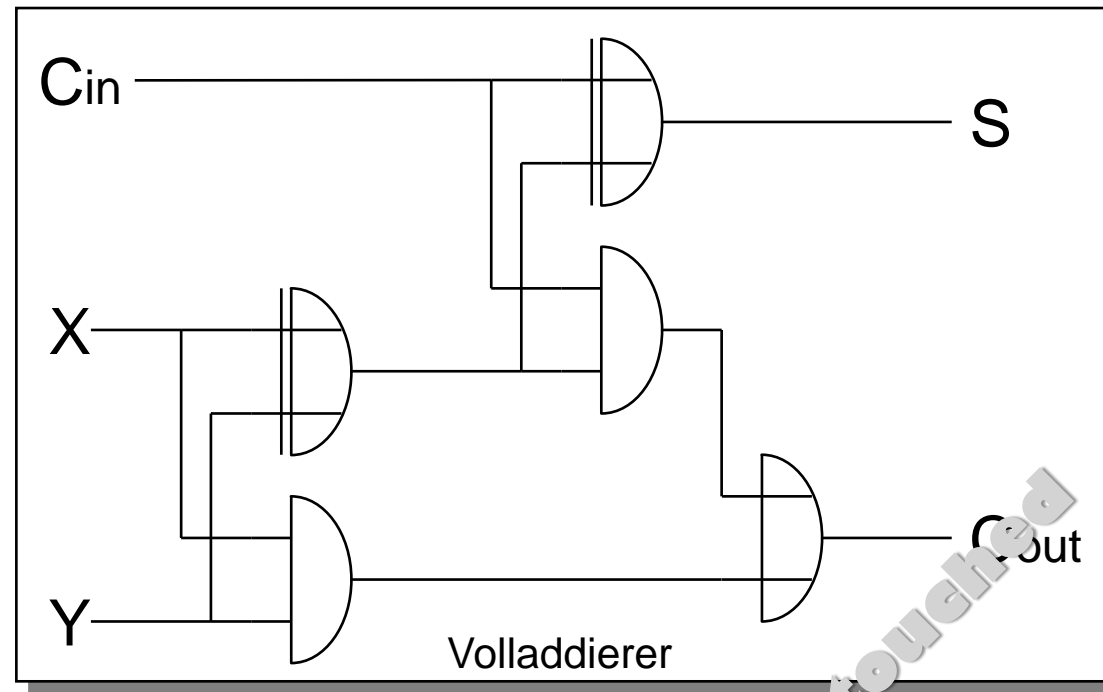
- Ziel ist die **Trennung** der **kombinatorischen** und der **sequentiellen** Logik
- 1. mögliche Schnittstelle zum Halbleiterhersteller
- endgültige Hardwarestruktur wird sichtbar
- **wichtige** Ebene des ASIC-Entwurfs
- Zeitmodell: diskrete Realzeit (Takte)
- beobachtbare Werte: Vektoren von Bits
- Arbeitsmittel: **VHDL**, **VERILOG**



Logik- oder Gatter-Ebene (Gate Level)

