Rechnerorganisation

0 Vorbemerkungen

0.1 Begriffe

- *Rechnerorganisation*: unter Rechnerorganisation versteht man den (hierarchischen) Aufbau sowie den zweckbestimmten Zusammenschluss der (Teil-) Einheiten eines Rechners.

- *Implementierung*: Konzept (Idee) → Umsetzung → Reales Gebilde
  Design (Entwurf) → Implementierung → Technisches System (Schaltkreis, Schaltung)

  Unter Implementierung versteht man die Umsetzung eines Designs in ein reales technisches System durch Verwendung bestimmter Technologien, Verfahren usw.
  Damit einher geht eine Vervollständigung und Konkretisierung des Designs.

- *Rechnerarchitektur*: Die Rechnerarchitektur ist eine abstrakte Sicht auf ein Rechnersystem, bei der konkrete Ausbildungen zum Beispiel aufgrund gewählter Technologien, Logiken oder Organisationsformen einfach nicht gesehen werden.
  Nur die einem Konzept (Idee) ursprünglich zugrundeliegenden Formen und Funktionen werden betrachtet.
  Eine Architektur kann durch verschiedene Organisationsformen implementiert werden. (Rechnerarchitektur ist abstrakter als Rechnerorganisation)

0.2 Ziele

- Kennenlernen grundlegender Baueinheiten und Organisationsformen vom elektronischen Digitalrechnern (von-Neumann-Typ)
- Grundlagen des EDA-gestützten* Logikentwurfs (VHDL) für programmierbare Bausteine (CPLD, FPGA)
- am Beispiel der Implementierung eines sehr einfachen „2-Bit-Rechners“ (bzw. „4-Bit-Rechners“) auf einem Entwurfsbord.

0.3 Aufbau – Infos – Literatur

Infos: http://www.tu-chemnitz.de/informatik/RA/educ/lv-ro.html

Aufbau: 2-3 Vorlesungen → 1 zentrale Übung → 1 Lab
(6-7 Zyklen)

* EDA *engl. Electronic Design Automation* (computergestützter Entwurf)

0.4 Prüfung/Tests

- unbewerteter Test
- Abschlussklausur (Schein, >50%)
- Bestandteil der Vordiplomprüfung

1 Einleitung

... und einige historische Entwicklungen

Antike	Rechnen mit Steinen	
Mittelalter	„Leggeld“ auf Rechentuch	
vor 800 Jahren	Ziffernrechnen „nach Methode der Inder“	
vor 450 Jahren	A. Ries	„Rechnen auf der Linilen“ „Rechnen mit der Feder“ (schriftliches Rechnen)
vor 400 Jahren	Mechanisierung des Ziffernrechnens Schickard – Additionsmaschine	
	... Pascal Leibnitz	
vor 150 Jahren	C. Babbage	„Father of Computing“ Difference Engine *Analytical Engine* (mill, lager, control)
1936	A. Turing	(universelle Berechenbarkeit)
vor 60 Jahren	K. Zuse	Z1,...Z4 - mechanisch - digital - frei programmierbar (aber: kein datenabhängiger Sprung)
...	ENIAC	Röhrenrechner (ca. 18000 Röhren)
...	von Neumann	– Rechnertyp / -vorschlag – Rechnerarchitektur Prinzip eines praktisch gebrauchsfähigen Allzweckrechners mit universeller Berechenbarkeit vorgeschlagen.
	→ IAS-Computer*	

* IAS *engl. Institute of Advanced Study* baute einen ersten Rechner des von-Neumann-Typs

Halbleiterentwicklung

1947	Schockley	Transistor
1958	Kilby (TI*)	mehrere Transistoren / Komponenten auf einem Chip – IC**
1959	R. Noyce	Planartechnologie***
1968	Noyce/Moore	→ Intel (*Int*egrated *El*ectronics)
1971	1.Mikroprozessor	
	→ Miniaturisierung	
	→ *"Moore's Law"* – alle 2 Jahre Verdopplung der Transistoren auf einem Chip	

* TI – Texas Instruments

** IC *engl. Integrated Circuit* Integrierter Schaltkreis

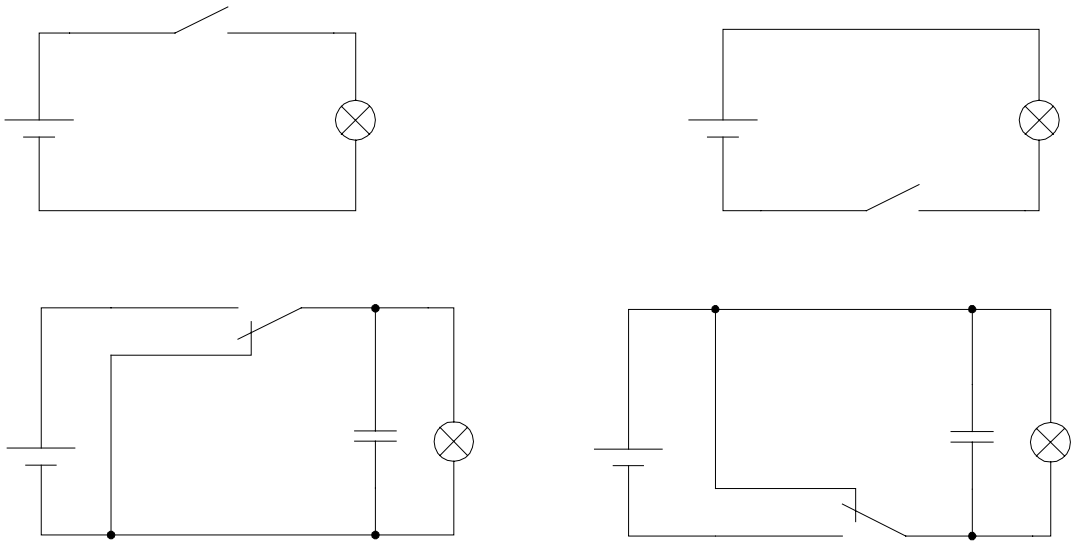
*** Planartechnologie – Technologie zur schichtweisen Fertigung von ICs

2 Einige Grundlagen zur Elektronik

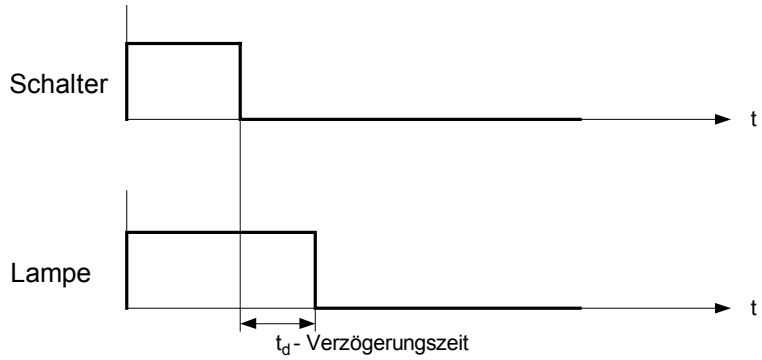
2.1 Begriffe

highakitv
(hochaktiv)

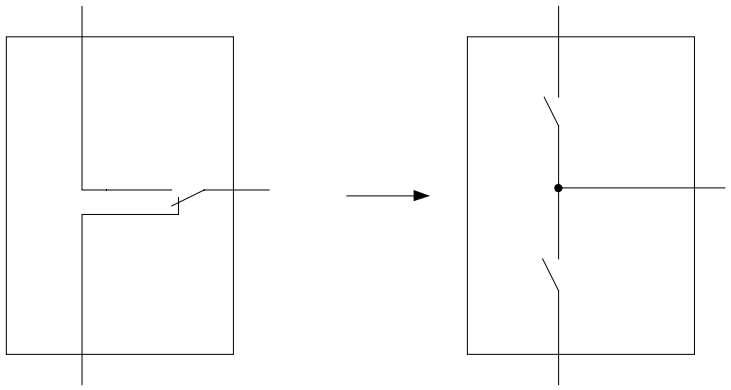
lowaktiv
(tiefaktiv)



