Einführung in die Technische Informatik
Skript

In der Hoffnung, dass es was nützt ...

Christian Kroh

29. Juni 2014, Dresden

Inhaltsverzeichnis

1 Allgemeine Einführung

1.1 Qualifikationsziele

Die Studierenden kennen Systemarchitekturen und Modellierungsparadigmen von VLSI-Systemen.
Sie sind in der Lage Beschreibungen von Hardware-Systemen durch Simulation zu verifizieren und mithilfe typischer Werkzeuge in reale Schaltungen umzuwandeln.
Sie können den Ressourcenbedarf, das Zeitverhalten und die Verlustleistung abschätzen oder evaluieren und daraus Entwurfsentscheidungen ableiten.

1.2 Literatur

- F. Kesel und R. Bartholomä: Entwurf von digitalen Schaltungen und Systemen mit HDLs und FPGAs, Oldenbourg Wissenschaftsverlag, ISBN 978-3-486-58976-4.
- H.-D. Wuttke und K. Henke: Schaltsysteme – Eine automatenorientierte Einfürung, Pearson Studium, ISBN 3-8273-7035-3.
- H. M. Lipp und J. Becker: Grundlagen der Digitaltechnik, Oldenbourg Wissenschaftsverlag, ISBN 978-3-486-59747-9.

Teil I

VLSI-Systementwurf

1 Einführung

1.1 Themenschwerpunkte

Verarbeitungsleistung

Im Vordergrund stehen:

- Schnelle Verarbeitung auch einzelner Bits
- Parallelität auf:
  - Bitebene
  - Befehlseben
  - Threadebene
  - Prozess- und Anwendungsebene
- dynamische Rekonfiguration

⇒ Erfüllung der gegebenen Anforderungen

Systemintegration

Im Vordergrund stehen:

- Mehrprozessorsyteme, Mehrkern-, Vielkernprozessorsysteme
- Mehr-Chip- / Einzel-Chip-Lösungen (System-on-a-Chip)
- Parallele Entwicklung von HW und SW (HW-/SW-Codesign)
- System-Prototyping (FPGA-Entwurf)

⇒ Kosteneinsparung, Entwicklungszeiteinsparung (Time-to-Market)

Verlustleistung

Im Vordergrund stehen:

- Verlustleistung im Standby (Akkubetrieb)
- Maximales Abwärmebudget

Kenngrößen:

- Statische und dynamische Verlustleistung
- MIPS pro Watt

⇒ Sowohl für eingebettete Systeme als auch für Server

Korrektheit

Aspekte:

- Verifikation eines Schaltkreises, simulativ / formal
- Profiling und Debugging unter Echtzeitbedingungen (Trace)

⇒ Fehlerfreier Erstentwurf

Fehlertoleranz

Toleranz gegenüber:

- Permanenten Fehlern (zeitunabhängig nach erstem Auftreten)
- Intermitierenden Fehlern (nur unter bestimmten Betriebsbedingungen)
- Transienten Fehlern (aufgrund statischer Störungen)

Fehlererkennung und -korrektur:

- Autonom durch Hardwarearchitektur
- Aus Kombination von HW und SW

⇒ Insbesondere wichtig für Sicherheitskritische, hochverfügbare und langlebige zuverlässige Systeme
⇒ Steigende Siginifikanz mit abnehmenden Strukturgrößen (Integrationsgrad) sowie steigender Transistoranzahl (Moore's Law)

1.2 Inhalte der Lehrveranstaltung

1.2.1 Inhalte der Vorlesung

1. Klassifikation von Schaltkreisen
2. Grundlagen des Schaltkreisentwurfs
3. Automatendarstellung, -kopplung, -vereinfachung
4. Hardwarebeschreibungssprachen
5. Programmierabare Schaltkreise, insbesondere FPGAs - Teil 1
6. Programmierabare Schaltkreise, insbesondere FPGAs - Teil 2
7. Modellierung und Simulation
8. Zeitverhalten und Test
9. Hochgeschwindigkeit und Verlsustleistung
10. Anwendungsbeispiele

1.2.2 Inhalte des Praktikums

1. Altera Quartus-Toolchain & Praktikumsboard DE0 mit Cyclone-3
2. Schaltnetze und Schaltwerke
3. Modularisierung
4. "Komplexe"Anwendung: Stoppuhr

1.3 Klassifikation von ICs

1.3.1 Zwei Sichten

Klassifizierung von integrierten Schaltkreisen (ICs) in

- Standarschaltkreis (Standard-IC) und
- applikationspezifische Schaltkreise (Application-Specific IC, **ASIC**)

unter zwei Gesichtspunkten möglich:

- **Herstellungssicht**
- **Entwurfssicht**

Herstellungssicht:

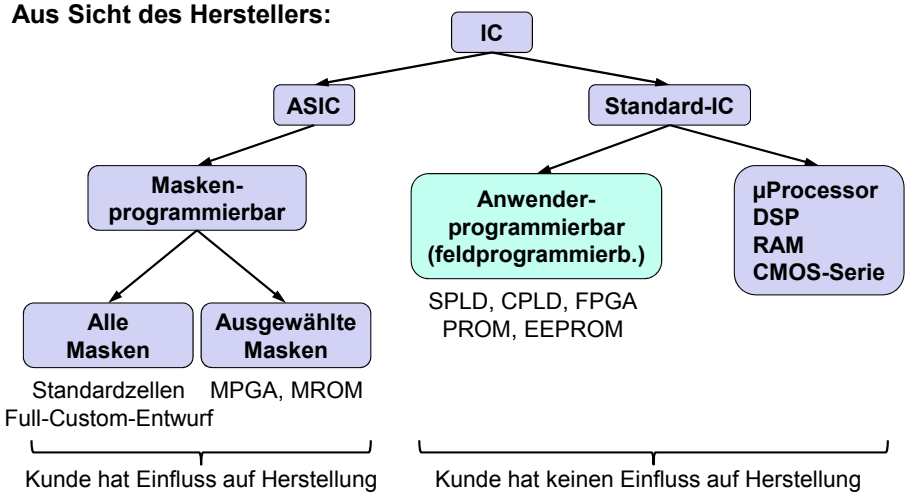
- Standard-IC = große Stückzahl für viele Kunden
- ASIC = für eine/n Kunden/Applikation speziell entwickelter und gefertigter IC mit zugeschnittener Funktionalität

Entwurfssicht:

- Standard-IC = (Hardware-)Funktionalität kann nicht vom Anwender beeinflusst werden
- ASIC = vom Anwender selbst entwickelte (Hardware-)Funktionalität

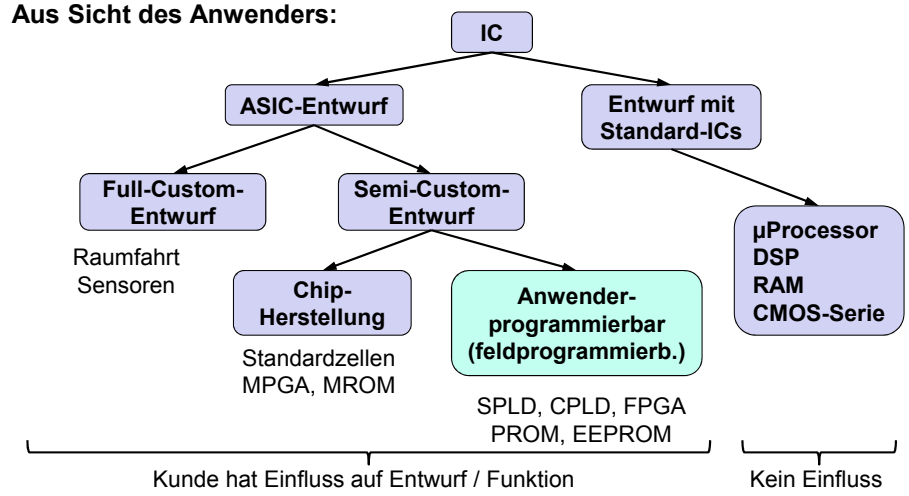
Klassifikation nach Herstellungssicht

Aus Sicht des Herstellers:



Klassifikation nach Entwurfssicht

Aus Sicht des Anwenders:



Abgrenzung der Entwurfsalternativen

Entwurfs-alternative	Transistor-layout	Gatter-position	Verdrah-tung	Funktion
Full-Custom	+	+	+	+
Standardzellen	(+)	+	+	+
MPGA, MROM	-	(+)	+	+
PLD, FPGA, PROM	-	-	(+)*	+*
Einzel-ICs	-	-	-	+

* = Einfluss nur indirekt durch Programmierung

1.3.2 Anwenderprogrammierbare IC

Merkmale:

- Field-Programmable ⇔ feldprogrammierbar
- Vor Ort (im Feld) vom Anwender programmierbar
- Hardware ist fix. Funktionalität kann aber mittels spezieller Konfiguration "programmiert"werden

Anwendung:

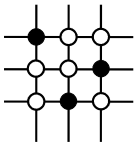
- Anwendungsspezifische IC bei kleinen und mittleren Stückzahlen
- Mehrfach neu programmierbar zwecks Optimierung und Fehlerbehebung, auch während des praktischen Einsatzes
- Einfache Integration eines ganzen Systems auf einem Chip
- Prototyping, HW-/SW-Codesign

1.3.3 Hardwareprogrammierung

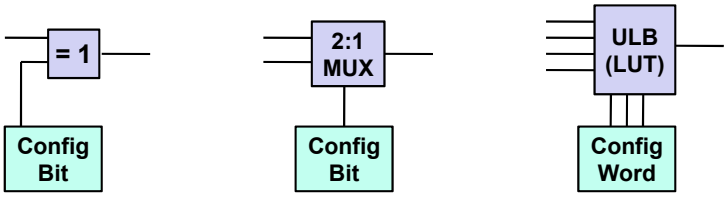
Hardwareprogrammierung

Programmiert (oder auch konfiguriert) werden können:

- Verdrahtung / Verbindung



- Funktionen



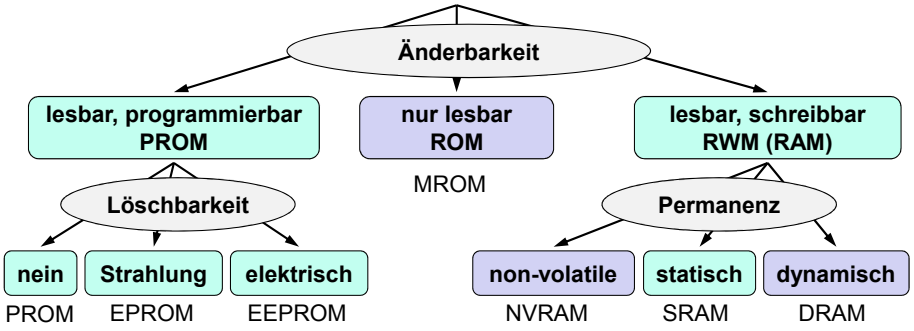
- Speicher: PROM, EPROM, EEPROM (Flash)

1.3.4 Programmiertechnologien

Programmiertechnologien (1)

Klassifikation hinsichtlich Änderbarkeit:

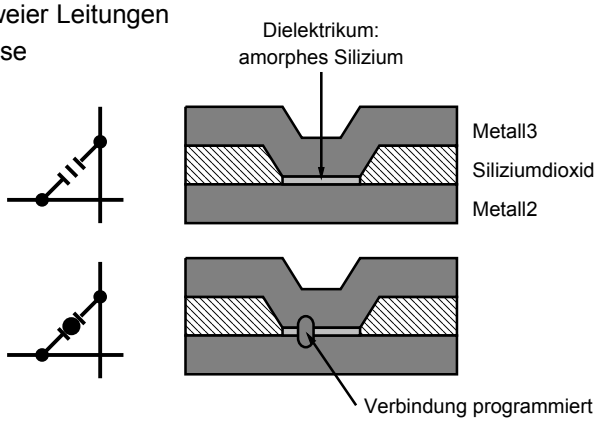
analog Halbleiterspeicher



Programmiertechnologien (2)

Antifuse:

- Programmierung elektrisch, aber nur einmalig
- Schalter oder Verbindung zweier Leitungen
- Beispiel: Metall-Metall-Antifuse

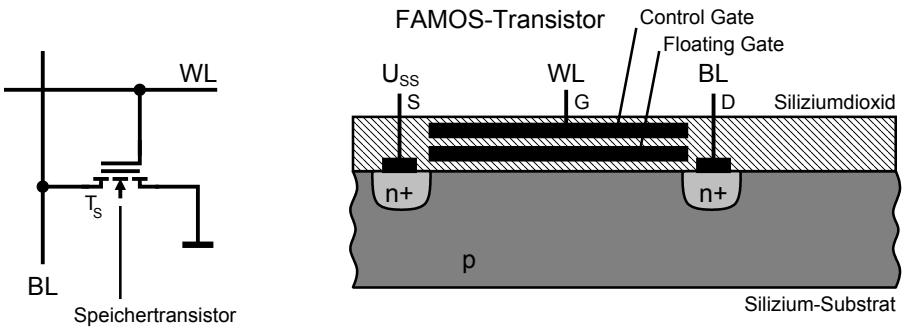


el u. Bartholomä: Entwurf von digitalen Schaltungen
Systemen mit HDLs und FPGAs

Programmiertechnologien (3)

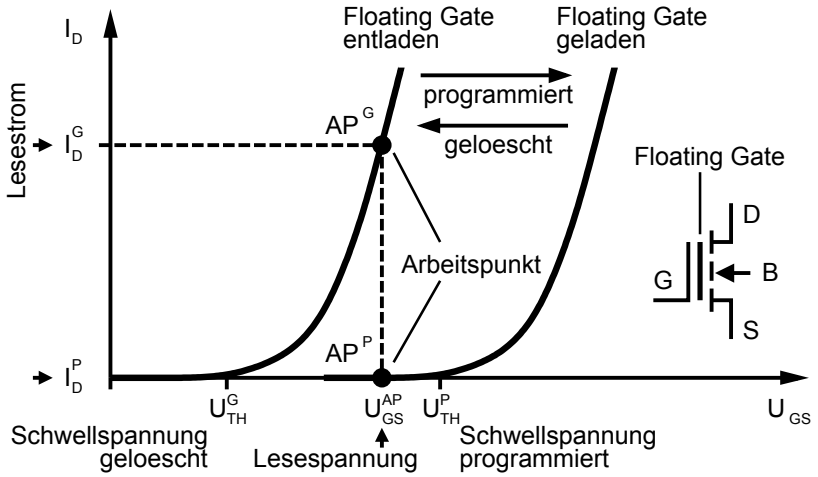
EPROM FAMOS-Transistor: (Floating-Gate Avalanche-injection MOS)

- Ladungsspeicherung (Elektronen) auf dem Floating-Gate.
- Programmierung elektrisch, Löschen durch UV-Bestrahlung (mehrmalig).



Schnittdarstellung planare EPROM-Speicherzelle

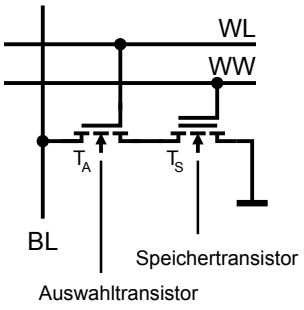
Kennlinie des FAMOS-Transistors:

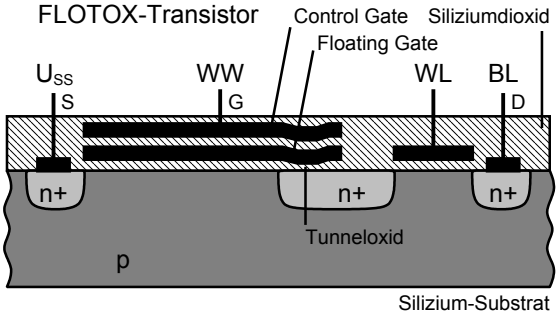


Programmiertechnologien (4)

EEPROM mit FLOTOX-Transistor: (Floating-Gate Tunneling Oxide)

- Ladungsspeicherung (Elektronen) auf dem Floating-Gate.
- Programmierung elektrisch, Löschen elektrisch über WW (Word Write).
- In-System Programming





Schnittdarstellung planare EEPROM-Speicherzelle

Programmiertechnologien (5)

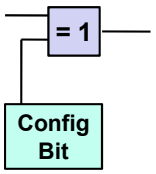
Flash-EEPROM:

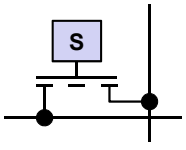
- EEPROM mit 1-Transistor-Speicherzelle (FLOTOX-Transistor)
- Nur blockweises Löschen
- NAND-Flash für hohe Speicherdichten
- NOR-Flash für Speicher mit geringen Zugrifsszeiten
- Problem: Haltbarkeit, z.B. Intel ETOX-Zelle:
  - Endurance von 10^5-10^6 Lösch-/Programmierzyklen.
  - Data Retention von ca. 10 Jahren.