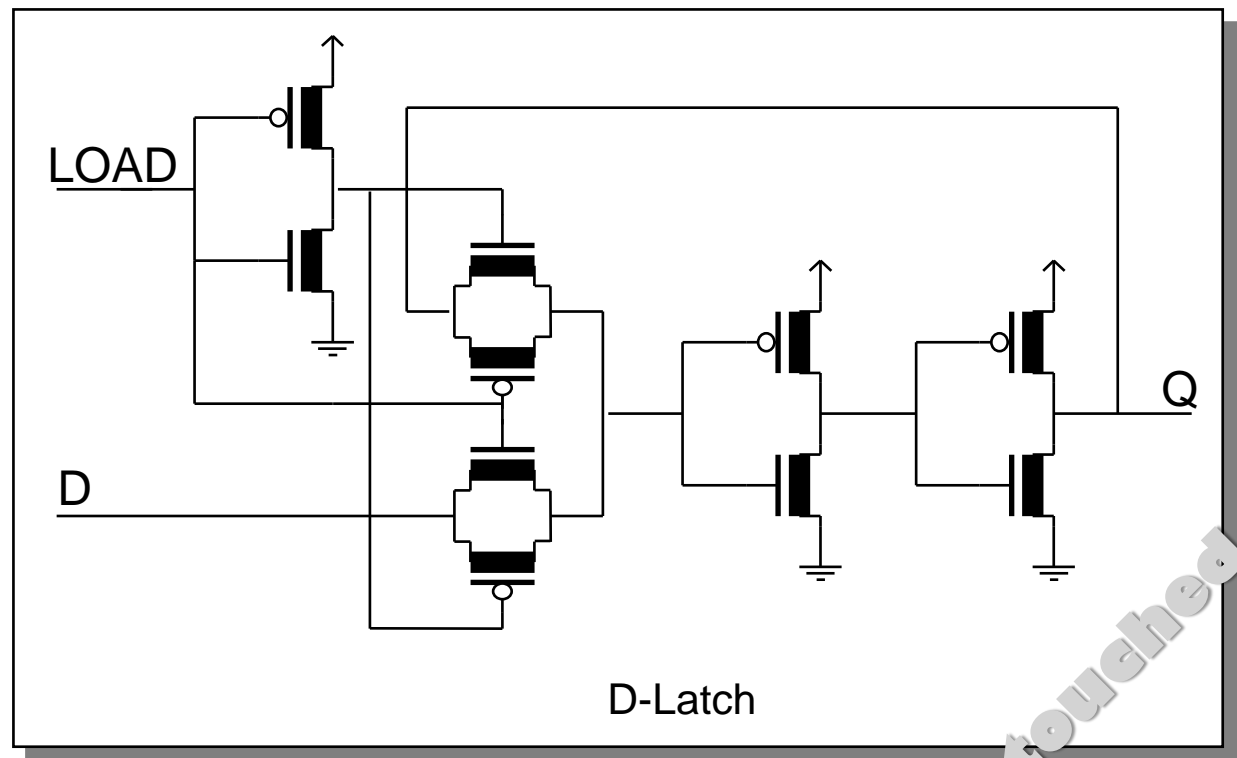
- Schaltungen auf Netzen von bekannten Gattern (AND, OR, INV, NAND, NOR, XOR, D-FF, ...)
- übliche Schnittstelle zum Halbleiterhersteller
- “Arbeitsebene” für viele Flow-Schritte (Timing-Simulation, backannotation, ATPG, Fehlerüberdeckung, P&R, ...)
- früher Schaltplaneingabe (Schematic Entry)

- Zeitmodell: kontinuierliche Realzeit
- beobachtbare Werte: Bi-Tupel (logischer Wert, Stärke)
- Arbeitsmittel: VERILOG, **VHDL** (VITAL)



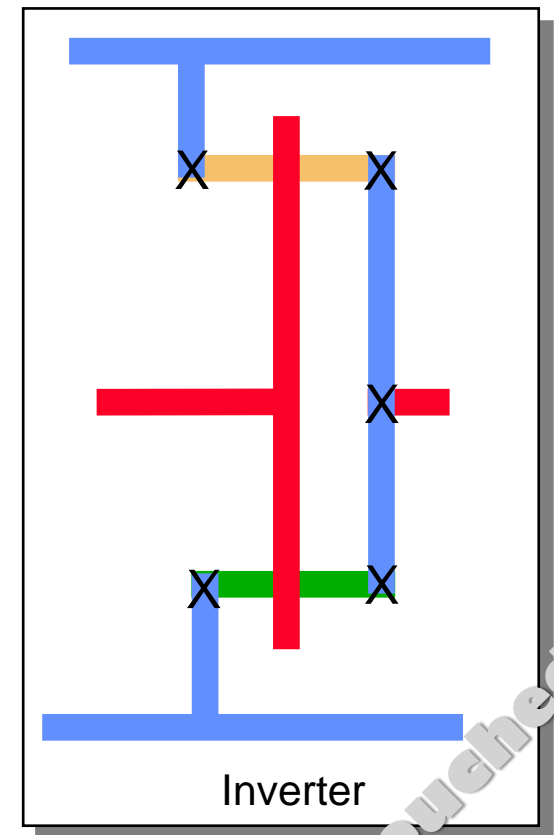
Schaltkreis-Ebene (Switch Level)