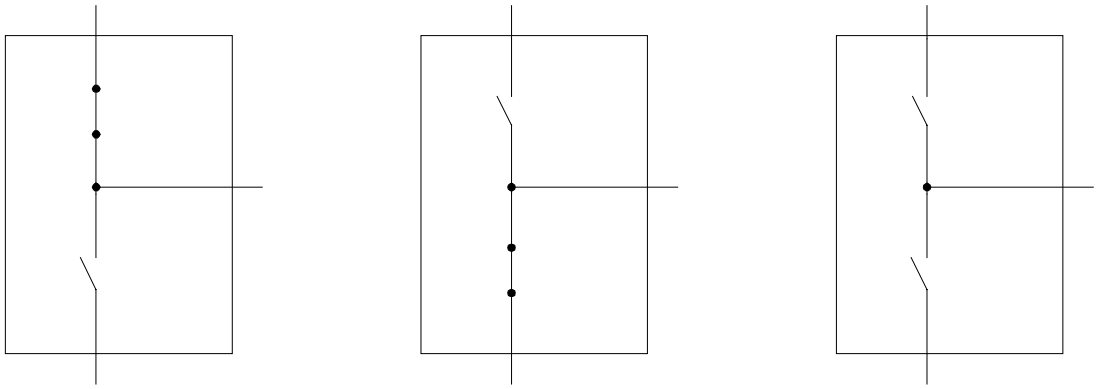
In allen elektronischen Schaltungen sind Kapazitäten (im Bild als Kondensator dargestellt) vorhanden. Um die Verzögerungszeiten zu minimieren wird deshalb der aktive Pol des Verbrauchers entweder mit High- oder Low-Potential verbunden.



Statt eines Umschalters kann man auch zwei einfache Schalter verwenden die wie folgt zusammen geschaltet werden:



Mit dieser Anordnung lassen sich 3 verschiedene Zustände am Ausgang (Anschluss auf der rechten Seite) erzeugen:



High = "1"

Low = "0"

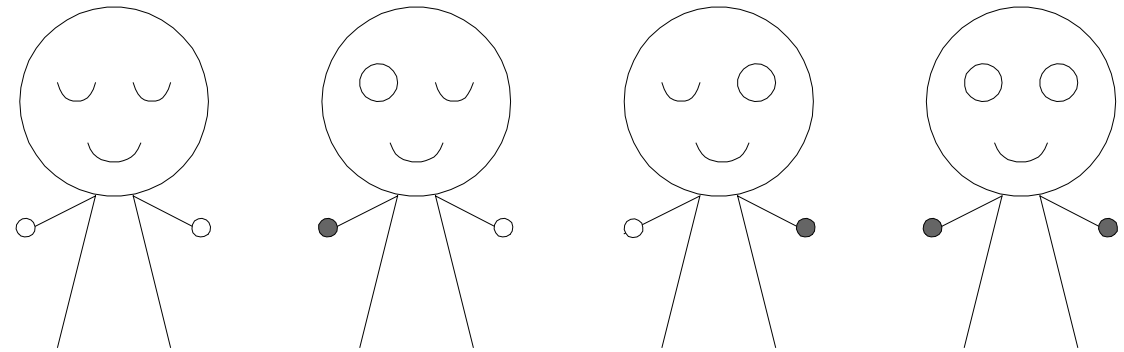
Tristate = "Z"

3 Entwurf einer einfachen Logikschaltung

° Struppi

am Beispiel dieser kleinen Puppe, werden wir einige Schaltungsvarianten betrachten.

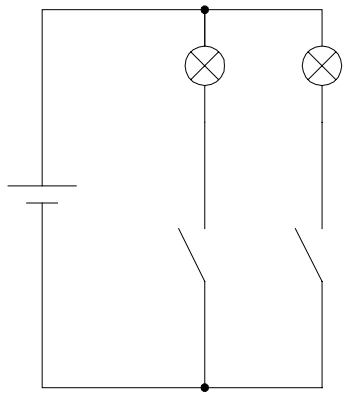
3.1 Schaltungsvariante 1



○ Schalter nicht betätigt

● Schalter betätigt

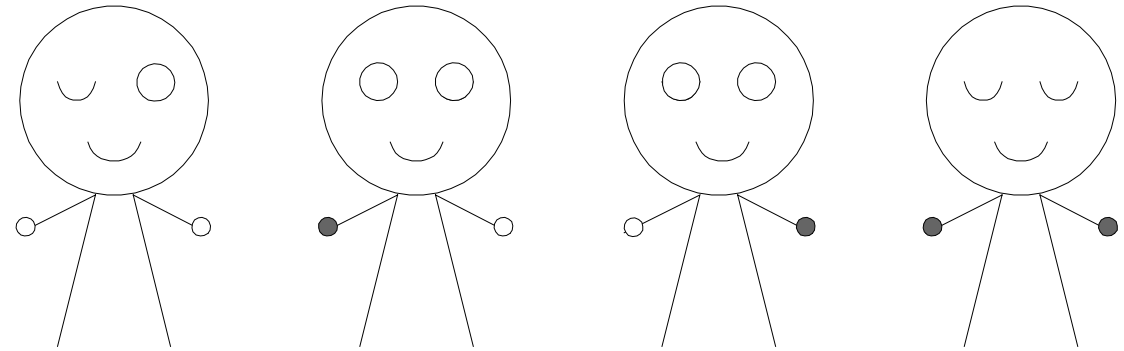
Schaltung:



3.2 Schaltungsvariante 2

Aufgabe:

Es soll eine logische Schaltung für folgendes Verhalten des „Struppi“ entwickelt werden.



○ Schalter nicht betätigt

● Schalter betätigt

logische Zusammenhänge

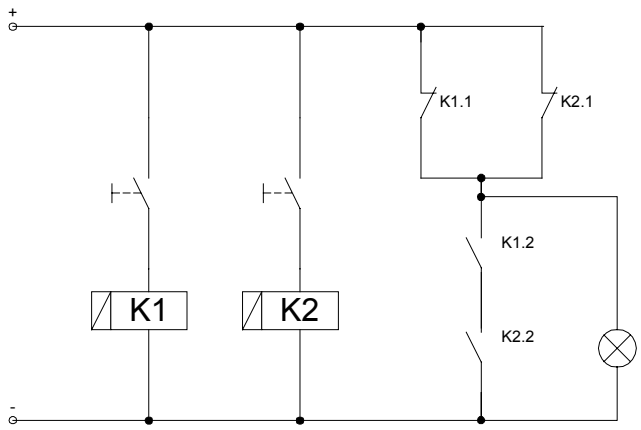
linke Hand (a)	rechte Hand (b)	linkes Auge (u)	rechtes Auge (v)
0	0	0	1
0	1	1	1
1	0	1	1
1	1	0	0

→ u = a XOR b

→ v = a NAND b

Schaltung für rechtes Auge (NAND-Logik)