Programmiertechnologien (6)

SRAM:

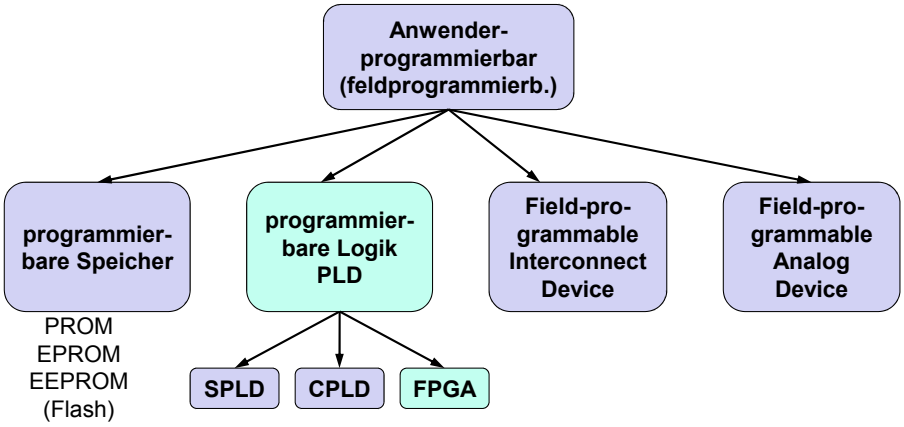
- 4 MOSFET pro Speicherzelle
- 1 Konfigurationsbit pro Speicherzelle zur
  - Funktionsauswahl
  - Ansteuerung eines Pass-Transistors
- Beliebig oft programmierbar, aber Verlust der Information bei Ausfall der Betriebsspannung
- Im Betrieb einfach programmierbar
  - Schreib-/Lesespeicher
  - Dynamische Rekonfiguration





1.3.5 Klassifikation Anwenderprogrammierbare IC

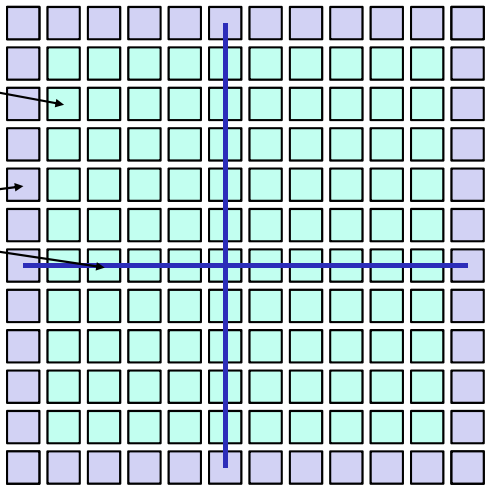
Klassifikation Anwenderprogrammierbare IC



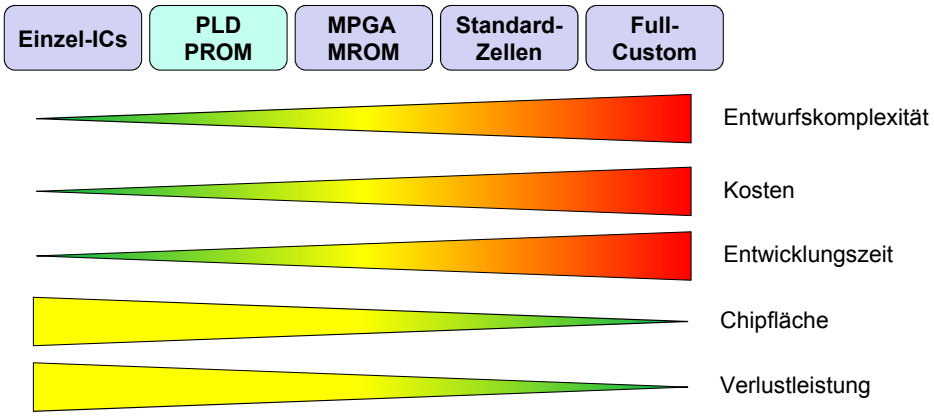
Beispiel: FPGA-Architektur

Grundlegende Bestandteile:

- Funktionsblöcke (FB):
  - angeordnet als Matrix,
  - Multiplexer- oder LUT-basiert.
- I/O-Zellen als spezielle FB.
- Allgemeine lokale Verdrahtung, sowie globale und dedizierte Signalleitungen.
- Spezielle Hard-Makros.



Gegenüberstellung der Entwurfsalternativen



PLD = Programmierbare Logik (SPLD, CPLD, FPGA)
PROM = Programmierbare Speicher (PROM, EPROM, EEPROM)

2 Schaltkreisentwurf

2.1 Abstraktionsebenen und Sichten

Ebenen des Entwurfs:

- Charakterisierung des jeweiligen Detailiertheitsgrades der Beschreibung des Entwurfsgrades
- Abstraktionsgrad von der eigentlichen physikalischen Realisierung
- Abstraktionsniveaus, Hierarchien

Sichten des Entwurfs:

- Betrachtung des Entwurfsgegenstandes aus verschiedenen Richtungen
- Sicht = Eigenschaften die den Entwurfsgegenstand Charakterisieren
- Alle Sichten auf jeder Entwurfseben ⇒ Y-Diagramm / x-Diagramm

2.1.1 Abstraktionsebenen

Systemebene: Systemkonzept des Entwurfsgegenstandes

Algorithmische Ebene: Algorithmische Beschreibung des Entwurfsgegenstandes

Register-Transfer Ebene: Datentransfer und -verarbeitung zwischen Registern

Logikebene: Beschreibung auf Gatterniveau

Schaltkreiseben: Transistorebene im weiteren Sinne, umfasst:

- Schalterebene
- Schaltungsebene
- Bauelementebene
- Technologieebene

2.1.2 Sichten

Verhaltenssicht: Beschreibung des zeitlichen Verhaltens durch charaktersierende Variablen und deren Werteverläufe über die Zeit

$$\vec{y}(t) = f(\vec{x}(t))$$

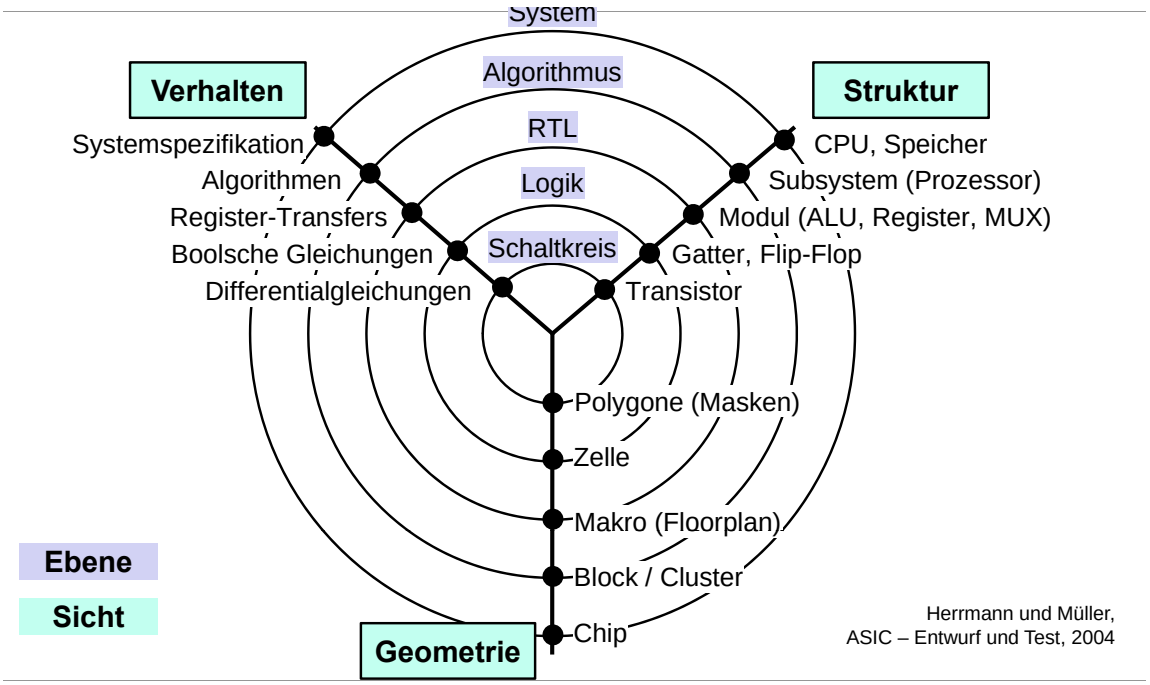
Struktursicht: Spezifizierung eines Objektes durch Subobjekte und deren Verbindungsstrukturen

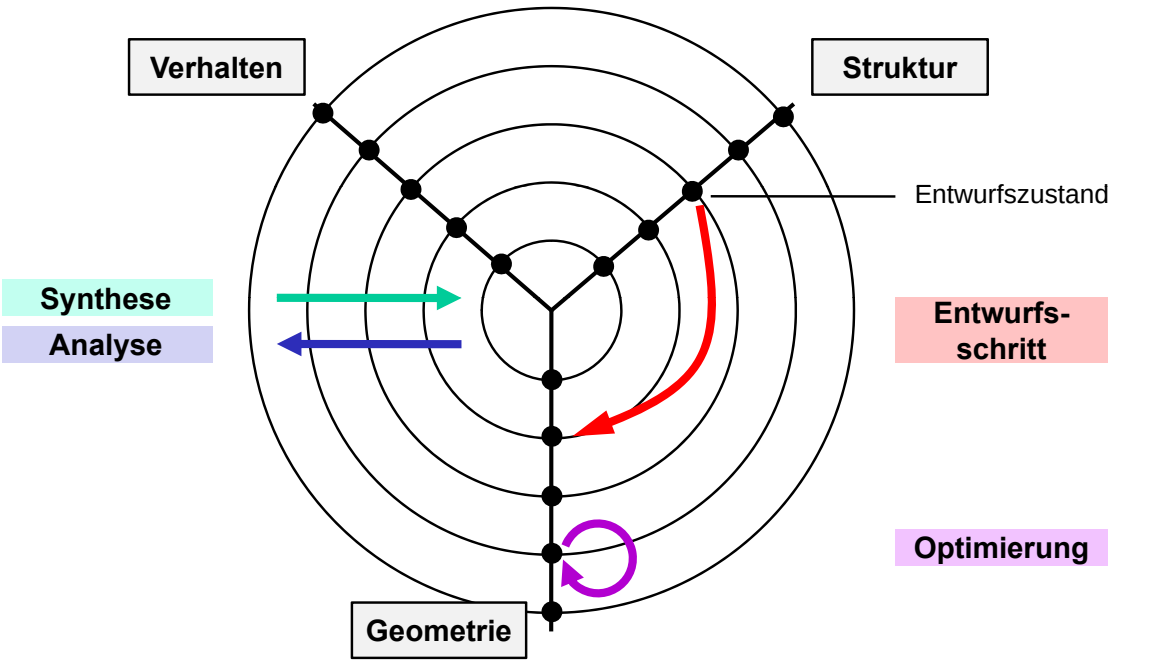
Geometriesicht: Räumliche Ausdehnung der Subobjekte

Testsicht: Existenz oder Nichtexistenz angenommener struktureller oder funktionaler Defekte
(F. J. Rammig, Systematischer Entwurf digitaler Systeme, B.G Teubner Stuttgart 1989)

2.1.3 Y-Diagramm nach Gajski

- Stellt Ebenen und Sichten in FOrm eines Y-Diagrammes dar
- Ursprünglich nur 3 Ebenen
- Heute: Erweiterung auf 5 Ebenen:
  - Systemebene
  - Algorithmische Ebene
  - Register-Transfer Ebene
  - Logikeben
  - Schaltkreisebene





2.2 Entwurfsablauf

Allgemein: Transformation einer Aufgabenstellung (Pflichtenheft) in einen fertigen Schaltkreis

Top-Down-Strategie:

- Systemebene → Schaltkreisebene
- Vorteil: Parallele Entwicklung auf unteren Ebenen
- Nachteil: Systemspezifikation zu Projektbeginn oft zu ungenau

Bottom-Up-Strategie:

- Analyse vorhandener Komponenten
- Zusammensetzen von neuen Komponenten auf höherer Ebene im Sinne der Aufgabenstellung
- Nachteil: globales Ziel wird nicht immer erreicht

⇒ **Meet-in-the-Middle**

Entwurfsschritt:

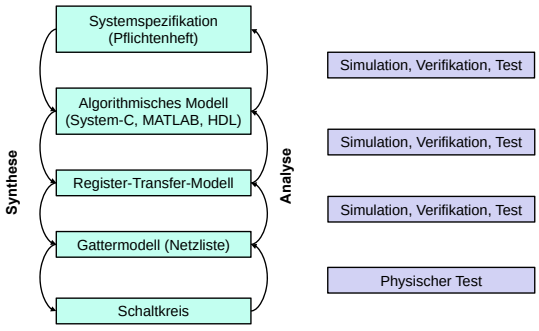
- generierende Aktivität
- überprüfende Aktivität

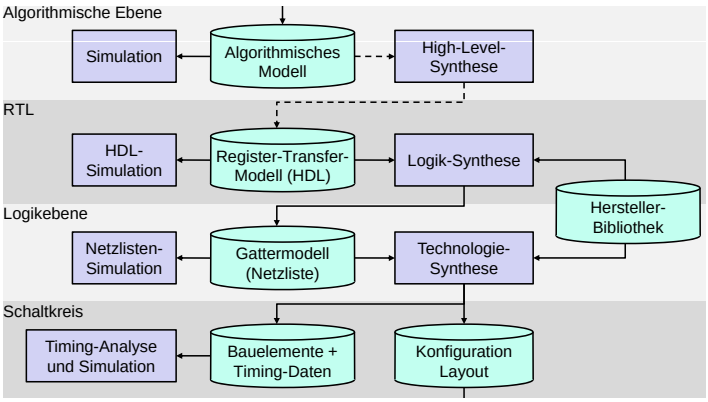
Syntheseschritt:

- Abbildung eines Entwurfsschrittes in Richtung auf das Entwurfsziel
- Abstraktionsgrad sinkt, Detailhiertheitsgrad steigt
- Einbringung neuer Informationen

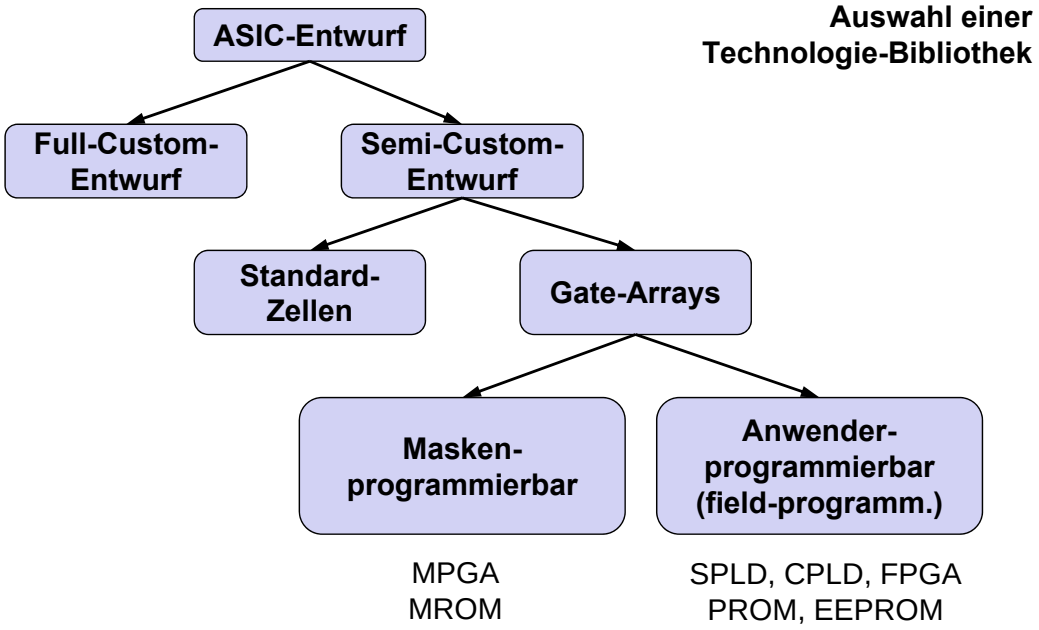
Analyseschritt:

- Abbildung eines Entwurfsschrittes in umgekehrter Richtung zum Syntheseschritt
- Gewinnung abstrakter Informationen durch Zusammenfassen und Generalisieren von Details (Extraktionsprozess)
- Beispiel: Validierung eines Syntheseschrittes

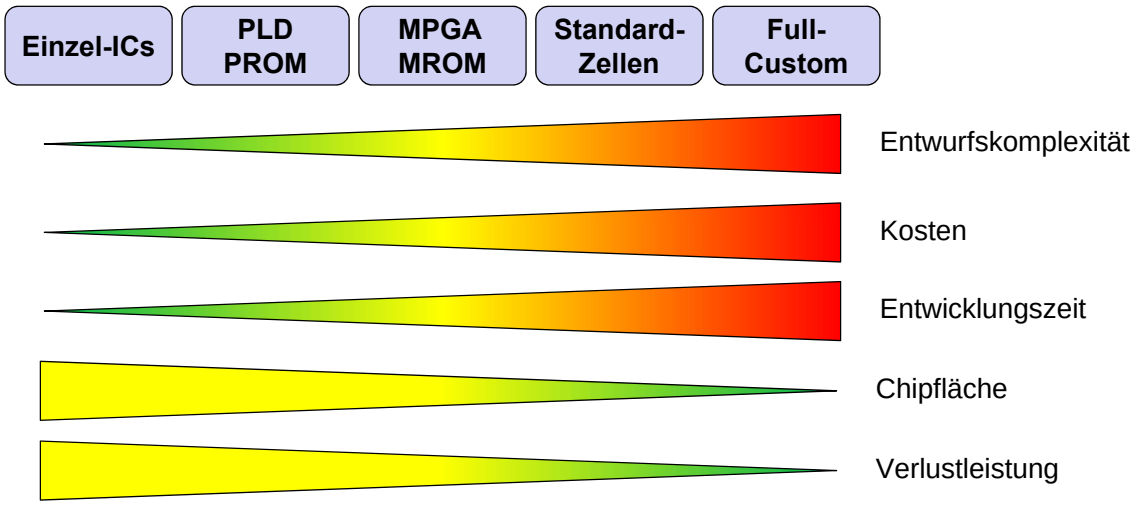




2.3 Entwurfsstile

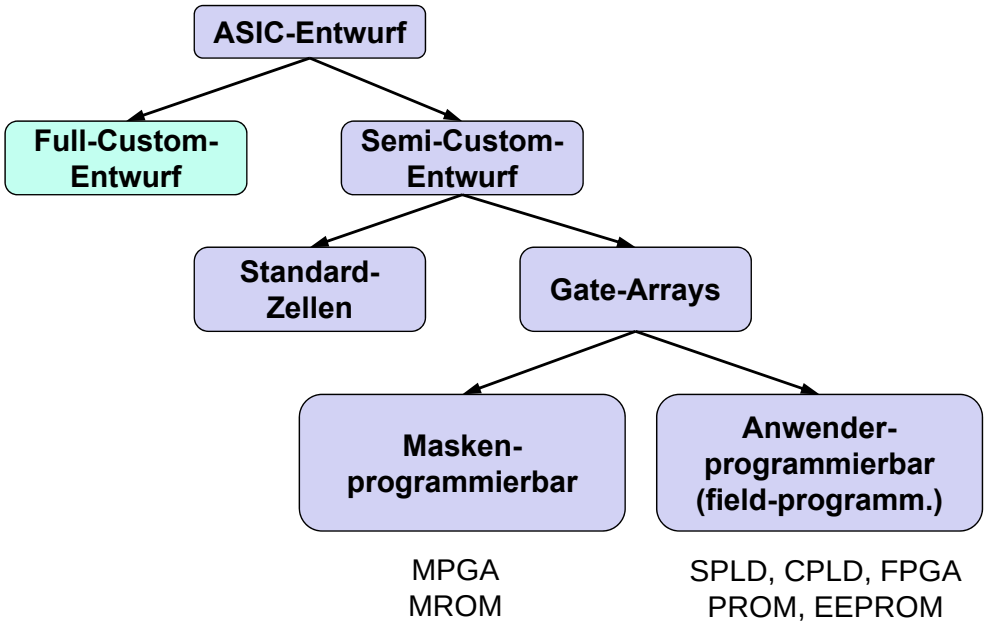


Gegenüberstellung Entwurfsalternativen



PLD = Programmierbare Logik (SPLD, CPLD, FPGA)
PROM = Programmierbare Speicher (PROM, EPROM, EEPROM)

2.3.1 Full-Custom Entwurf



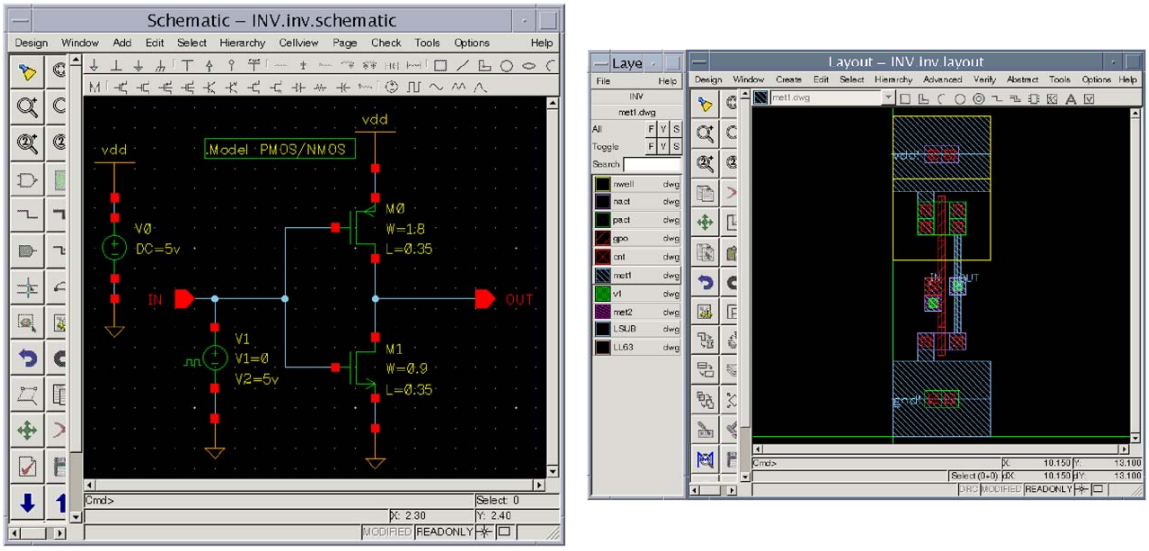
Merkmale:

- Platzierung und Verdrahtung selbst entworfener Transistoren & Gatter
- Auch Mischung von Analog- und Digitaltechnik, z.B. für spezielle I/O-Signaltreiber
- Erfüllung spezieller Anforderungen, z.B. gehärtet gegen Strahlung
- Häufig nur auf kleine Teilschaltungen angewendet
- Hoch qualifizierte Entwicklungsingenieure mit Detailkenntnissen zu den Prozessen erforderlich

Anwendung:

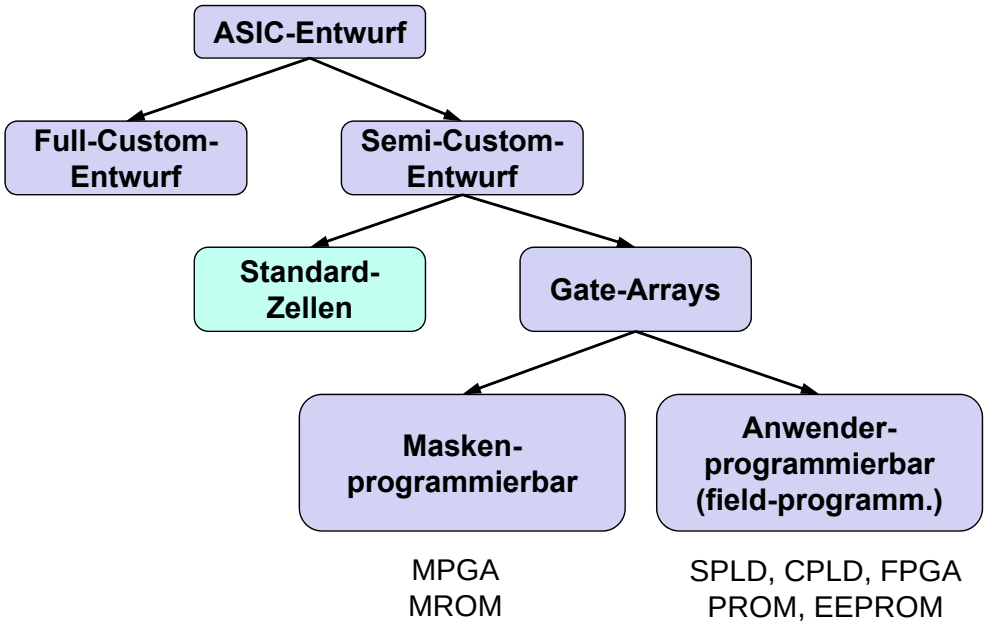
- Sensorik, Mixed-Signal-Schaltungen
- Raumfahrt

Beispiel: Inverter



CEC Huada Electronic Design: ZENI --- Full Custom IC Design Flow Workshop, http://www.zeni-eda.com

2.3.2 Standardzellenentwurf



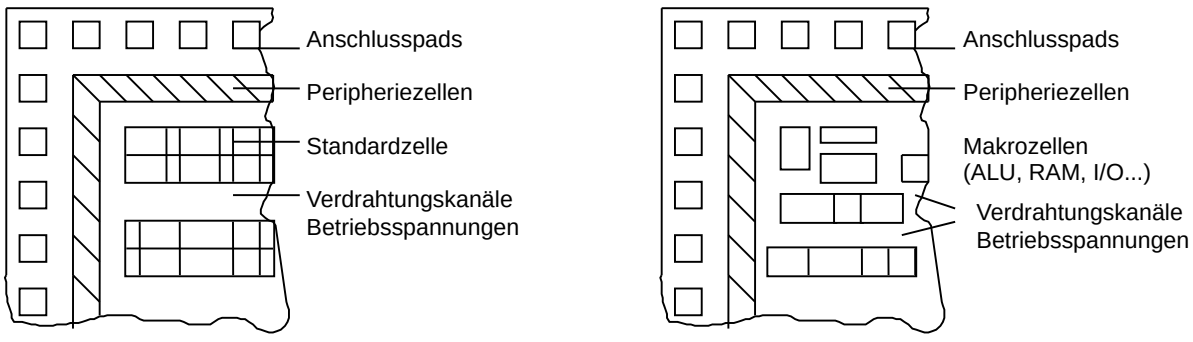
Merkmale:

- Platzierung und Verdrahtung vorgegebener Gatter- oder Makrozellen
- Auswahl einer Technologiebibliothek nach:
  - Fertigungstechnologie, Strukturbreite und Funktion
  - High-Speed, Low-Leakage oder Mischung aus Beidem
- Werkzeuggestützte Platzierung und Verdrahtung
- Makrozellen:
  - Vordefiniert oder per Generator kundenspezifisch erzeugt
  - Bsp: RAM, ALU, I/O-Komponenten

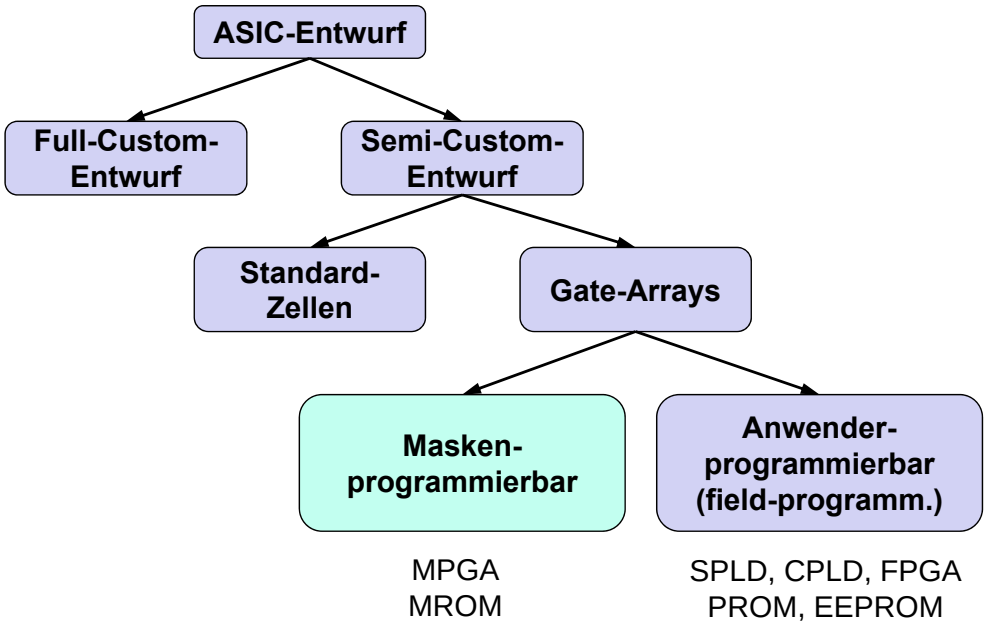
Anwendung:

- Anwendungsspezifische Schaltkreise mit hohen Stückzahlen
- Starke Optimierung bzgl. Chipfläche, Geschwindigkeit und Verlustleistung

Konventionelle Architektur vs. Strukturierte Architektur



2.3.3 Maskenprogrammierbare Gate Arrays



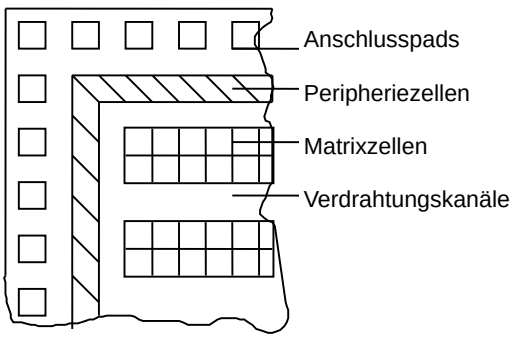
Merkmale:

- Festlegung der Funktion mittels Masken bei der Halbleiterfertigung
- Ausgewählte Masken: teilweise vorgefertigte IC (Master)
- Beispiele:
  - MPGA (Maskprogrammable Gate-Array): auch MGA
    - Vorgefertigte universelle Gatter/Makros mit fester Anordnung
    - Verdrahtung kundenspezifisch
  - MROM: ROM dessen Inhalt vom Kunden mit Masken festgelegt wird

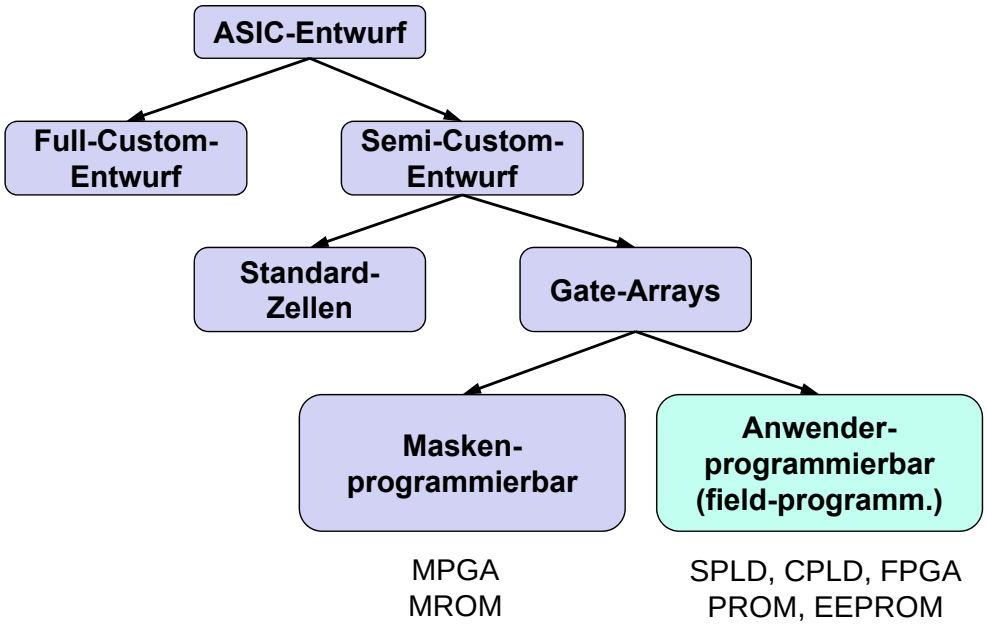
Anwendung:

- Anwendungsspezifische Schaltkreise bei mittleren Stückzahlen
- Kostenoptimiert mit reduzierten Optimierungspotential

MPGA



2.3.4 Anwenderprogrammierbare IC



Merkmale:

- Field-Programmable ⇔ feldprogrammierbar
- Vor Ort (im Feld) vom Anwender programmierbar
- Hardware ist fix. Funktionalität kann aber mittels spezieller Konfiguration "programmiert"werden

Anwendung:

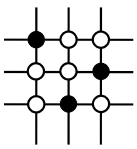
- Anwendungsspezifische IC bei kleinen und mittleren Stückzahlen
- Mehrfach neu programmierbar zwecks Optimierung und Fehlerbehebung, auch während des praktischen Einsatzes
- Einfache Integration eines ganzen Systems auf einem Chip
- Prototyping, HW-/SW-Codesign

Hardwareprogrammierung

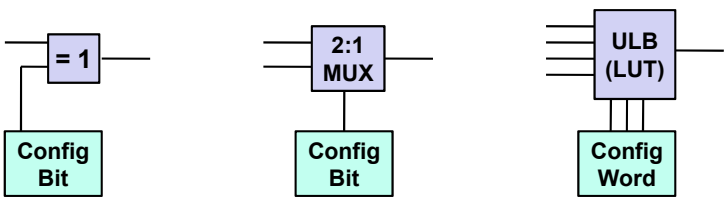
Hardwareprogrammierung

Programmiert (oder auch konfiguriert) werden können:

- Verdrahtung / Verbindung

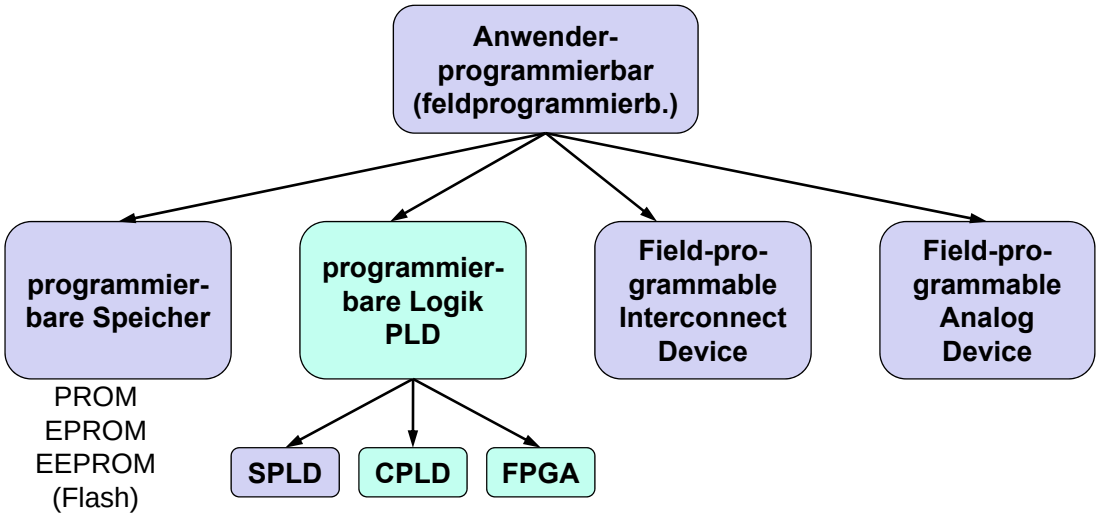


- Funktionen



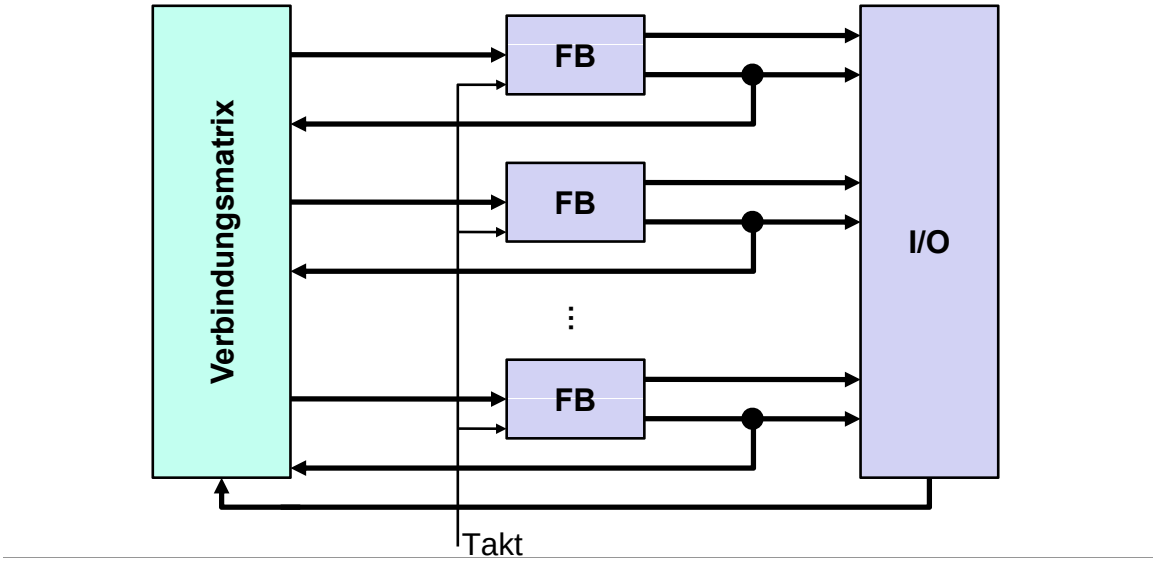
- Speicher: PROM, EPROM, EEPROM (Flash)

Klassifikation Anwenderprogrammierbare IC



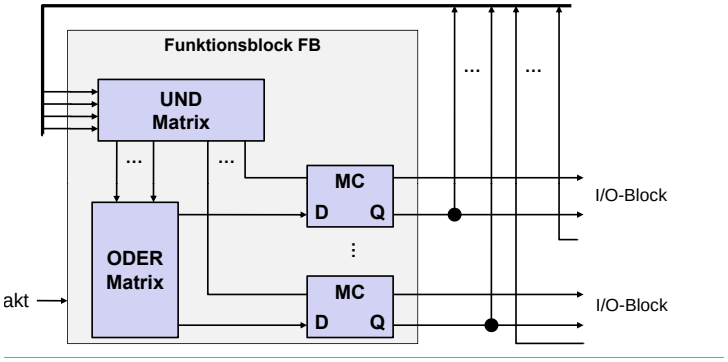
CPLD

Globale Vernetzung einer kleiner Anzahl von Funktionsblöcken (FB)

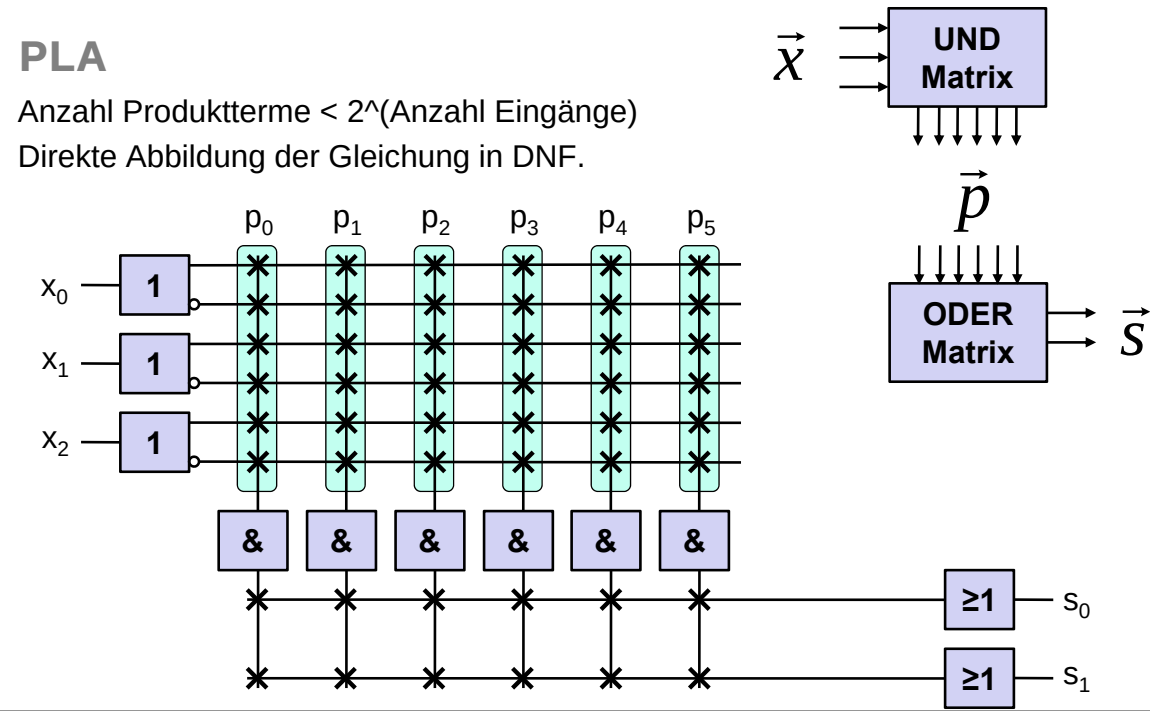


CPLD-Funktionsblock

Funktionsblock bestehend aus PLA (Und/Oder-Matrix) und Makrozellen (MC)

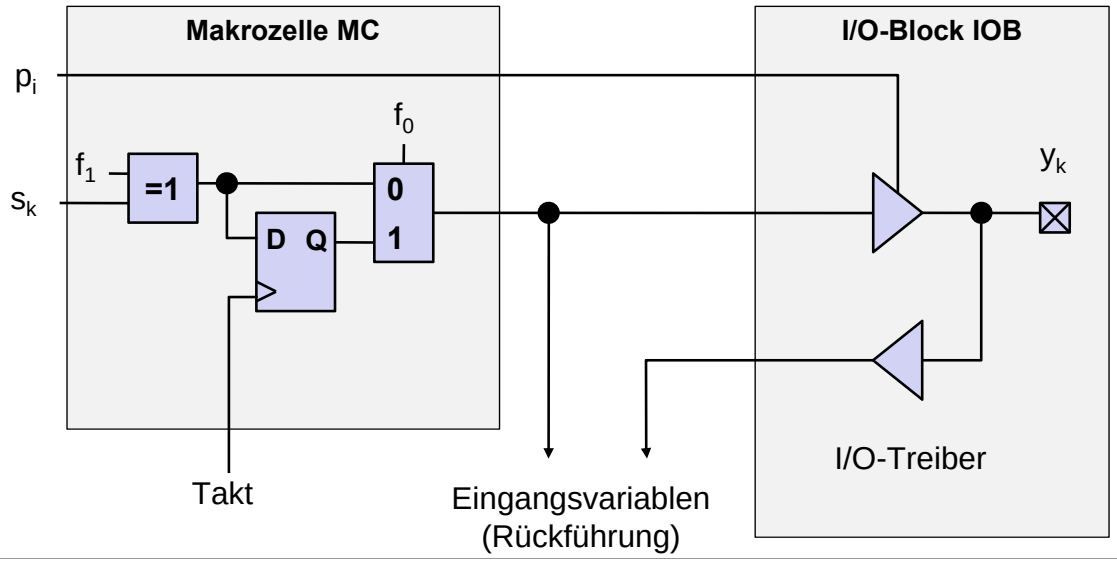


PLA



Makrozellen + I/O-Block

Verschiedene Betriebsspannungen für digitale Lokig (Core) und I/O-Pads



Konfiguration der Makrozelle:

f0	Summenterm s_k
0	nicht negiert
1	negiert

f1	Ausgang y_k
0	kombinatorisch
1	Register

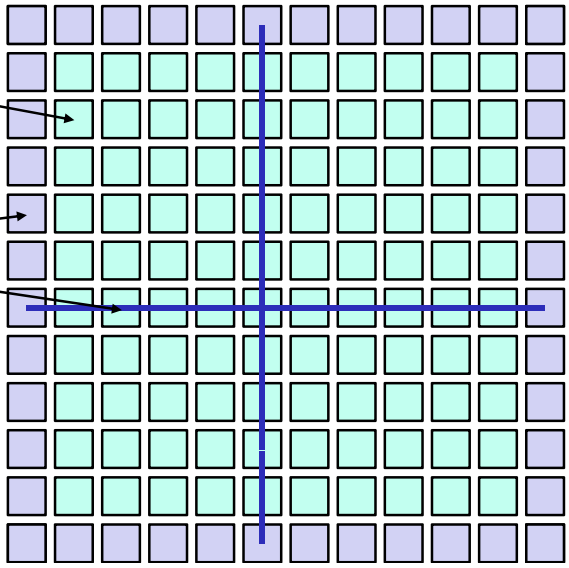
Steuerung des I/O-Blocks:
Umschaltung des I/O-Pins zwischen Ein- und Ausgang (Tri-State) zur Laufzeit mittels seperatem Produktterm möglich.

FPGA-Architektur

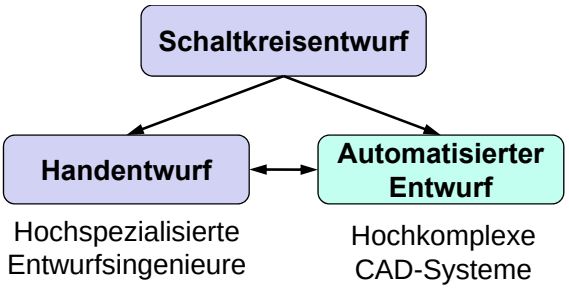
Grundlegende Bestandteile:

- Funktionsblöcke (FB):
  - angeordnet als Matrix,
  - Multiplexer- oder LUT-basiert.
- I/O-Zellen als spezielle FB.
- Allgemeine lokale Verdrahtung, sowie globale und dedizierte Signalleitungen.
- Spezielle Hard-Makros.

Details in einer späteren VL



2.4 Entwurfswerkzeuge



Auswahl CAD-Werkzeuge:

- Cadence
- Xilinx ISE
- Altera Quartus
- Synopsys

3 Automaten

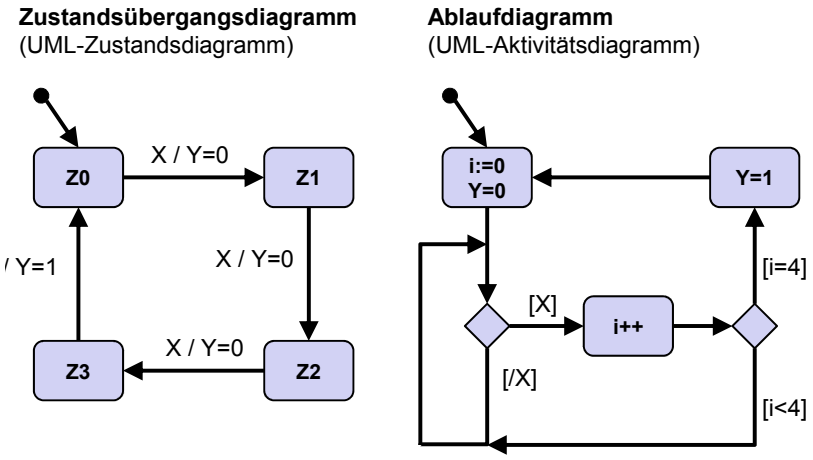
3.1 Automatendarstellung

3.1.1 Betrachtungsweisen

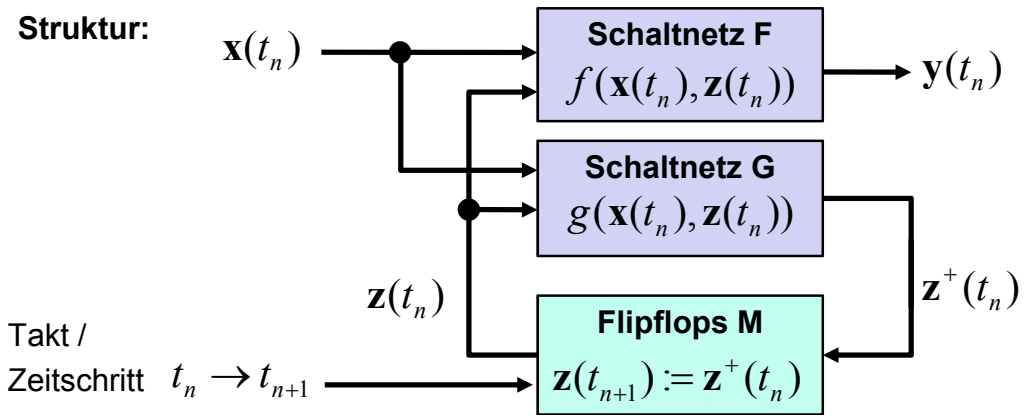
Verschiedene Sichten/Semantik:

- Zustandsübergangsdiagramm (state diagram)
  - Vernetzung von Zuständen
  - Beispiele: UML-Zustandsdiagramm, Automatengrpahen, SM Charts, GRAFCET, Sequential Function Charts (SFC)
- Ablaufdiagramm (flow chart)
  - Vernetzung von Prozessen
  - Zustand ergibt sich aus der Verkettung aller Variablenzustände
  - Beispiele: UML-Aktivitätsdiagramm, Programmablaufplan (PAP)

Binärzähler mod 4 mit Trigger X und Übertrag Y



3.1.2 Automatengraphen



Endlicher Automat: Menge der möglichen Eingabezeichen, Ausgabezeichen und inneren Zustände ist endlich ...