- **Transistoren**, Widerstände, Kondensatoren
- Zeitmodell: kontinuierliche Realzeit
- beobachtbare Werte: kontinuierlicher Wertebereich für Spannungen, Ströme (analog) oder Bi-Tupel (digital)
- Arbeitsmittel:
SPICE (analog)
aber auch Verilog (digital)



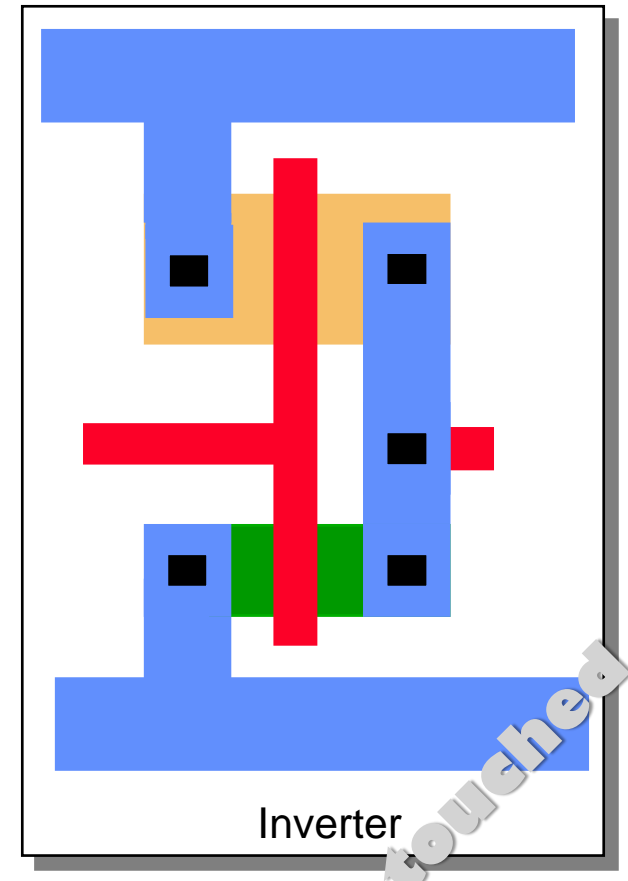
Symbolische Layout-Ebene

- **Stick-Diagramm** (“erweiterte Schaltkreis-Ebene”)
- isolierte Schichten (Layer) in **farbiger** Darstellung
- verschiedene Leitungen eines Layers dürfen sich nicht kreuzen
- zusätzliche Topologieinformation
=> logische Anordnung der Komponenten wird bestimmt
- nicht maßstabsgetreu
- Zeitmodell: kontinuierliche Realzeit
- beobachtbare Werte: kontinuierlicher Wertebereich
für Spannungen, Ströme
- Arbeitsmittel: SPICE, Editor/Papier



geometrische Layout-Ebene

- **maßstabsgetreue** Vergrößerung der schichtenweisen Strukturen im fertigen Chip
- geometrische Entwurfsregeln (Mindestabstände und Mindestbreiten)
- simuliertes Layout ist die Schnittstelle des Full-Custom-Designers
- geometrische Layout wird in textuelle Beschreibungsform (GDS2, CIF2.0) gewandelt. Nach dem Tape-Out werden hiermit die Masken für die Chipfertigung erstellt
- mittels Extraktor Überführung in Schaltkreisebene
- Zeitmodell: kontinuierliche Realzeit
- beobachtbare Werte: kontinuierlicher Wertebereich für Spannungen, Ströme
- Arbeitsmittel: SPICE



Zusammenfassung

- Einführung von Abstraktionsebenen ermöglicht
 - das **Verständnis** komplexer Systeme
 - den Entwurf **komplexer** Systeme
- | | | |
|---------|---|--|
| hoch | ↑ | Funktionale Ebene (Behavioral Level) |
| | | Systeme-Ebene (System Level) |
| | | Register-Transfer-Ebene (RTL) |
| | | Logik- oder Gatter-Ebene (Gate-Level) |
| | | Schaltkreis-Ebene (Switch-Level) |
| | | Symbolische Layout-Ebene |
| niedrig | ↓ | Geometrische Layout-Ebene |
- mit dem Fortschreiten der Integrationsdichte werden die implementierten Funktionen komplexer
- mit der steigenden Komplexität nimmt die Bedeutung der höheren Ebenen weiter zu
- ASIC-Entwicklung wird Software-Entwicklung ähnlicher