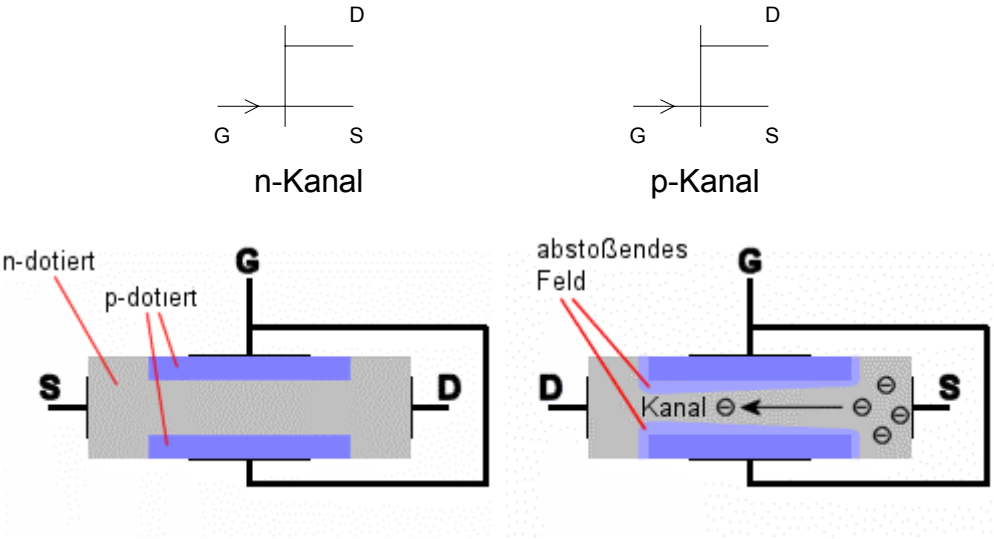
° Mechanische Steuerung von Kontakten (Schaltern)



° Elektronische Steuerung

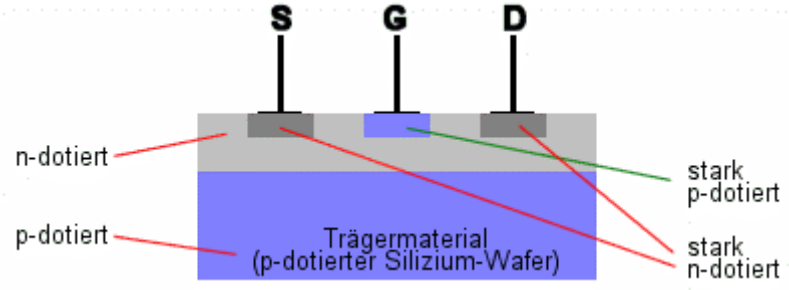
Für die elektronische Steuerung kommen auch sogenannte Feldeffekttranisitoren (FET) zum Einsatz. Es gibt 6 verschiedene Typen, dessen Aufbau und Wirkungsweise hier kurz erwähnt werden soll.

Sperrschicht FET



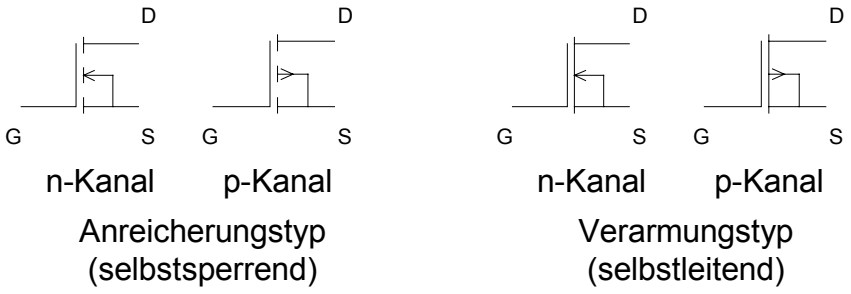
Hier im Bild dargestellt der schematische Aufbau eine n-Kanal Sperrschicht FET.

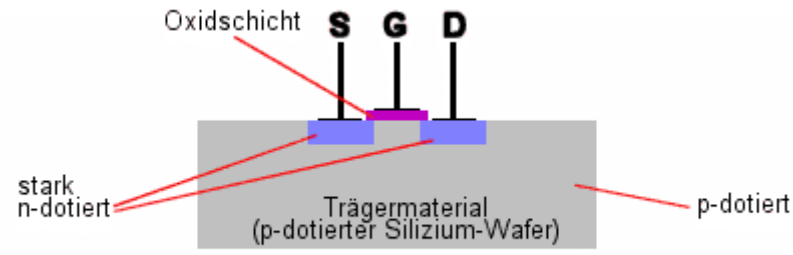
Wird zwischen Gate (G) und Source (S) eine Spannung $U_{GS} < 0$ angelegt so verbreitert sich die Sperrschicht am P-N-Übergang und behindert den Stromfluss durch den Kanal abhängig von der Größe der Spannung.



Sperrschicht-FET in Planartechnologie

Isolierschicht FET (IGFET)





Hier im Bild dargestellt der Aufbau eine n-Kanal Isolierschicht FET (Anreicherungstyp)

Wird zwischen Gate (G) und Source (S) eine Spannung $U_{GS} > 0$ angelegt so werden die freien Elektronen des Trägermaterials (Substrat) in Richtung Gate bewegt wo sich dadurch eine n-Kanal aufbaut, der einen Stromfluss von Drain (D) nach Source (S) ermöglicht.

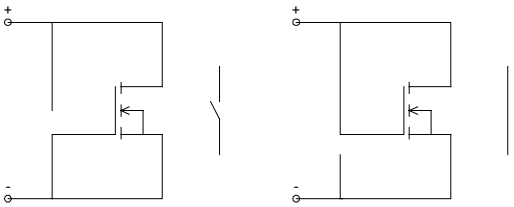
Wichtig für die Funktion ist das eine leitende Verbindung zwischen Source (S) und dem Trägermaterial (Substrat /Bulk) besteht, die meist herstellerseitig bereits vorhanden ist.

Ähnlich ist die Funktionsweise eines IGFET vom Verarmungstyp. Er besitzt bereits einen sehr dünnen Kanal (daher auch selbstleitend), der mit Hilfe der Gate-Source-Spannung verbreitert oder verschälter werden kann.

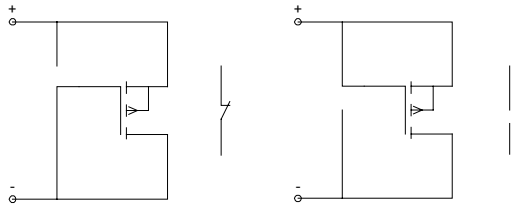
Für die Verwendung in digitalen Schaltungen sind der N-Kanal- und P-Kanal-Anreicherungstyp von besonderer Bedeutung.