Synchron getakteter Automat: Zustandsübergänge aller Speicherglieder erfolgen gleichzeitig, synchron zu einem Taktsignal

Notation

Beschreibung endlicher synchroner Automaten:

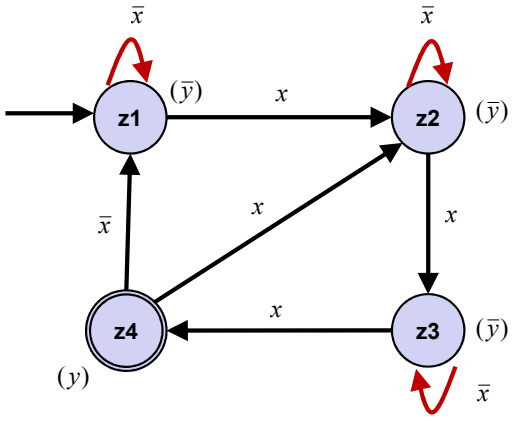
Eingabealphabet: $\mathbf{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_l\}$
Ausgabealphabet: $\mathbf{Y} = \{\mathbf{y}_1, \ldots, \mathbf{y}_m\}$
Zustandsmenge: $\mathbf{Z} = \{\mathbf{z}_1, \ldots, \mathbf{z}_k\}$
Anfangszustand: $\mathbf{z}(t_0) \in \mathbf{Z}$
Menge der Endzust.: $\mathbf{E} \subseteq \mathbf{Z}$
Übergangsfunktion: $g : (\mathbf{x}_\lambda, \mathbf{z}_\kappa) \rightarrow \mathbf{z}_r$
Ausgabefunktion: $f : (\mathbf{x}_\lambda, \mathbf{z}_\kappa) \rightarrow \mathbf{y}_\mu$

Automat: $\mathbf{A} = (\mathbf{X}, \mathbf{Z}, \mathbf{Y}, \mathbf{z}(t_0), \mathbf{E}, f, g)$

Weitere Vereinbarungen:

Eingabezeichen: $\mathbf{x}_\lambda = (x_1, \ldots, x_l) \in \mathbf{X}$
Ausgabezeichen: $\mathbf{y}_\mu = (y_1, \ldots, y_m) \in \mathbf{Y}$
Zustand: $\mathbf{z}_\kappa = (z_1, \ldots, z_k) \in \mathbf{Z}$
Eingangsvariable: $x_\lambda \in \mathbf{U}$
Ausgangsvariable: $y_\mu \in \mathbf{V}$
Zustandsvariable: $z_\kappa \in \mathbf{W}$
Menge der Eing.-var.: $\mathbf{U} = \{x_1, \ldots, x_l\}$
Menge der Ausg.-var.: $\mathbf{V} = \{y_1, \ldots, y_m\}$
Menge der Zust.-var.: $\mathbf{W} = \{z_1, \ldots, z_k\}$

Grafische Darstellung



$\mathbf{X} = \{(x), (\bar{x})\}$
$\mathbf{Y} = \{(y), (\bar{y})\}$
$\mathbf{Z} = \{\mathbf{z}_1, \mathbf{z}_2, \mathbf{z}_3, \mathbf{z}_4\}$
$\mathbf{z}(t_0) = \mathbf{z}_1$
$\mathbf{E} = \{\mathbf{z}_4\}$

Was fehlt?

⇒ **Prüfung auf:** Vollständigkeit und Widerspruchfreiheit

Getaktete Automaten

Theoretische Informatik

- Verarbeitung von EIngabezeichen sofern vorhanden
- Jedes Zeichen in der Eingabe wird einmalig verarbeitet

Technische Informatik

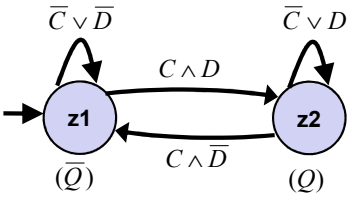
Taktung des Automaten mit einem Taktsignal → Abtatstung der Eingabe
Konsequenzen:

- Zeichen für "keine Eingabe "erforderlich
- Abtastung "derselben Eingabe "in aufeinanderfolgenden Takten möglich

⇒ Korrekte Modellierung des Taktsignals erforderlich

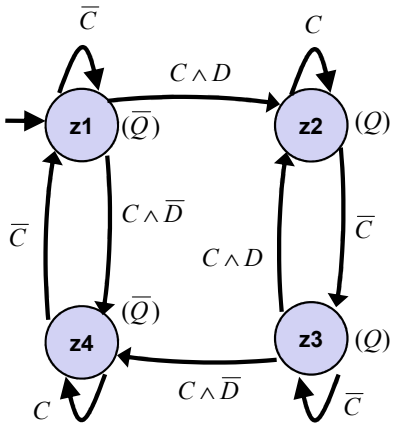
Beispiel - Taktzustandsgesteuertes D-Flip-Flop (Latch)

(Taktsignal C als Eingabe)



$$\mathbf{X} = \{(\overline{C},\overline{D}),(\overline{C},D),(C,\overline{D}),(C,D)\}$$
$$\mathbf{Y} = \{(\overline{Q}),(Q)\}$$
$$\mathbf{Z} = \{\mathbf{z}_1,\mathbf{z}_2\}$$
$$\mathbf{z}(t_0) = \mathbf{z}_1$$
$$\mathbf{E} = \{\}$$

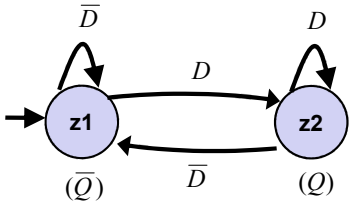
Beispiel - Taktflankengesteuertes D-Flip-Flop
(Taktsignal C als Eingabe)



$$\mathbf{X} = \{(\overline{C},\overline{D}),(\overline{C},D),(C,\overline{D}),(C,D)\}$$
$$\mathbf{Y} = \{(\overline{Q}),(Q)\}$$
$$\mathbf{Z} = \{\mathbf{z}_1,\mathbf{z}_2,\mathbf{z}_3,\mathbf{z}_4\}$$
$$\mathbf{z}(t_0) = \mathbf{z}_1$$
$$\mathbf{E} = \{\}$$

➔ Zu kompliziert und auch nicht notwendig!

Beispiel - Taktflankengesteuertes D-Flip-Flop
(Definition: Zustandsübergang nur bei taktflanke)



$$\mathbf{X} = \{(\overline{D}),(D)\}$$
$$\mathbf{Y} = \{(\overline{Q}),(Q)\}$$
$$\mathbf{Z} = \{\mathbf{z}_1,\mathbf{z}_2\}$$
$$\mathbf{z}(t_0) = \mathbf{z}_1$$
$$\mathbf{E} = \{\}$$

3.1.3 SM-Charts

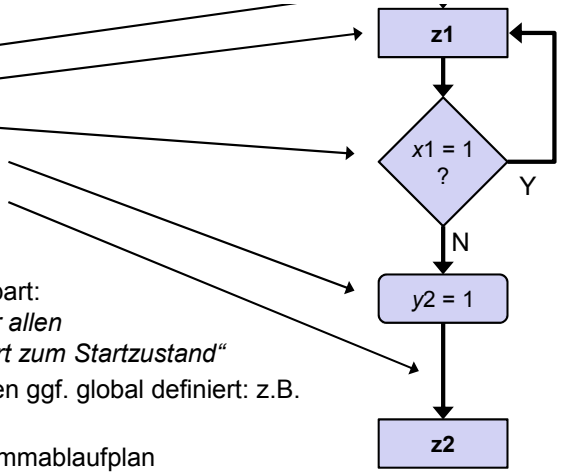
Basiselemente:

- Start
- Zustand
- Verzweigung
- Ausgabe (Mealy/Moore)
- Verbindung / Leserichtung

Weiteres:

- Reset i. Allg. global vereinbart:
  z.B. *„Reset hat Priorität vor allen anderen Eingaben und führt zum Startzustand“*
- Standardwerte für Ausgaben ggf. global definiert: z.B.
  „Standardmäßig: y2 = 0“.

Achtung: SM Chart **!=** Programmablaufplan



Verzweigungen

Weitere Merkmale:

- Selbstdefinierte Abkürzungen für komplexe Ausdrücke üblich

Teil II

Entwurf eingebetteter Systeme

Teil III

Parallelverarbeitung

Teil IV

Appendix

1 VLSI-Systementwurf Praktikum

Aufgabe 1

Implementieren Sie in VHDL einen Dekoder für die Umwandlung einer 4-Bit-Binärzahl in die 7-Segment-Darstellung einer Hexadezimalziffer. Die Position der Segmente a bis g können Sie der Beschreibung des Praktikumsboards entnehmen. Beachten Sie, dass die Ansteuerung der Segmente low-aktiv erfolgt.

Für die Implementierung ist es zweckmäßig statt 8 Einzelsignalen (a bis g sowie Dezimalpunkt) einen 8-Bit-Signalvektor als Ausgangssignal vorzusehen. **Hinweis:** Nutzen Sie `case`- oder `select`-Statements zur Beschreibung des Dekoders.

Für die Eingabe der Binärzahl sind die Schiebeschalter SW3 bis SW0 zu nutzen. Steuern Sie nur das rechte Segment der 4-stelligen Anzeige an.

Hinweis: Die Aufgabe ist als Schaltnetz (ohne Taktsignal) zu lösen.

Überprüfen Sie die korrekte Funktion des Dekoders auf dem Praktikumsboard. Werten Sie die benötigten FPGA-Ressourcen im Praktikumsprotokoll aus.

1.1 Aufgabe 1 - Binär-Dekoder

1.1.1 Entwurf

Input 4-Bit Binärzahl durch Schieberegister SW3 ... SW0

Output 7-Segmente Darstellung einer Hexadezimalziffer (8 Einzelsignale = 7 Segmente + 1 Punkt)

Input				Output								
SW3	SW2	SW1	SW0	Hex	A	B	C	D	E	F	G	dot
0	0	0	0	0	0	0	0	0	0	0	1	1
0	0	0	1	1	1	0	0	1	1	1	1	1
0	0	1	0	2	0	0	1	0	0	1	0	1
0	0	1	1	3	0	0	0	0	1	1	0	1
0	1	0	0	4	1	0	0	1	1	0	0	1
0	1	0	1	5	0	1	0	0	1	0	0	1
0	1	1	0	6	0	1	0	0	0	0	0	1
0	1	1	1	7	0	0	0	1	1	1	1	1
1	0	0	0	8	0	0	0	0	0	0	0	1
1	0	0	1	9	0	0	0	0	1	0	0	1
1	0	1	0	A	0	0	0	1	0	0	0	1
1	0	1	1	b	1	1	0	0	0	0	0	1
1	1	0	0	C	0	1	1	0	0	0	1	1
1	1	0	1	d	1	0	0	0	0	1	0	1
1	1	1	0	E	0	1	1	0	0	0	0	1
1	1	1	1	F	0	1	1	1	0	0	0	1

1.7.1 Decoder.vhdl Code

1.1.2 Auswertung

Ressourcenbedarf

- 7 Logik-Elemente
- 12 Pins

Aufgabe 2

Implementieren Sie in VHDL ein Schaltnetz zur Bestimmung der Hamming-Distanz zweier 4-Bit-Worte.

Für die Ein- und Ausgabe sind zu verwenden:

Wort 1	Schiebeschalter SW3 bis SW0
Wort 2	Schiebeschalter SW7 bis SW4
Ergebnis	rechte Ziffer des 7-Segment-Blockes

Überprüfen Sie die korrekte Funktion des Schaltnetzes auf dem Praktikumsboard. Werten Sie die benötigten FPGA-Ressourcen im Praktikumsprotokoll aus.

1.2 Aufgabe 2 - Hamming-Distanz

1.2.1 Entwurf

Input 2 4-Bit Werte

- 1.Wert: 4-Bit Binärzahl durch Schieberegister SW3 ... SW0
- 2.Wert: 4-Bit Binärzahl durch Schieberegister SW7 ... SW4

Output 7-Segmente Darstellung einer Hexadezimalziffer (8 Einzelsignale = 7 Segmente + 1 Punkt)

Ansatz SW3 ... SW0 und SW7 ... SW4 logisch xor verknüpfen und Ergebnis direkt auf 7-Segmente Anzeige mappen

(1.7.2 Hamming.vhdl Code)

1.2.2 Auswertung

Ressourcenbedarf

- 9 Logik-Elemente
- 16 Pins

Aufgabe 3

Entwickeln Sie einen Modulo-*n*-Zähler, der bei einer Taktung mit 50 MHz einen Impuls pro Sekunde erzeugt. Die Impulslänge soll 1 Taktperiode betragen. Nutzen Sie die so erzeugte Impulsfolge zum periodischen Linksrotieren eines zyklischen 10-Bit-Schieberegisters um eine Bitstelle pro Sekunde.

Hinweise:

- Alle Register sind synchron mit ein- und demselben Taktsignal zu takten.
- Verwenden Sie einen additiven Operator zur Realisierung des Zählers.
- Bei einem Reset ist das Schieberegister mit einer ‚1' an der rechten Stelle zu initialisieren.
- Realisieren Sie die Rotation durch eine geeignete Konkatenation.

Für die Ein- und Ausgabe sind zu verwenden:

Takt	50 MHz
Reset	Schiebeschalter SW0
Schieberegister	LED-Zeile

a) Bei welchem Zählerstand ist der Zähler zurückzusetzen? Wann muss die Ausgabe des Impulses erfolgen?

b) Welcher Funktion entspricht der Impuls aus Sicht des Schieberegisters?

c) Implementieren Sie das Schaltwerk in VHDL. Überprüfen Sie die korrekte Funktion des Impulses und des Schieberegisters im Simulator.

d) Überprüfen Sie die korrekte Funktion des Schaltwerks auf dem Praktikumsboard. Werten Sie die benötigten FPGA-Ressourcen und die maximale Taktfrequenz im Praktikumsprotokoll aus.

1.3 Aufgabe 3 - Modulo-n-Zähler

1.3.1 Entwurf

a) Der Zähler ist nach 50 Millionen Schritten zurückzusetzen (50 MHz Takt entspricht 50 Millionen Taktperioden pro Sekunde)

b) Für das Schieberegister ist der Zählerzustand ein Enable-Signal

c)

Input

- 50MHz Takt
- Reset (Schiebeschalter SW0)

Output LED-Zeile

Ansatz 2 Komponenten: Schieber und Zähler

Zähler gibt alle 50-Millionen Taktperioden (50MHz Takt ergibt $50 \cdot 10^6$ Taktschritte pro Sekunde) einen Takt lang ein enable-Signal aus. (1.7.3 Zaehler.vhdl-Code)

Schieber beinhaltet den Zähler als Komponente und verschiebt bei dessen enable-Signal die LED-Anzeige um eine Stelle pro Takt. (1.7.3 Schieber.vhdl-Code)

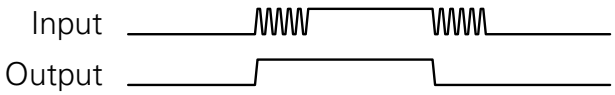
1.3.2 Auswertung

Ressourcenbedarf

- 60 Logik-Elemente
- davon 38 dedizierte Logik-Elemente
- 12 Pins
- maximale Taktfrequenz von 250 MHz

Aufgabe 4

Die Taster des Praktikumsboards sind – abhängig von der Revision des Boards – nicht entprellt. Ihre Prelldauer beträgt bis zu 3 ms.



Waveform eines prellenden Eingangssignals und des entsprechenden entprellten Ausgangssignals.