Anforderungen an HDLs

- formale, korrekte Beschreibung eines HW-Modells
- für Mensch und Maschine leicht lesbare Form
- leicht und schnell erlernbar
- möglichst viele Abstraktionsebenen können, auch gemischt, vorkommen (*abgesehen von der Layout-Ebene – hierfür z.B. EDIF, CIF, GDS-II*)
- Unabhängigkeit von der verwendeten Technologie (CMOS, GaAs, TTL, ...)
- Unterstützung eines modularen Entwurfs und der Wiederverwendbarkeit von Teilschaltungen
- gängige SW-Entwurfsverfahren (top-down, bottom-up, library-based, ...) sollen unterstützt werden
- Standardisierung der HDL soll Austausch der Modelle (und damit Simulation auf verschiedenen Rechnersystemen) ermöglichen
- ausreichende Geschwindigkeit des Simulators auch bei sehr großen und komplexen Modellen



Begriffe

"Normale" Programmiersprache:

- Der Compiler übersetzt den Code in lauffähigen Code - die Applikation.
- Diese SW-Applikation ist das Ziel des SW-Entwicklungs-Prozesses.

HDL:

- Der Compiler übersetzt den HDL-Code in Simulations-Code, der zur Verifikation der (späteren) HW dient
- Die **Synthese** (der "Silicon-Compiler") erzeugt aus dem HDL-Code "etwas" (konkret eine Netzliste), das ein bestimmender Schritt zu Erzeugung der HW ist.
- Die HW-Applikation ist letztlich das Ziel des Entwicklungs-Prozesses.



Unterschiede

Das Programmieren mit einer HDL ähnelt/"folgt" in vielen Punkten dem Programmieren mit einer "normalen" Programmiersprache. Sehr vieles aus dem Software-Engineering gilt auch in der HW-Entwicklung mit einer HDL.

In einigen Punkten ist es jedoch grundverschieden.

Bei der Programmierung mit einer "normalen" Programmiersprache ist das Ziel:

- performanter Code
- wartbarer Code

Bei der Programmierung mit einer HDL ist das Ziel:

- performante HW
- "kleine" HW
- wartbare HW
(und da diese per Synthese aus VHDL erzeugt wird, wartbarer VHDL-Code)
- Dokumentation der HW-Funktion
- akzeptabel performater Simulations-Code



Unterschiede (Beispiel)

Bei der Programmierung mit einer "normalen" Programmiersprache

- werden "Dinge" nur berechnet, wenn Sie gebraucht werden
- die Dynamik der Programm-Ausführung führt den jeweiligen Code dann nur aus, wenn das "Ergebnis" auch wirklich benötigt wird
- ist Zeit kein Thema
(Ausnahme Echtzeit-Programmierung - hier wichtig "Reaktionszeit der SW")

HW ist statisch, es kann **nicht** spontan in einer Berechnung neue HW in einem IC ergänzt werden.

Bei der "Programmierung" mit einer HDL

- werden "Dinge" auch(bzw. immer) berechnet, obwohl Sie (gerade) gar nicht benötigt werden, weil die HW statisch existent ist und dies Synthese und Wartbarkeit erleichtert
- ist Zeit eine wichtige Größe
haben Dinge ("Ereignisse") ein Bezug zur Zeit (z.B. HW liefert nach 2,537ps Ergebnis)

Bemerkung:

Das dynamische Rekonfigurieren von HW wird z.B. durch FPGA unterstützt.

Unterschiede

- die HDL/Sprache selbst und der zugehörige Simulator gehen in der "Praxis" Hand in Hand.
Häufig wird hier nicht sauber unterschieden
- Formal gibt es 3 Schritte
 - analyzing
 - elaboration
 - simulation
- In der Praxis scheint es meist nur zwei Schritte zu geben
 - compilation
 - loading design
 - simulation
- VHDL-Beschreibungen ("Programme") laufen auf einem Simulator.
Deswegen wird gesagt, dass VHDL keine Programmiersprache ist.