Schaltverhalten

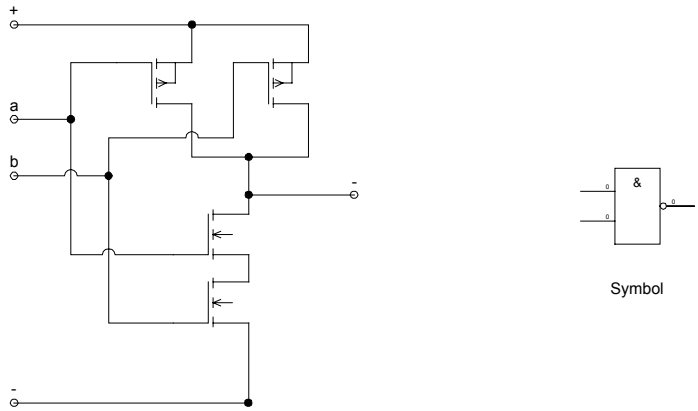
n-Kanal (Anreicherungstyp)(selbstsperrend)



p-Kanal (Anreicherungstyp)(selbstsperrend)

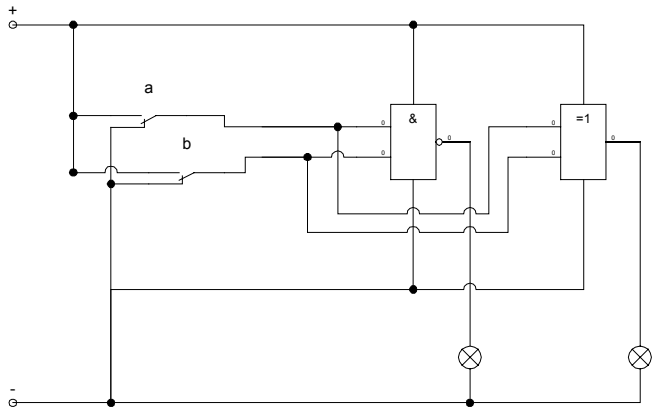


° NAND-Gatter mit Komplementären Transistoren (CMOS*)



Symbol

° Logikschaltung Struppi Schaltungsvariante 2



° 2 Implementierungsbeispiel

(a) Nichtprogrammierbare Logik
z.B. SN7400 4er-NAND
... einige Schaltkreise, Batterie, Schalter, Draht

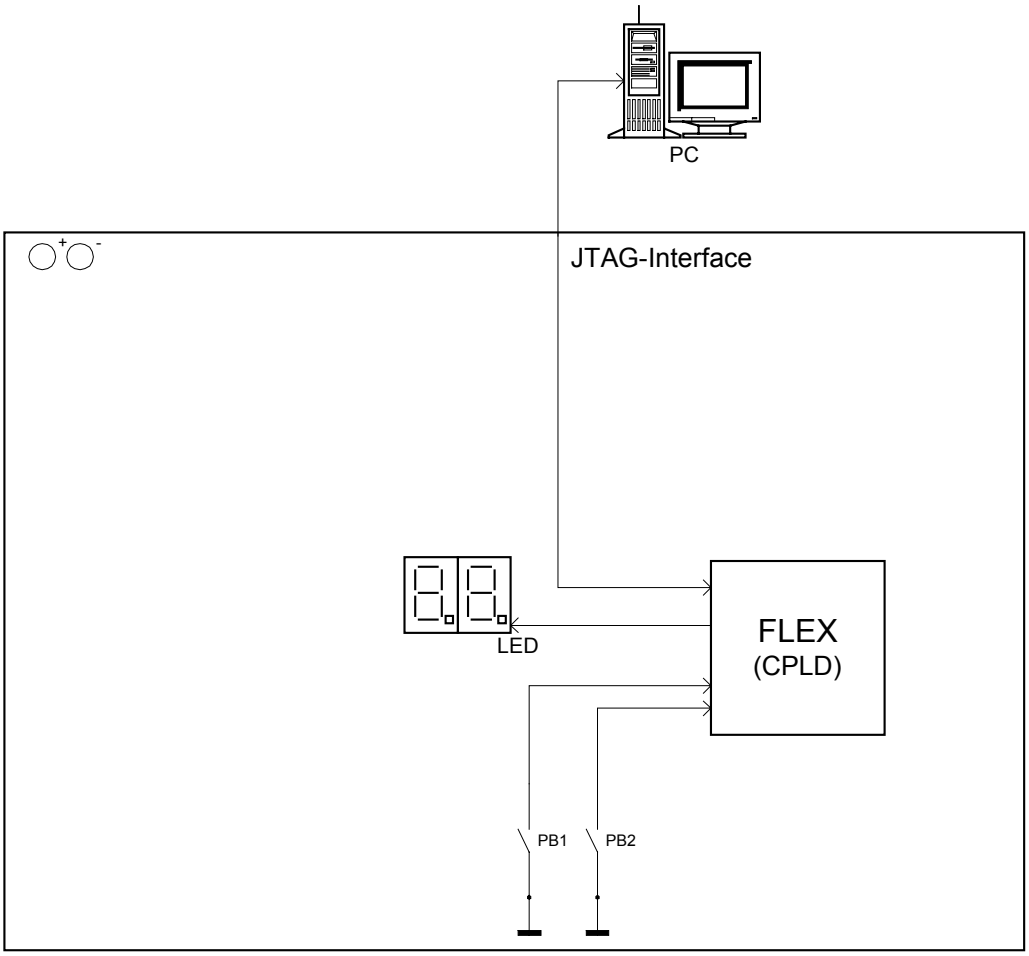
(b) Halbvorgefertigte Schaltungen verwenden, die einfach und schnell *ergänzt* und *verändert*, das heißt (um-)konfiguriert, werden können.
z.B. ein sogenanntes Rapid-Prototypig-Board
z.B. ALTERA

Ergänzen:
- durch steckbare Verbindungsleitungen
- zusätzlich aufsteckbare Logik

Verändern:
- durch Schaltkreise, die viele Logikelemente (-blöcke) enthalten und deren Zusammensetzung (Verbindungen) konfigurierbar ist.
- die Einstellung erfolgt durch Anlegen bzw. Senden einer entsprechend kodierten Bitsequenz an entsprechende „Stelleingänge“
z.B. Laden eine JTAG-Files
- die „Stellinformation“ (JTAG-File) wird z.B. mit Hilfe eine Programms - eines Compilers bzw. eines Synthesestools (MAX+PlusII) - aus der Logikbeschreibung (Grafik, Text) automatisch erzeugt.

* CMOS *engl. Complementary Metal Oxid Semiconductor*

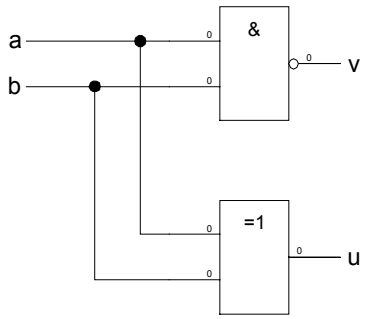
° Auszug aus dem ALTERA-Board



Design Entwurf Eingabe	→	Compilierung	→	Download des Programmierfiles (JTAG-File)

4 Traditioneller Entwurf mit Standardlogik

„Struppi-Logik“ (Beispiel)



***Problem**: Welche Baugruppen beziehungsweise Logiktechnologie soll verwendet werden?*

Logik-Technologien

CMOS	V	Standard CMOS
	HCT	High Speed CMOS Technologie
	AHCT	Advanced High Speed CMOS Technologie
TTL	TTL	Transistor-Transistor-Logik
	FTTL	Fast-Transistor-Transistor-Logik
Schottky	S	Schottky
	LS	Low-Power-Schottky
	ALS	Advanced-Low-Power-Schottky
	ABT	Advanced Bi CMOS Technologie

Integrationsgrade

Wir arbeiten hauptsächliche mit niedrig- bis mittelintegrierten Logiken

SSI	Small-Scale-Integration	(bis	10 Gatter pro Chip)
MSI	Medium-Scale-Integration	(bis	100 Gatter pro Chip)
LSI	Large-Scale-Integration	(bis	1000 Gatter pro Chip)
VLSI	Very-Large-Scale-Integration	(über	1000 Gatter pro Chip)
(ULSI	Ultra-Large-Scale-Integration)		

***Problem**: Welche Baureihen sind verfügbar? Welche Schaltkreise sind vorhanden?*

CD 4xxx CMOS-Reihe

Listenauszug:

4007 ..... 3 komplementäre Transistorpaare

4069 ..... 6x Inverter (Negatoren)