Entwickeln Sie – unabhängig davon, ob ihr Board entprellte Taster besitzt – einen Automaten, der mit Hilfe eines Zählers nach einer am Eingang vom Taster erkannten Flanke für diese Dauer alle weiteren ignoriert und so ein entsprechend entprelltes Ausgangssignal liefert. Der Zähler ist hierbei mit dem Boardtakt von 50 MHz zu takten. Nutzen Sie diesen Entprellautomaten zum sicheren Ein- und Ausschalten einer LED jeweils durch den Druck desselben Tasters. Nutzen Sie für die Ansteuerung der LED einen zweiten Automaten.

Für die Ein- und Ausgabe sind zu verwenden:

Takt	50 MHz
Eingabe	Taster BTN2
Ausgabe	LED0

a) Erstellen Sie jeweils State-Machine-Charts für den Entprellautomaten und den LED-Automaten. Abfrage und Steuerung des Zählers erfolgt durch den Entprellautomaten mittels geeigneter selbstdefinierter Signale.

b) Welcher Typ von Automatenkopplung ist zu verwenden? Über welche Signale erfolgt die Kopplung?

c) Implementieren Sie beide Automaten als getrennte VHDL-Module und überprüfen Sie die korrekte Funktion mittels Simulation.

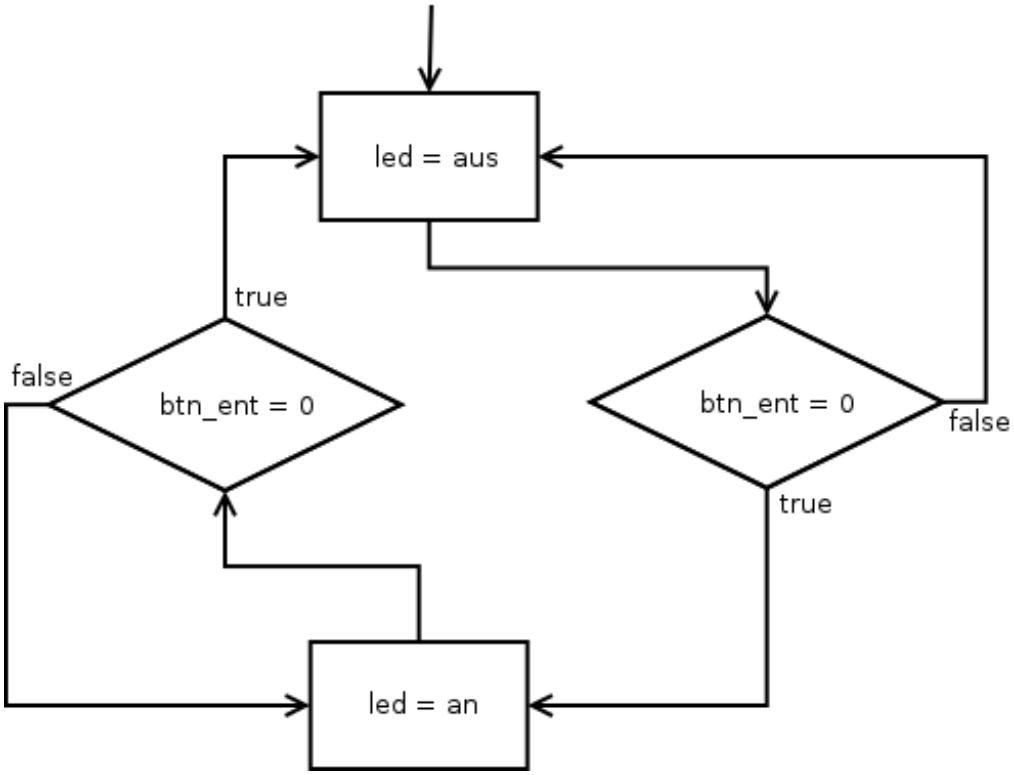
d) Implementieren Sie ein Top-Level-Modul welches beide Automaten miteinander koppelt. Überprüfen Sie die korrekte Funktion des Schaltwerks auf dem Praktikumsboard. Werten Sie die benötigten FPGA-Ressourcen und die maximale Taktfrequenz im Praktikumsprotokoll aus.

1.4 Aufgabe 4 - Entprell-Automat

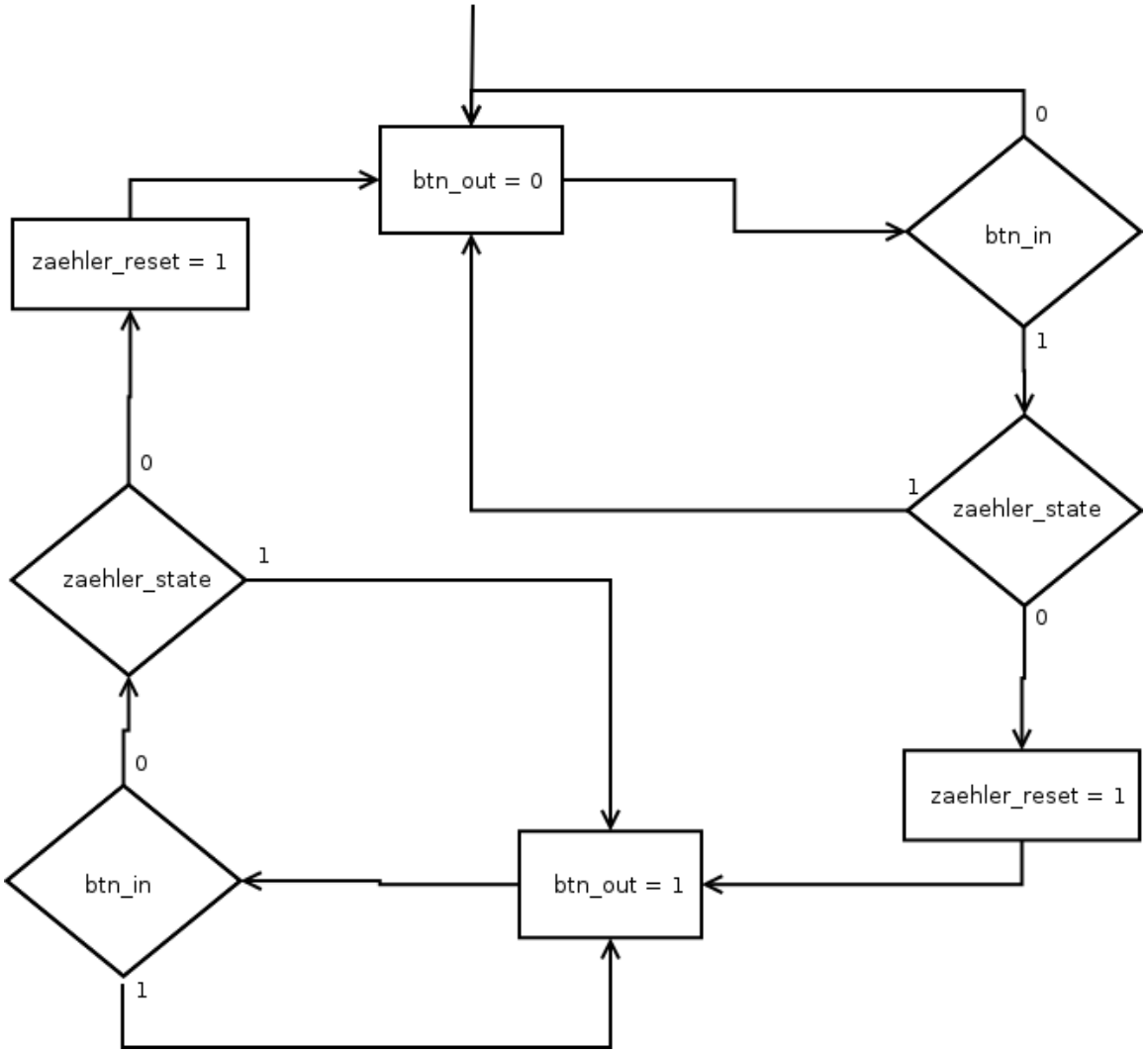
1.4.1 Entwurf

State-Machine-Charts

LED enthält die Komponente Entprellung und verbindet die Ein- und Ausgangssignale. (1.7.4 LED.vhdl-Code)



Entprellung enthält die Komponente Zaehler, der bei der Veränderung des Eingangsignals gestartet wird und für 3ms weitere Änderungen ignoriert. (1.7.4 Entprellung.vhdl-Code)



Zaehler implementiert einen Zähler, der durch ein Signal definierte Schritte zählt. Ausgegeben wird der aktuelle Zustand des Zählers. Eingegeben ein Reset-Signal. (1.7.4 Zaehler.vhdl-Code)

Kopplung

Es wird eine synchrone Automatenkopplung über die Ausgangssignale mit einem Taktsignal verwendet.

1.4.2 Auswertung

Ressourcenbedarf

- 86 Logik-Elemente
- davon 79 dedizierte Logik-Elemente
- 44 Register
- 3 Pins
- maximale Taktfrequenz von 178 MHz

Aufgabe 5

Entwickeln Sie eine Multiplex-Ansteuerung für die 4-stellige 7-Segment-Anzeige des Erweiterungsboards um eine längere Zeichenkette auszugeben. Dabei soll jede halbe Sekunde der Text eine Stelle nach links verschoben werden.

Implementieren sie einen Zähler welcher Impulse mit geeigneter Frequenz ausgibt. Verwenden Sie für den Zähler den 50 MHz-Takt.

Zeigen Sie die Zeichenkette „HALLO" an, wobei beginnend mit vollständig leerer Anzeige die Zeichenkette von rechts nach links die vier Stellen der Anzeige durchlaufen soll. Ist das letzte Zeichen nach links aus der Anzeige gewandert soll die Zeichenkette erneut ausgegeben werden.

Durch die Betätigung des Resets sollen alle vier 7-Segment-Blöcke gelöscht und danach wieder mit der Anzeige des ersten Zeichens der Zeichenkette in der rechten Stelle begonnen werden

Für die Ein- und Ausgabe sind zu verwenden:

Takt	50 MHz
Reset	Schiebeschalter SW0
Ausgabe	7-Segment-Block

Gliedern sie ihren Systementwurf mindestens in drei Module: Top-Level, Decoder für Textzeichen und Multiplexer für den anzuzeigenden Teil der Zeichenkette.

a) Wie viele Bits werden für die Kodierung eines Textzeichens benötigt? Kodieren Sie die Textzeichen mit einem eigenen Code und dokumentieren Sie diesen im Protokoll!

b) Implementieren Sie Decoder und Multiplexer als getrennte VHDL-Module und überprüfen Sie die korrekte Funktion mittels Simulation.

c) Implementieren Sie ein Top-Level-Modul welches beide Automaten miteinander koppelt. Überprüfen Sie die korrekte Funktion des Schaltwerks auf dem Praktikumsboard. Werten Sie die benötigten FPGA-Ressourcen und die maximale Taktfrequenz im Praktikumsprotokoll aus.

1.5 Aufgabe 5 - HALLO-Anzeige

1.5.1 Entwurf

zu a) Es müssen 5 Zeichen kodiert werden (H, A, L, O, Leerzeichen).

$ld\,5 = 3$

Daher werden für eine Binärkodierung mindestens 3 Bits benötigt.

Input			Output								
BIT2	BIT1	BIT0	CHAR	A	B	C	D	E	F	G	DOT
0	0	0		1	1	1	1	1	1	1	1
0	0	1	H	1	0	0	1	0	0	0	1
0	1	0	A	0	0	0	1	0	0	0	1
0	1	1	L	1	1	1	0	0	0	1	1
1	0	0	O	0	0	0	0	0	0	1	1

b) Für das Schieberegister ist der Zählerzustand ein Enable-Signal

c) (1.7.5 Hallo.vhdl-Code)

1.5.2 Auswertung

Ressourcenbedarf

- 73 Logik-Elemente
- 61 Register
- 34 Pins
- maximale Taktfrequenz von 262 MHz

Aufgabe 6

Implementieren Sie eine auf Zehntelsekunden genaue Stoppuhr mit dem Boardtakt von 50 MHz als Referenz. Das Starten und Anhalten der Stoppuhr soll durch den Druck eines zu entprellenden Tasters ausgelöst werden. Das Rücksetzen der Stoppuhr soll durch das globale Reset erfolgen. Der aktuelle Stand der Stoppuhr ist dezimal auf dem 4-stelligen 7-Segment-Block auszugeben. Dabei sind eine Stelle für die Minutenzählung, zwei für die Sekunden und die verbleibende für die Zehntelsekunden vorzusehen. Verwenden Sie die Dezimalpunkte zur passenden optischen Unterteilung der Anzeige. Die Zeitmessung soll, entsprechend der verfügbaren Stellenzahl, „modulo 10 Minuten“ erfolgen.

Hinweise:

- Nutzen Sie einen BCD-Zähler pro Stelle mit einem entsprechenden Wertebereich.
- Kaskadieren Sie die BCD-Zähler mit Hilfe von Übertragsimpulsen.

Für die Ein- und Ausgabe sind zu verwenden:

Takt	50 MHz
Reset	Schiebeschalter SW0
Start/Stopp	Taster BTN2
Ausgabe	7-Segment-Anzeige

a) Überlegen Sie sich eine geeignete Zerlegung des Gesamtsystems in Teilkomponenten und spezifizieren Sie die deren Schnittstellen. Besprechen Sie kurz ihre Lösung mit dem Praktikumsbetreuer.

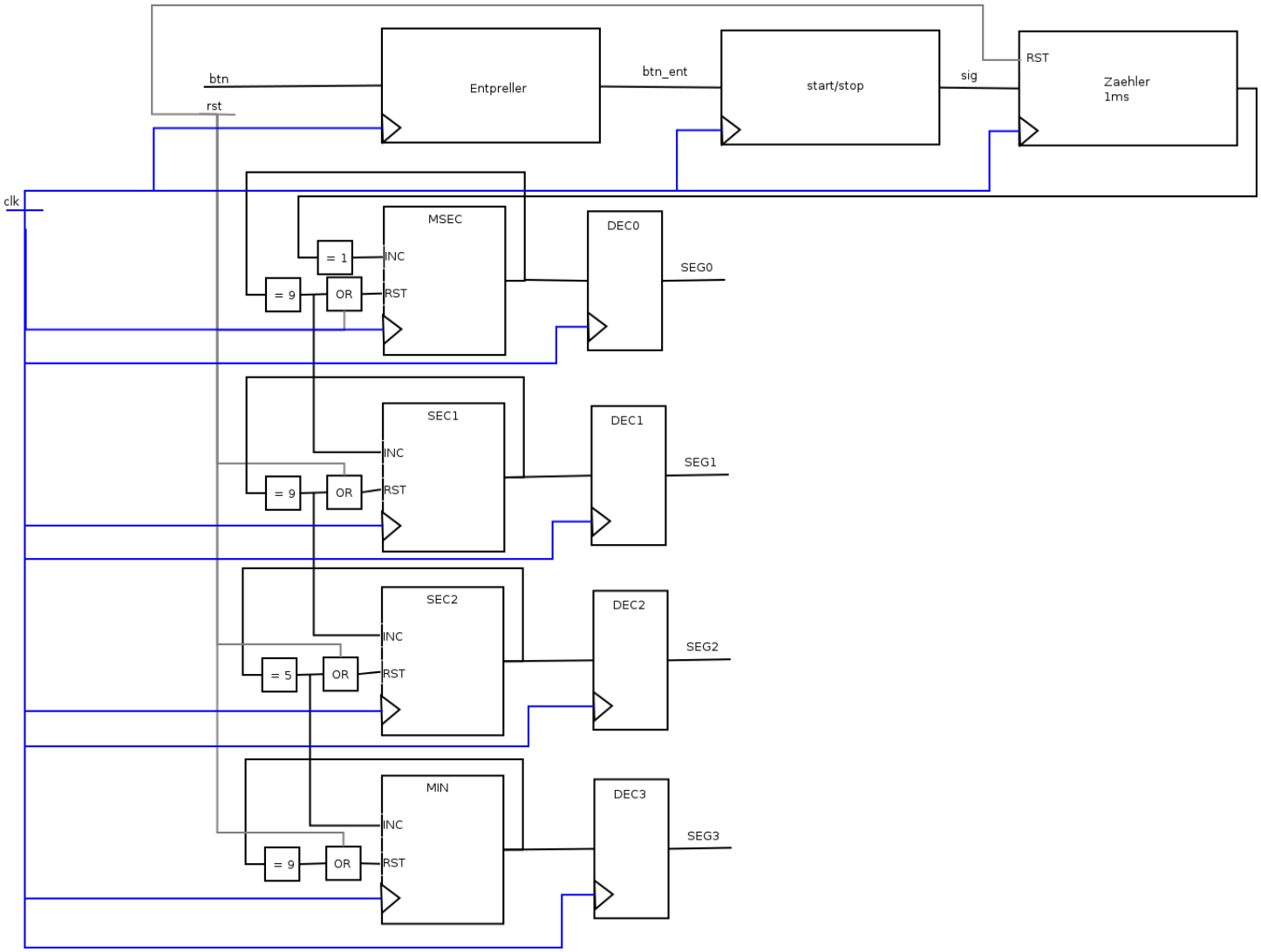
b) Welcher Typ von Automatenkopplung ist zu verwenden? Über welche Signale erfolgt die Kopplung?

c) Implementieren Sie die Teilkomponenten in VHDL. Erstellen Sie dazu für jeden Automaten ein State-Machine-Chart. Überprüfen Sie die korrekte Funktion jeder Teilkomponente mittels Simulation.