Es wird unterschieden zwischen

- Code der HW beschreibt, also später synthetisiert werden soll (also Synthese-fähig ist) und entsprechende Eigenschaften haben muss. Dies ist später häufig RTL-Code. Solcher Code beschreibt auch eine **Struktur** ("Topologie von HW-Elementen")
- Code der nicht das Ziel hat konkrete HW zu beschreiben
- Das könnte sein
 - Test-Code zur Verifikation (Stichwort "TestFrame")
 - Referenz- oder Spezifikations-Code
- Solcher Code ist typischer Weise **Behavioral Code** und beschreibt (ausschließlich) ein Verhalten
- Behavioral Code muss sich nicht von normalen Software-Code unterscheiden
- Eine mögliche Struktur, die beschrieben wird, erhebt keinen Anspruch auf die spätere HW abgebildet zu werden.

Die HDL: **VHDL**

Grundsätzliches

- VHDL ist nicht Case Sensitive
- VHDL ist stark typisiert (*strong typing*) sehr ähnlich Java
- Unter VHDL müssen Dinge immer erst deklariert werden bevor sie benutzt werden
- VHDL trennt zwischen "Deklarations-Teil" und "Anweisungs-Teil"
- PASCAL-Kenntnisse sind für VHDL nützlich - werden aber nicht vorausgesetzt
Es macht keinen Sinn extra PASCAL für VHDL zu lernen
Bemerkung richtet sich nur an "die", die PASCAL mal gelernt haben

Typen

skalar/enumeration

- CHARACTER
- BIT
- BOOLEAN
- INTEGER
- REAL
- TIME

- STD_LOGIC
- STD_ULOGIC

array

STRING
BIT_VECTOR

STD_LOGIC_VECTOR
STD_ULOGIC_VECTOR

access, file

Datentypen

- CHARACTER für ASCII-Zeichen
- BIT für binäre Werte (0 oder 1) - in der HW denkt man in Bit
werden wir aber nicht/kaum nutzen - sondern std_logic
- BOOLEAN für boolsche Werte (true oder false)
- INTEGER für ganzzahlige numerische Werte ("meist" 32 Bit)
- REAL für Floating Point Werte
- TIME nimmt Zeit auf (später Arbeitsbegriff "V-Zeit")
"Dinge"/Ereignisse werden einer Zeit zugeordnet
- STRING für Zeichenketten (Achtung! VHDL ist nicht Java ;-))
- BIT_VECTOR ist ein ARRAY über BIT
werden wir aber nicht/kaum nutzen - sondern std_logic_vector

Datentypen

- ACCESS ist ein Pointer (Achtung! VHDL ist weder Java noch C ☺)
- FILE ist ein Datentyp für Dateien
- Für Testbenches (TestFrames) bzw. behavioral Code macht FILE vereinzelt Sinn
- ACCESS ist eher exotisch
- Beide sind nicht für synthetisierbaren Code geeignet

"Objekte"

- CONSTANT - echte Konstanten
- VARIABLE - kennzeichnet echte primitive Variablen
Wertzuweisungen an Variablen erfolgen sofort
- SIGNAL - eine "andere" "Variablen"-Art
Wertzuweisungen an Signale erfolgen immer verzögert
Modelliert Laufzeiten - HW braucht immer Zeit
"Gibt es so nicht in einer normalen Programmiersprache"
- FILE - hat Datentyp und "Objekt"-Eigenschaften
- Beispiel:

```
variable COUNTER : integer;
```

Assignments

2 Zuweisungen

- := Variable Assignment

Wert-Zuweisung erfolgt **unmittelbar sofort** an Variable

Variablen bzw. Variable Assignments werden innerhalb von Prozessen genutzt für algorithmische temporäre "Größen"

- <= Signal Assignment

Wert-Zuweisung erfolgt **verzögert** an Signal

Signale repräsentieren HW-Verbindungen und werden genutzt um "Werte" zwischen den Prozessen/Komponenten auszutauschen

- Bemerkung
In VHDL ist die Zeit zweidimensional