4011 ..... 4x 2er NAND

4012 ..... 4x 2er NOR

4030 ..... 4x 2er XOR

4081 ..... 4x 2er AND

4071 ..... 4x 2er OR

4077 ..... 4x 2er XNOR

4013 ..... 2x D-Flip-Flop

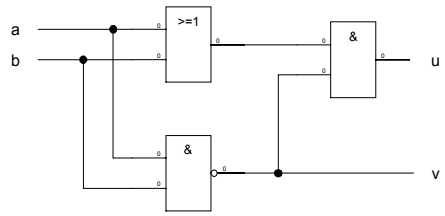
4024 ..... 7-stufiger Binärzähler

Weitere Informationen zu Logikschaltkreisen bei Texas-Instruments
siehe http://www.texas-instruments.com

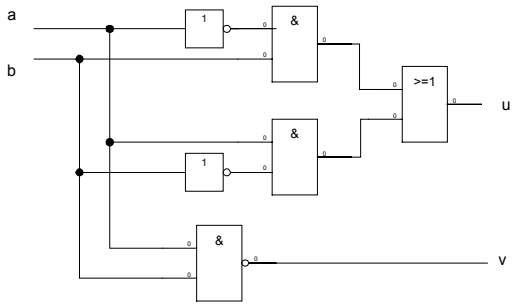
***Problem**: Welche IC's sollen wir verwenden? Aus welchen IC's das Ganze synthetisieren?*

Es gibt mehrere Lösungen! – Welche ist optimal? – Was heißt optimal für unsere Aufgabenstellung?

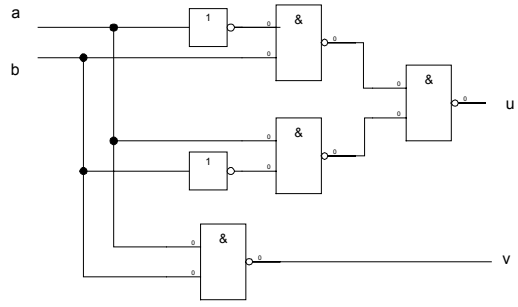
(a)



(b)

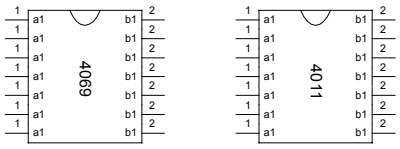


(c)

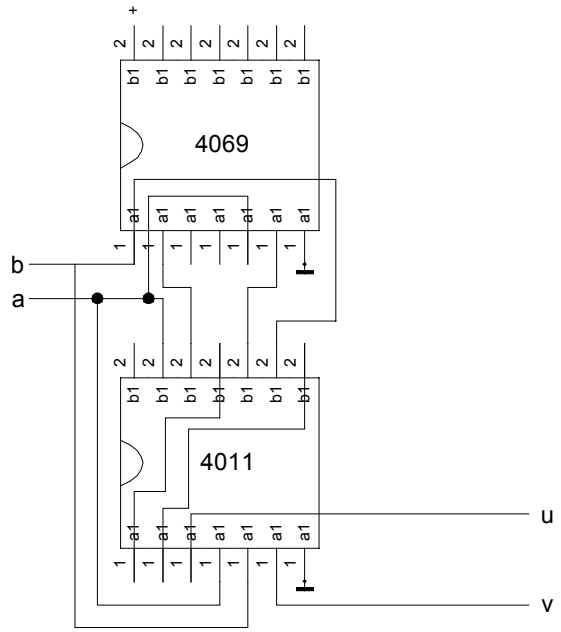


Realisierung der Variante (c)

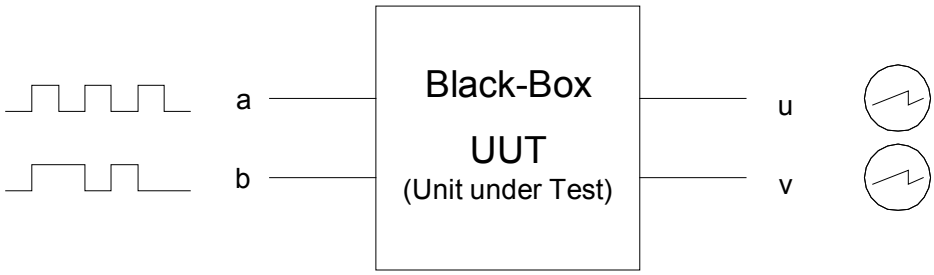
Wir verwenden 2 Schaltkreise ein 6er Negator und ein 4er NAND



Problem: *Wie die Funktionalität unter den IC's verteilen? (***Binding***)*
*Wohin die IC's auf der Leiterplatte setzten? (***Place***)*
*Welche Verbindungswege wählen? (***Routing***)*



Testen (Schaltung)



Signalverläufe vorgeben
Testvektoren erzeugen
Stimulusgenerator

Signalausgänge beobachten
und vergleichen
Response Monitor

Ziel: - eventuelle Herstellungsfehler (Löten, Verdrahten) erkennen und beseitigen.
- eventuelle logische Fehler beseitigen!

Frage: Hätte man nicht schon durch Simulation auf der Basis der logischen Schaltung Fehler früher erkennen können?

Antwort: Ja!

Frage: Muss man zur logischen Simulation die *Netzliste* verwenden, das heißt die textuelle Beschreibung (Liste) der Gatter, Flip-Flops und deren Verbindung?

Antwort: Nicht unbedingt!
Eine Beschreibung des Verhaltens des Systems (Black-Box-Prinzip) welche nicht auf die tatsächlich realisierte Verschaltungsstruktur eingeht, ist oftmals leichter und übersichtlicher.

Problem – Wie beschreiben?

Zum Beispiel eine Hardwarebeschreibungssprache (HDL*)

HDL's:
- VHDL – Very High speed Integrated Circuit Description Language
- Verilog
- System C

…
- ABEL-HDL

…

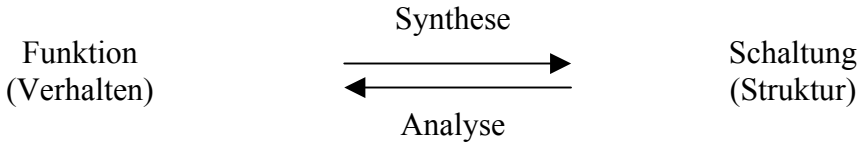
VHDL wurde zur Beschreibung, insbesondere zu Simulationszwecken entwickelt.

→ Beispiel: Verhaltensbeschreibung

```
u <= a XOR b
v <= a NAND b
```

→ Idee: Verhaltensbeschreibung nicht nur zur Beschreibung und Simulation verwenden. Aus der Verhaltensbeschreibung automatisch eine passende Struktur- bzw. Schaltungsbeschreibung ableiten, das heißt synthetisieren.