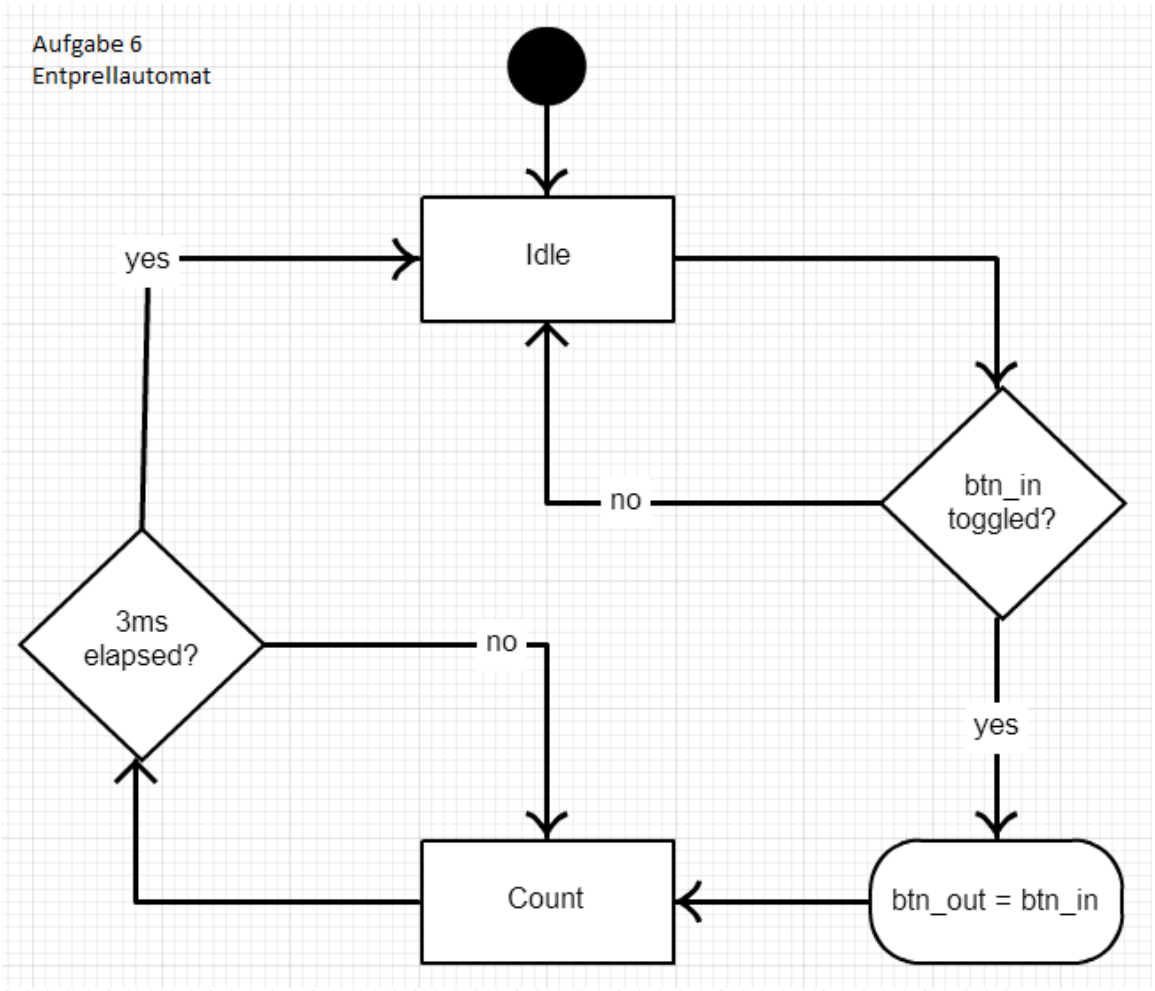
d) Implementieren Sie ein Top-Level-Modul welches alle Komponenten miteinander verbindet. Überprüfen Sie die korrekte Funktion der Schaltung auf dem Praktikumsboard. Werten Sie die benötigten FPGA-Ressourcen und die maximale Taktfrequenz im Praktikumsprotokoll aus.

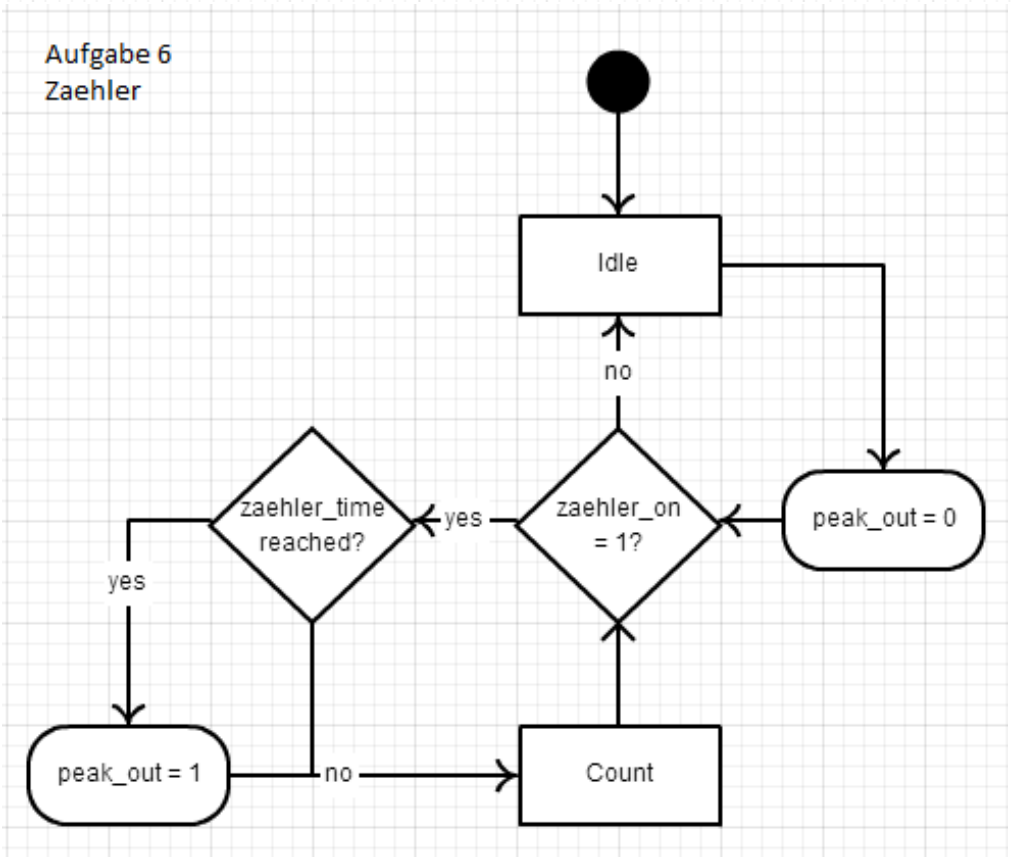
1.6 Aufgabe 6 - Stoppuhr

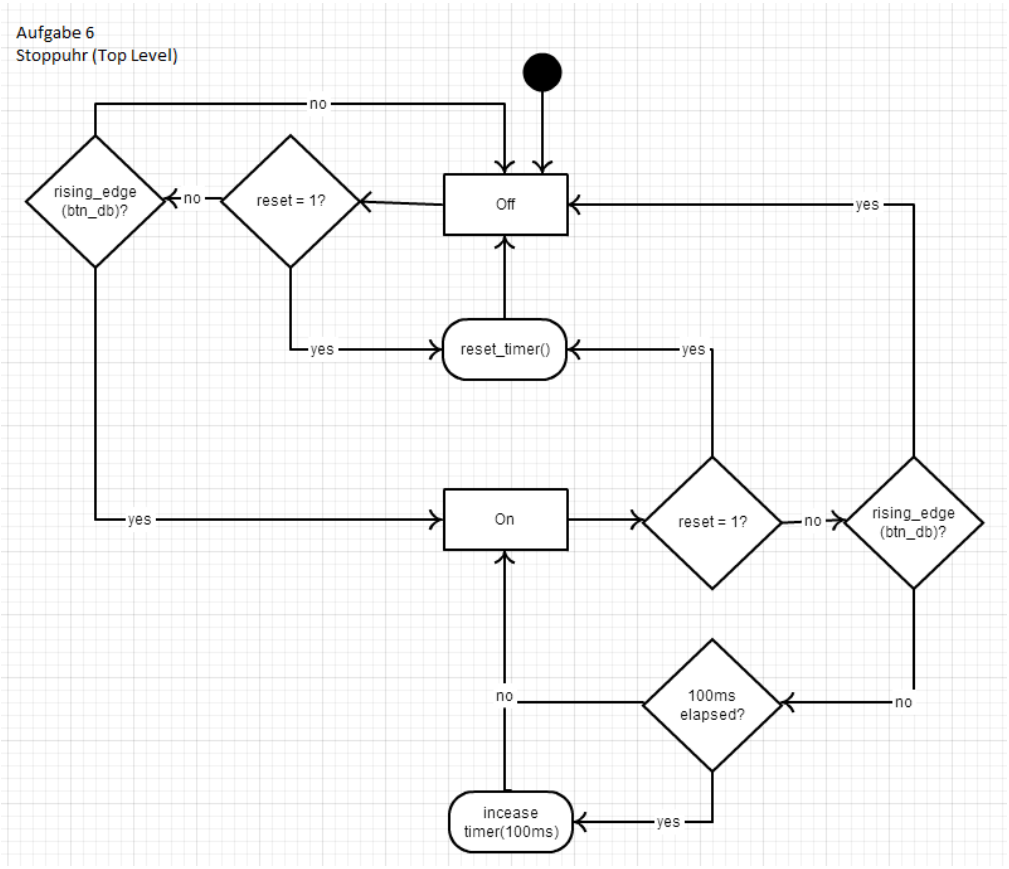
1.6.1 Entwurf



State-Machine-Charts







Kopplung
Es wird eine synchrone Automatenkopplung über die Ausgangssignale mit einem Taktsignal verwendet.

1.6.2 Auswertung

Ressourcenbedarf

- 163 Logik-Elemente
- davon 121 dedizierte Logik-Elemente
- 35 Pins
- maximale Taktfrequenz von 225 MHz

1.7 Anhang

1.7.1 01-Aufgabe Code

Listing 1.1: VHDL-Code Decoder.vhdl

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.numeric_std.all;
4
5  entity Decoder is
6
7    port (
8      sw : in std_logic_vector(3 downto 0);
9      cc : out std_logic_vector(7 downto 0));
10
11 end Decoder;
12
13 architecture Dec1 of Decoder is
14 begin
```

```
15
16    ----------------------------------------------------------------
17    -- Outputs: 4 bit breites Wort in Hexadezimale 7-Segment Anzeige
18    ----------------------------------------------------------------
19    with sw select
20      cc <= "00000011" when "0000",
21          "10011111" when "0001",
22          "00100101" when "0010",
23          "00001101" when "0011",
24          "10011001" when "0100",
25          "01001001" when "0101",
26          "01000001" when "0110",
27          "00011111" when "0111",
28          "00000001" when "1000",
29          "00001001" when "1001",
30          "00010001" when "1010",
31          "11000001" when "1011",
32          "01100011" when "1100",
33          "10000101" when "1101",
34          "01100001" when "1110",
35          "01110001" when "1111";
36  end dec1;
```

1.7.2 02-Aufgabe Code

Listing 1.2: VHDL-Code Hamming.vhdl

```
1   library ieee;
2   use ieee.std_logic_1164.all;
3   use ieee.numeric_std.all;
4
5   entity Hamming is
6
7     port (
8       sw1 : in std_logic_vector(3 downto 0); -- Erstes 4bit Wort
9       sw2 : in std_logic_vector(3 downto 0); -- Zweites 4bit Wort
10      cc : out std_logic_vector(7 downto 0)); -- 7 Segment Ausgabe
11
12  end Hamming;
13
14  architecture ham1 of Hamming is
15    signal xo : std_logic_vector(3 downto 0);
16  begin
17
18    xo <= sw1 xor sw2; -- Jede Stelle nur 1, wenn sich die Woerter an der Stelle unterscheiden
19
20    -- Je nach Anzahl der Einsen im Signal "xo" wird die Ausgabe 0-4 ausgegeben.
21    with xo select
22      cc <= "00000011" when "0000",
23          "10011111" when "0001",
24          "10011111" when "0010",
25          "00100101" when "0011",
26          "10011111" when "0100",
27          "00100101" when "0101",
28          "00100101" when "0110",
29          "00001101" when "0111",
30          "10011111" when "1000",
31          "00100101" when "1001",
32          "00100101" when "1010",
33          "00001101" when "1011",
34          "00100101" when "1100",
35          "00001101" when "1101",
36          "00001101" when "1110",
37          "10011001" when "1111";
38  end ham1;
```

1.7.3 03-Aufgabe Code

Listing 1.3: VHDL-Code Schieber.vhdl

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.numeric_std.all;
4  use Ieee.std_logic_unsigned.all;
5
6  entity Schieber is
7
8    port (
9      clk : in std_logic;
10     rst : in std_logic;
11     ld : out std_logic_vector(9 downto 0)
12     );
13
14
15 end Schieber;
16
17
18 architecture schieb1 of Schieber is
19   component zaehler            -- komponente zaehler in architektur einbinden
20     port (
21       clk : in std_logic;
22       clk_out : out std_logic
23       );
24   end component;
25   signal state : std_logic_vector(9 downto 0) := "0000000001"; -- initialanzeige der led-zeile
26   signal shift : std_logic;
27
28 begin
29   custom_clk : zaehler PORT MAP (clk => clk, clk_out => shift);
30
31   process(clk)
32   begin
33
34     if rising_edge(clk) then
35       if rst = '1' then          -- bei reset zustand der led-zeile zuruecksetzen
36         state <= "0000000001";
37       elsif shift = '1' then         -- kein reset und signal vom zaehler
38         state <= state(8 downto 0)&state(9);    -- zustand der linken led, rechts wieder einfuegen
39       end if;
40     end if;
41   end process;
42
43   ld <= state;
44
45 end schieb1;
```

Listing 1.4: VHDL-Code Zaehler.vhdl

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.numeric_std.all;
4
5  entity Zaehler is
6
7    port (
8      clk : in std_logic;
9      clk_out : out std_logic
10     );
11
12
13 end Zaehler;
14
15 architecture zae1 of Zaehler is
```

```
16
17
18      signal counter : unsigned(26 downto 0) := (others => '0'); -- Zaehler mod 50.000.000
19      signal state : std_logic := '1';        -- zaehler-zustand (1 => fertig, 0 => zaehlt)
20  begin
21
22    process(clk, state, counter)
23    begin
24
25      if rising_edge(clk) then
26
27        state <= '0';
28        if counter = to_unsigned(50000000, counter'length) then -- prueft ob counter == 50 mio
29          counter <= (others => '0');       -- falls true, ist 1 sekunde verstrichen -> counter reset
30          state <= '1';          -- zaehler-zustand auf fertig setzen
31
32        else
33          counter <= counter + 1;        -- sonst weiterzaehlen
34        end if;
35
36      end if;
37    end process;
38
39    clk_out <= state;
40  end zae1;
```

1.7.4 04-Aufgabe Code

Listing 1.5: VHDL-Code LED.vhdl

```
1   library ieee;
2   use ieee.std_logic_1164.all;
3   use ieee.std_logic_unsigned.all;
4   use ieee.numeric_std.all;
5
6   entity LED is
7
8     port (
9       clk : in std_logic;
10      btn : in std_logic;
11      ld : out std_logic
12      );
13  end LED;
14
15  architecture led1 of LED is
16    -- Komponente Entprellung fuer das btn-Signal wird eingebunden
17    component Entprellung
18      port (
19        clk : in std_logic;
20        btn_in : in std_logic;
21        btn_out : out std_logic
22        );
23    end component;
24
25    signal led_sig : std_logic := '0';
26    signal btn_led : std_logic;
27    signal btn_out : std_logic;
28    signal btn_out_d : std_logic;
29    signal btn_in_d :std_logic := '0';
30    signal btn_in :std_logic := '0';
31
32    begin
33    custom_entpreller : Entprellung PORT MAP (
34                    clk => clk,
35                    btn_in => btn_in,
36                    btn_out => btn_out);
```