Begriffe: Analyse, Synthese

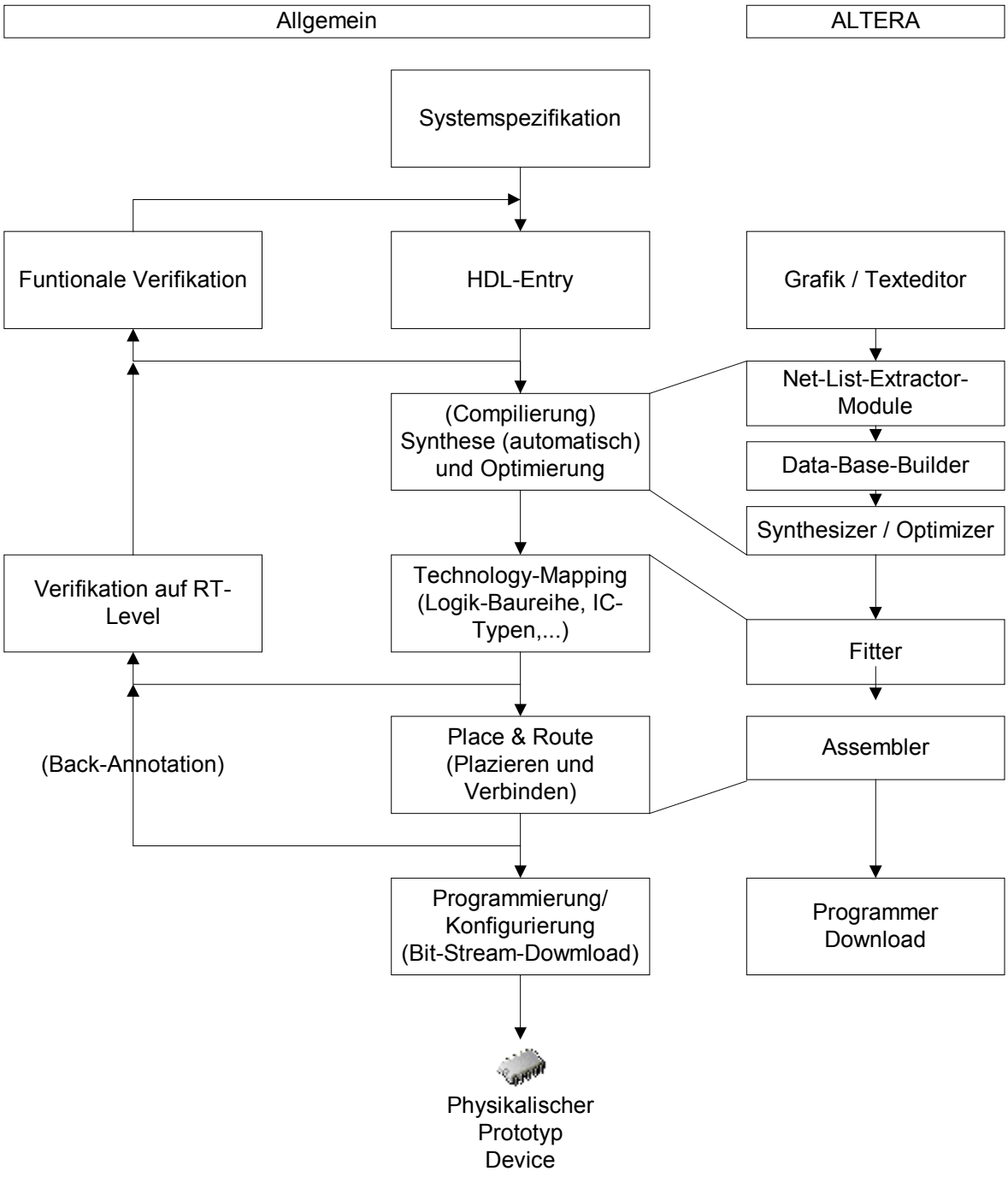


Entwurfsebenen (Design-Levels)

System-Ebene (System-Level)	Ganze Rechner, SUB-Systeme, CPU, Speicher, I/O, ...
Register-Transfer-Ebene (RTL)	Register, Zähler, Coder, Decoder, Multiplexer, Demultiplexer, Teiler, ...
Logik-Ebene (Gatelevel)	Gatter, Flip-Flops, Verbindungsleitungen
Schaltkreisebene (Circuit-Level)	

* HDL *engl. Hardware Description Language* Hardwarebeschreibungssprache

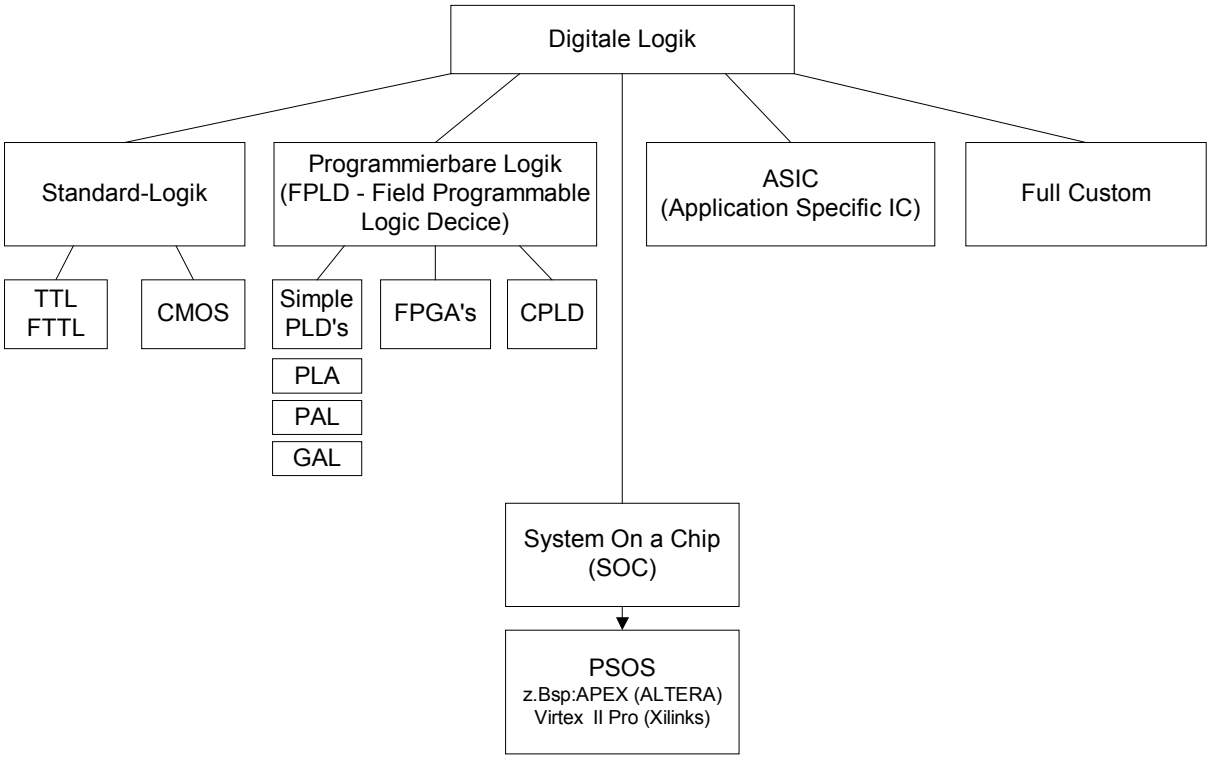
5 Entwurfsablauf – (Design-Flow)



Netlist-Extractor:	Schaffen einer einheitlichen toolinternen Darstellungsform.
Data-Base-Builder:	Schaffen einer *flachen* nicht hierarchischen und technologieunabhängigen Netz-Listen-Beschreibung (Representation) durch Auflösen / Einbinden von Submodulen aus Packages.
Fitter:	Einpassen, Abbilden der Schaltung auf den verwendeten Schaltkreis.
Assembler:	Erzeugen des Programmier-Files (JEDEC-File)

6 Digitale Logik-Technologien

6.1 Überblick



FPLD Field Programmable Logic Device. Ist ein anwendungsprogrammierbarer Schaltkreis (das heißt beim Kunden programmierbar).

PLA Programmable Logic Array
... Nachbildung einer Funktion in disjunktiver Normalform
... Produkt- und Summenterme sind programmierbar SOP...Sum of Products
... klassische PLA ist maskenprogrammiert
... Vorteil: maximale Flexibilität
... Nachteil: Aufwand!

PAL Programmable Array Logic (Phillips)
... ist eine PLA-Variante mit fest vorgegebenen S-Termen
... P-Terme frei programmierbar
... der begrenzten Funktionalität begegnet man durch verschiedene Varianten.
... klassische PAL ist „fuse programmed“
... weiterentwickelte PAL-Varianten
  ... Hinzunahme von Flip-Flops
  ... Flip-Flops verschiedenen Typs
  ... Flip-Flop-Taktung flexibel
  ... Ausgangstreiber flexibel

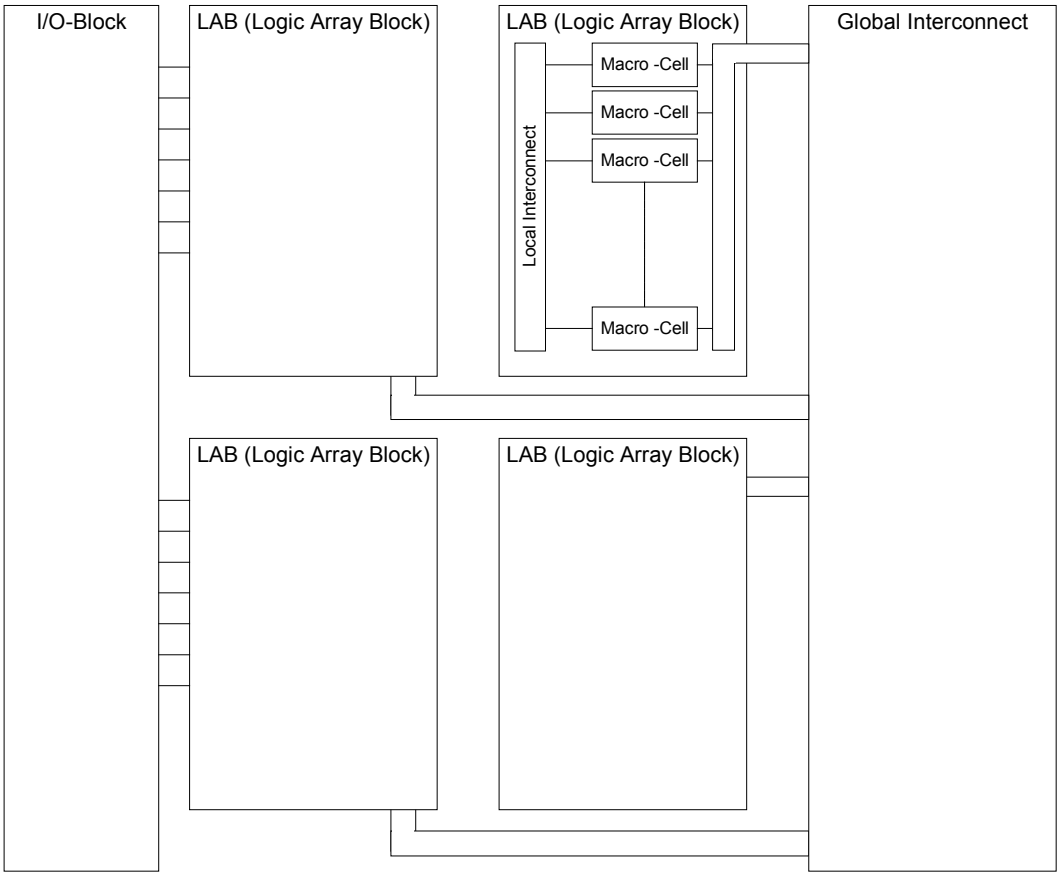
GAL Gate Array Logic (Lattice)
… eine GAL kann verschiedene PAL’s nachbilden
... sind EEPROM programmiert

SPLD ... begriffliche Zusammenfassung für GAL, PAL, PLA.

CPLD Complex Programmable Logic Device

… entsteht durch Zusammenfassen von SPLD's (Macrocells) zu komplexen Anordnungen.

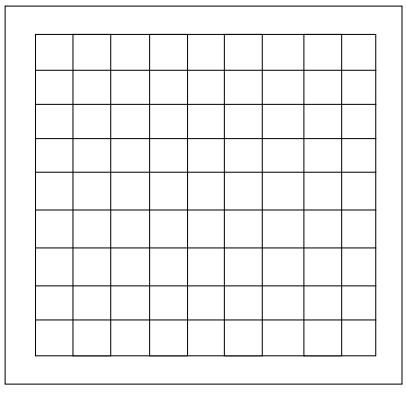
Struktur-Prinzip:



Beispiel: MAX 7000 CPLD von ALTERA
- 16 Macro-Cells zu einem LAB zusammengefasst
- EEPROM-basierte Programmierbarkeit (Konfiguration bleibt erhalten)
- In-Circuit-programmable (in der Schaltung programmierbar)
- ca. 2500 Gatter

FPGA Field Programmable Gate Array

... Weiterentwicklung der Gate-Array-Technik

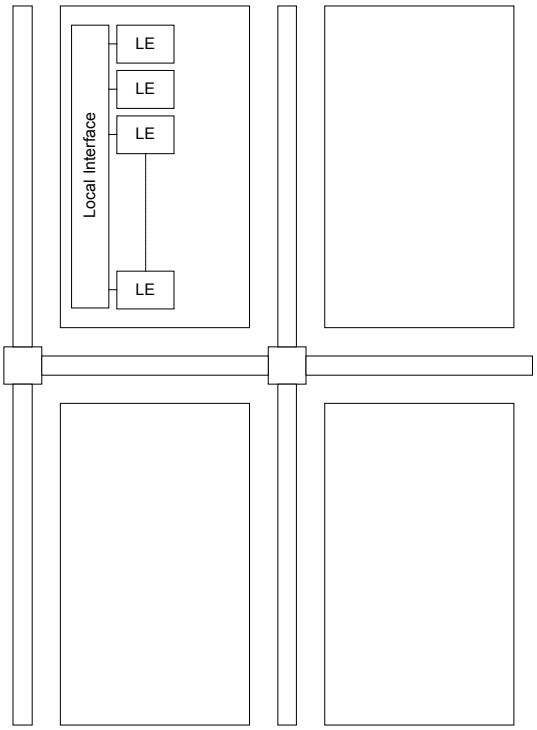


(Struktur)

... steht für Feld von Gattern
→ Weiterentwicklung zu Logic-Elements (LE)
... Logic-Elements sind über ein verteiltes Verbindungsnetz gekoppelt
... Verbindungsnetz ist programmierbar
→ über SRAM
Programmierung nach Abschalten der Versorgungsspannung verloren.

Beipiel: FLEX 10K FPGA von ALTERA

Flexible-Logic-Elements-Matrix
- zweidimensionales Feld von Blöcken (LAB) aus
8LUT (Lookup-Table) basierten Logic-Elements (LE)
- SRAM basiert.



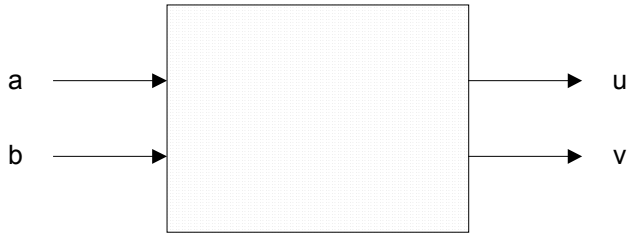
7 VHDL

- ursprünglich zur Beschreibung und Simulation von Hardware entwickelt (nicht für Synthese)
  ➔ nicht alles synthetisierbar
  ➔ verschiedene Synthesetools können unterschiedlich „viel“ bzw. „gut“ synthetisieren.
- 1980 erste Tools, die einige Teile von VHDL (RT-Level) in eine Netz-Liste übersetzen konnten.
  ➔ verwendbar für Design
- kennt 3 grundsätzliche Beschreibungsarten:
  - Datenflussbeschreibung
  - Verhaltensbeschreibung
  - Strukturbeschreibung (Struktur-Hierarchien)
- modelliert Signale mittels diskreter Zustände (0,1,Z,...)

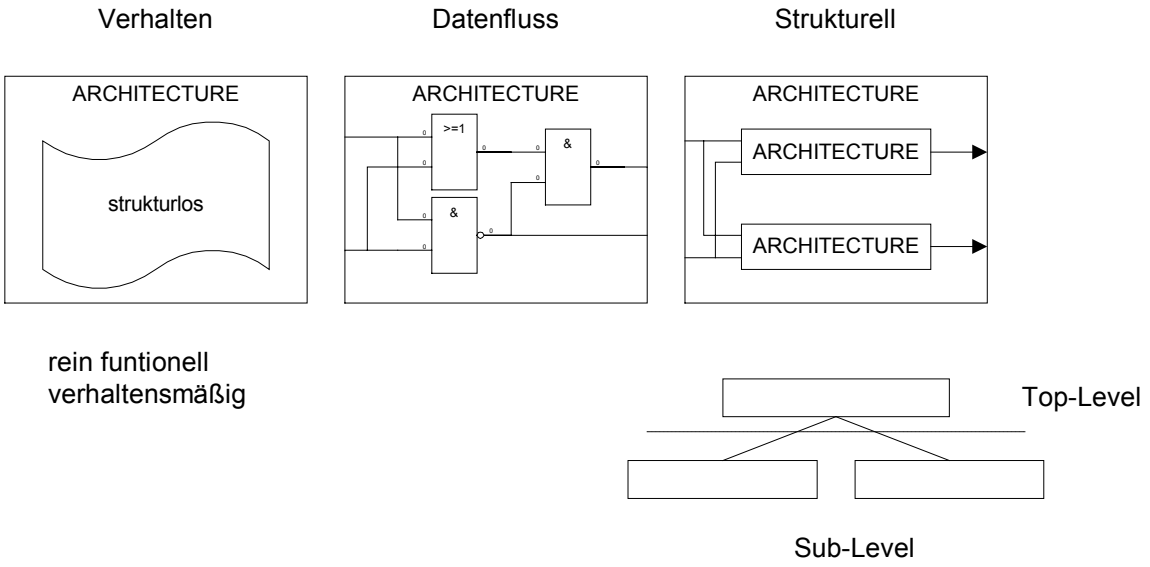
7.1 Prinzipieller Ansatz zur Systembeschreibung

- Ein System besteht aus einer oder mehreren hierarchisch geordneten Einheiten
- Eine Einheit (Subsystem) wird getrennt nach „Außensicht“ ENTITY und „Innensicht“ ARCHITECTURE beschreiben.

Außenansicht ENTITY (Interfacebeschreibung)



Innenansicht ARCHITECTURE



- strukturierte Beschreibung – bei komplexer Verknüpfung (komplexer als Kombination Grundverknüpfungen)
- parallel ablaufende (Verknüpfungs-)Prozesse werden verhaltensmäßig mit den üblichen Ausdrucksmitteln einer sequentiellen Programmiersprache beschrieben.
- unabhängige Signalflüsse.

7.2 VHDL-Beschreibung (Struppi-Beipielschaltung)

```
LIBRARY IEEE;                          {Name der verwendeten vordefinierten
                                        Designbibliothek}
USE IEEE.STD_LOGIC_1164.ALL;           {Bibliothek enthält Package
                                        "STD_LOGIC_1164" welches einen Satz
                                        vordefinierter Funktionen und Typen enthält.
                                        Insbesondere definiert es einen 9-wertigen
                                        Signaltyp. Falls keine Angabe wird das
                                        STANDARD Package verwendet}
ENTITY struppi IS                      {Interfacedefinition für Einheit struppi}
   PORT (a,b:IN bit; u,v:OUT bit);
END;


ARCHITECTURE struppi_df1 OF struppi IS
BEGIN
   u <= a XOR b;                       {Beschreibung des "Innenlebens"}
   v <= a NAND b;                      {Anweisungen sind als parallel zu betrachten}
END;
```

7.3 VHDL – Syntaktisches Grundgerüst

IEEE-Standard – VHDL Language Reference Manual

design-unit →

{context-item}

library-unit

library-unit →

entity-declaration | architecture-body | configuration-declaration

…

context-item →

library-clause | use-clause

`LIBRARY` id_list;

`USE` selected_name …;

entity-declaration →

`ENTITY` id `IS`

[...]

`PORT` interface_list;

[…]

`END` id;

architecture-body →

`ARCHITECTURE` id `OF` entity_name `IS`

{declaration-items}

`BEGIN`

concurrent_statements;

`END` id;

concurrent_statement →

concurrent_signal_assignments | component_instantion_assignments | process_statement

concurrent_signal_assignment →

simple_signal_assignment

conditional_signal_assignment

selected_signal_assignment

VHDL-Operatoren

einstellig	`NOT, ABS, +, -`

zweistellig	`AND, NAND, OR, NOR, XOR, XNOR`
	`MOD, REM`
	`+, -, *, /, **`
	`&,=,<,<=,>=,/=`

Signaltypen

modellieren ein Signal unterschiedlich genau (diskret)

{0,1}

{0,1,X} X ... unbestimmt

{0,1,X,Z} Z ... Tristate/Hochohmig

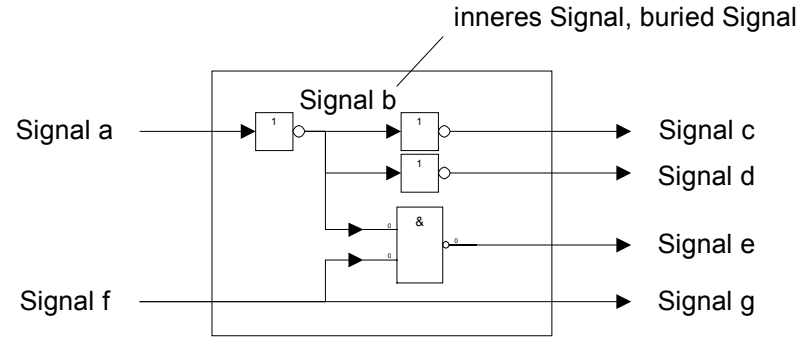
Signal-Zuweisung (Signal-assignment)

Ein Signal ist jede Änderung einer physikalischen Größe, die einem eine Information gibt.

Elektrische Signale breiten sich über elektrische Leitungen aus.
(Lichtgeschwindigkeit 30cm/ns)

Elektrische Daten(-signale) fließen zwischen den Logikeinheiten von Eingängen zu den Ausgängen.

➔ Datenfluss (DataFlow)



„Verknotete“ elektrische Drähte führen ein- und dasselbe Signal

➔ Knoten (Node)

Signal_assignments weisen Signalen Werte zu. Die Werte können sich durch logische Verknüpfungen ergeben.

```
b <= NOT a;
c <= NOT b;
d <= NOT b;
e <= f NAND b
g <= f;
```

} parallele Zuweisungen

Concurrent – Modellierung

Beispiel: R-S-Latch

```
ENTITY rslatch IS
  PORT (s,r:IN bit; q,nq:BUFFER bit);
END;

ARCHITECTURE rslactch_df OF rslatch IS
BEGIN
  q <= s NAND nq;
  nq <= r NAND q;
END;
```