```
37   process (btn_led )
38   begin
39       if btn_led = '0' then -- aenderung der led bei uebergang des entprellten btn-signals in den aktiven
             zustand
40         led_sig <= not led_sig;
41       end if;
42   end process;
43
44   -- synchronisierung der signale
45   process (clk)
46   begin
47     if rising_edge(clk) then
48       btn_in <= btn_in_d;
49       btn_in_d <= btn;
50       btn_out_d <= btn_out;
51       btn_led <= btn_out_d;
52     end if;
53   end process;
54
55
56   ld <= not led_sig;
57 end led1;
```

Listing 1.6: VHDL-Code Entprellung.vhdl

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.std_logic_unsigned.all;
4  use ieee.numeric_std.all;
5
6
7  entity Entprellung is
8
9    port (
10     clk : in std_logic;
11     btn_in : in std_logic;
12     btn_out : out std_logic
13     );
14
15 end Entprellung;
16
17 architecture entprel1 of Entprellung is
18
19   signal btn_old : std_logic := '0';
20   signal state : std_logic := '0';
21
22   signal zaehler_state : std_logic :='0';
23   signal zaehler_state_d : std_logic :='0';
24   signal zaehler_reset : std_logic := '0';
25
26
27
28   -- Zaehler-Komponente wird eingebunden
29   component Zaehler
30     port (
31       clk : in std_logic;
32       count_steps : in unsigned(31 downto 0);
33       counter_reset : in std_logic;
34       counter_state : out std_logic
35       );
36   end component;
37
38 begin
39   custom_zaehler : Zaehler PORT MAP (
40               clk => clk,
41               count_steps => to_unsigned(150000, 32),
```

```
42                -- zu zaehlende Schritte, bis deaktivierung des counter_state signals
43                counter_state => zaehler_state,
44                counter_reset => zaehler_reset);
45
46    -- entprellung des eingangsignals btn_in
47    process(clk)
48    begin
49      if rising_edge(clk) then
50        if state = '0' then -- falls ausserhalb der prelldauer
51          if btn_in /= btn_old then -- falls das eingangssignal sich aendert, wird der entpreller gestartet
52            zaehler_reset <= '1';
53            btn_old <= btn_in;
54          end if;
55        else
56          -- zaehler_reset soll nur einen takt aktiv sein
57          if zaehler_reset = '1' then
58            zaehler_reset <= '0';
59          end if;
60        end if;
61      end if;
62    end process;
63
64    -- synchronisierung des zaehler-zustands
65    process (clk)
66    begin
67      if rising_edge(clk) then
68        state <= zaehler_state_d;
69        zaehler_state_d <= zaehler_state;
70      end if;
71    end process;
72
73    -- ausgabe des entprellten signals
74    btn_out <= btn_old;
75  end entprel1;
```

Listing 1.7: VHDL-Code Zaehler.vhdl

```
1   library ieee;
2   use ieee.std_logic_1164.all;
3   use Ieee.std_logic_unsigned.all;
4   use ieee.numeric_std.all;
5
6   entity Zaehler is
7
8     port (
9       clk : in std_logic;
10      count_steps : in unsigned(31 downto 0); -- zu zaehlende taktschritte, bis zur deaktivierung des
            counter_state signals
11      counter_reset : in std_logic;
12      counter_state : out std_logic);
13
14
15  end Zaehler;
16
17  architecture zae1 of Zaehler is
18
19    signal reset : std_logic := '0';
20    signal reset_d : std_logic := '0';
21    signal state : std_logic := '0';
22    signal counter : unsigned(31 downto 0) := (others => '0');
23  begin
24
25    process(clk)
26    begin
27
28      if rising_edge(clk) then
```

```
29      if reset = '1' then -- zuruecksetzen des zaehlers
30        counter <= (others => '0');
31      end if;
32
33      if counter < count_steps then -- zaehler aktiv
34        state <= '1';
35        counter <= counter + 1;
36      else
37        state <= '0';
38      end if;
39    end if;
40  end process;
41
42  -- synchronisierung des reset-signals
43  process(clk)
44  begin
45    if rising_edge(clk) then
46      reset_d <= counter_reset;
47      reset <= reset_d;
48    end if;
49  end process;
50
51
52  counter_state <= state;
53 end zae1;
```

1.7.5 05-Aufgabe Code

Listing 1.8: VHDL-Code hallo.vhdl

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.std_logic_unsigned.all;
4  use ieee.numeric_std.all;
5
6  ------------------------------------------------------------------
7  -- Top Level:
8  -- Verbindet den Multiplexer mit 4 Decodern und legt den Ausgang
9  -- je eines Decoders an eine 7 Segment Anzeige
10 ------------------------------------------------------------------
11 entity Hallo is
12
13   port (
14     clk : in std_logic;
15     rst : in std_logic;
16     seg1 : out std_logic_vector(7 downto 0);  --------------
17     seg2 : out std_logic_vector(7 downto 0);  -- 7 Segment
18     seg3 : out std_logic_vector(7 downto 0); -- Ausgaenge
19     seg4 : out std_logic_vector(7 downto 0)); --------------
20
21 end Hallo;
22
23 architecture hello of hallo is
24
25   component Multiplex
26     port (
27       clk : in std_logic;
28       rst : in std_logic;
29       led_out : out std_logic_vector(11 downto 0)
30     );
31   end component;
32
33   component Decoder
34     port (
35       clk : in std_logic;
36       code : in std_logic_vector(2 downto 0);
```

```
37      decoded : out std_logic_vector(7 downto 0));
38
39    end component;
40
41    signal dig : std_logic_vector(11 downto 0); -- nimmt 12bit Wort aus dem Multiplexer entgegen
42    signal dig0 : std_logic_vector(2 downto 0); ----------------------------------------
43    signal dig1 : std_logic_vector(2 downto 0); -- 4*3bit die je ein Zeichen aus dem 12bit
44    signal dig2 : std_logic_vector(2 downto 0); -- Wort des Multiplexers abzweigen
45    signal dig3 : std_logic_vector(2 downto 0); ----------------------------------------
46
47  begin
48
49    ---------------------------------------------------------------------------
50    -- Multiplexer Ausgang wird an das Signal "dig" angelegt
51    -- Je ein Signal mit je einem Zeichen wird als Eingang eines Decoders angelegt
52    ---------------------------------------------------------------------------
53    mult : Multiplex PORT MAP( clk => clk, rst => rst, led_out => dig);
54    dec0 : Decoder PORT MAP(clk => clk, code => dig0, decoded => seg4);
55    dec1 : Decoder PORT MAP(clk => clk, code => dig1, decoded => seg3);
56    dec2 : Decoder PORT MAP(clk => clk, code => dig2, decoded => seg2);
57    dec3 : Decoder PORT MAP(clk => clk, code => dig3, decoded => seg1);
58
59    -- abzweigen von je 3bit (ein Zeichen) aus dem Multiplexer Ausgang
60    dig0 <= dig(11 downto 9);
61    dig1 <= dig(8 downto 6);
62    dig2 <= dig(5 downto 3);
63    dig3 <= dig(2 downto 0);
64
65  end hello;
```

Listing 1.9: VHDL-Code Decoder.vhdl

```
1   library ieee;
2   use ieee.std_logic_1164.all;
3   use ieee.std_logic_unsigned.all;
4   use ieee.numeric_std.all;
5
6   entity Decoder is
7
8     port (
9       clk : in std_logic;
10      code : in std_logic_vector(2 downto 0);
11      decoded : out std_logic_vector(7 downto 0));
12  end Decoder;
13
14  architecture decoder1 of Decoder is
15
16    signal decoded_out : std_logic_vector(7 downto 0) := (others => '0'); -- Signal, dass an den Ausgang
          "decoded" angelegt wird
17
18    begin
19
20    ---------------------------------------------------
21    -- Dekodieren eines 3bit breiten Wortes in ein 8bit
22    -- breites Wort fuer die 7 Segment Anzeige
23    ---------------------------------------------------
24    process (clk)
25    begin
26      if rising_edge(clk) then
27        case code is
28          when "000" => decoded_out <= "11111111"; -- _
29          when "001" => decoded_out <= "10010001"; -- H
30          when "010" => decoded_out <= "00010001"; -- A
31          when "011" => decoded_out <= "11100011"; -- L
32          when "100" => decoded_out <= "00000011"; -- O
33          when others => decoded_out <= "11111111";
```

```
34      end case;
35    end if;
36  end process;
37
38  decoded <= decoded_out;
39
40  end decoder1;
```

Listing 1.10: VHDL-Code Multiplex.vhdl

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.numeric_std.all;
4
5
6  entity Multiplex is
7
8    port (
9      clk : in std_logic;
10     rst : in std_logic;
11     led_out : out std_logic_vector(11 downto 0) -- 12bit breiter Ausgang (3bit je Zeichen)
12     );
13
14 end Multiplex;
15
16 -- 000: _
17 -- 001: H
18 -- 010: A
19 -- 011: L
20 -- 100: O
21
22 architecture multi of Multiplex is
23 -- _ _ _ _ H A L L O _ _ _
24   signal tex : std_logic_vector(35 downto 0) := "000000000000001010011011100000000000"; -- kompletter
       Schriftzug der einmal durchlaufen wird
25   signal counter : unsigned(24 downto 0) := (others => '0'); -- Zaehlersignal (mod 25.000.000)
26   signal mul : unsigned(3 downto 0); -- Steuersignal Multiplexer (mod 10) : 9 moegliche 12bit breite
       Teilworte des kompletten Schriftzugs
27
28 begin
29
30   ----------------------------------------------------------------
31   -- Inkrementieren des Steuersignals "mul" alle 25.000.000 Takte
32   ----------------------------------------------------------------
33   process(clk)
34   begin
35     if rising_edge(clk) then
36       if(rst = '1') then
37         counter <= (others => '0');
38         mul <= (others => '0');
39       elsif(counter = "101111101011110000011111l") then
40         counter <= (others => '0');
41         mul <= mul + 1;
42         if(mul = "1000") then
43           mul <= "0000";
44         end if;
45       else
46         counter <= counter + 1;
47       end if;
48     end if;
49   end process;
50
51   ------------------------------------------------------------------------------------------------------
52   -- Je nach Steuersignal "mul" wird ein anderes 12bit breites Teilwort des kompletten Schriftzugs
       ausgegeben
53   ------------------------------------------------------------------------------------------------------
```

```
54    with mul select
55      led_out <=  tex(35 downto 24) when "0000",
56              tex(32 downto 21) when "0001",
57              tex(29 downto 18) when "0010",
58              tex(26 downto 15) when "0011",
59              tex(23 downto 12) when "0100",
60              tex(20 downto 9) when "0101",
61              tex(17 downto 6) when "0110",
62              tex(14 downto 3) when "0111",
63              tex(11 downto 0) when "1000",
64              (others => '0') when others;
65
66  end multi;
```

1.7.6 06-Aufgabe Code

Listing 1.11: VHDL-Code Stoppuhr.vhd

```
1   library ieee;
2   use ieee.std_logic_1164.all;
3   use ieee.std_logic_unsigned.all;
4   use ieee.numeric_std.all;
5
6   entity Stoppuhr is
7
8     port (
9       clk : in std_logic;
10      rst : in std_logic;
11      onoff : in std_logic;
12      seg1 : out std_logic_vector(7 downto 0);
13      seg2 : out std_logic_vector(7 downto 0);
14      seg3 : out std_logic_vector(7 downto 0);
15      seg4 : out std_logic_vector(7 downto 0));
16
17  end Stoppuhr;
18
19  architecture uhr of Stoppuhr is
20
21    -- Zaehler: gibt jede Zehntelsekunde einen Peak aus
22    component Zaehler
23      port (
24        clk : in std_logic;
25        zaehler_time : in unsigned(31 downto 0);
26        zaehler_on : in std_logic;
27        peak_out : out std_logic
28        );
29    end component;
30
31    -- entprellt das Eingangssignal des (on/off) Buttons
32    component EntprellAutomat
33      port (
34        clk : in std_logic;
35        btn : in std_logic;
36        btnout : out std_logic
37        );
38    end component;
39
40    -- 4 Decoder: Je ein Decoder dekodiert eine Stelle der aktuellen Zeit fuer die 7-Segment Anzeige
41    component Decoder
42      port (
43        clk : in std_logic;
44        code : in std_logic_vector(3 downto 0);
45        decoded : out std_logic_vector(7 downto 0)
46        );
47    end component;
48
```

```
49    -- Steuersignal und Ausgabesignal des Zaehlers
50    signal timer_on, peak : std_logic := '0';
51
52    -- Je ein Signal fuer je eine Stelle der aktuellen Zeit
53    signal min, sec1, sec2, ms : unsigned(3 downto 0) := (others => '0');
54
55    -- Steuersignal fuer die Stoppuhr und Signale fuer den entprellten Button, sowie des alten Signalpegels
          des Buttons
56    signal running, onoff_db, onoff_old : std_logic := '0';
57
58    -- Signale fuer die 7-Segment Anzeige
59    signal segMin, segSec1, segSec2, segMs : std_logic_vector(7 downto 0) := (others => '0');
60
61
62  begin
63
64    zaehl : Zaehler PORT MAP (clk => clk,
65                    zaehler_time => to_unsigned(5000000, 32),
66                    zaehler_on => timer_on,
67                    peak_out => peak);
68
69    prell : EntprellAutomat PORT MAP (
70                    clk => clk,
71                    btn => onoff,
72                    btnout => onoff_db);
73
74    -- Jeder Decoder dekodiert eine Stelle der aktuellen Zeit
75    dec0 : Decoder PORT MAP(clk => clk, code => std_logic_vector(min), decoded => segMin);
76    dec1 : Decoder PORT MAP(clk => clk, code => std_logic_vector(sec1), decoded => segSec1);
77    dec2 : Decoder PORT MAP(clk => clk, code => std_logic_vector(sec2), decoded => segSec2);
78    dec3 : Decoder PORT MAP(clk => clk, code => std_logic_vector(ms), decoded => segMs);
79
80
81    process(clk)
82    begin
83      if rising_edge(clk) then
84        if(rst = '1') then           -- aktiver Reset setzt alles zurueck und stoppt die Uhr
85          running <= '0';
86          timer_on <= '0';
87          min <= (others => '0');
88          sec1 <= (others => '0');
89          sec2 <= (others => '0');
90          ms <= (others => '0');
91        else
92          onoff_old <= onoff_db;
93          if(onoff_db = '1' and onoff_db /= onoff_old) then  ---------------------------------------
94            running <= not running;        -- on/off umschalten wenn btn gedrueckt
95            timer_on <= not timer_on;       ---------------------------------------
96          end if;
97
98          if(running = '1') then         -- Wenn die Uhr laeuft...
99            if(peak = '1') then         -- ...und der Zaehler einen Peak ausgibt...
100             if(ms = to_unsigned(9, 4)) then     ----------------------------------------
101               if(sec2 = to_unsigned(9, 4)) then   --
102                 if(sec1 = to_unsigned(5, 4)) then  --
103                   if(min = to_unsigned(9, 4)) then --
104                     min <= (others => '0');   --
105                     sec1 <= (others => '0');  --
106                     sec2 <= (others => '0');  --
107                     ms <= (others => '0');   --
108                   else        --
109                     min <= min + 1;     -- ...erhoehe die aktuelle Zeit
110                     sec1 <= (others => '0');  -- um eine Zehntelsekunde (mod 10 Minuten)
111                     sec2 <= (others => '0');  --
112                     ms <= (others => '0');   --
113                   end if;        --
```

```
              else         --
                sec1 <= sec1 + 1;     --
                sec2 <= (others => '0');   --
                ms <= (others => '0');    --
              end if;         --
            else          --
              sec2 <= sec2 + 1;       --
              ms <= (others => '0');     --
            end if;          --
          else           --
            ms <= ms + 1;        --
          end if;           --
        end if;            --
      end if;             ----------------------------------------

    end if;
  end if;
end process;

seg4 <= segMin and "11111110"; -- Punkt der 7-Segment Anzeige aktivieren
seg3 <= segSec1;
seg2 <= segSec2 and "11111110"; -- Punkt der 7-Segment Anzeige aktivieren
seg1 <= segMS;

end uhr;
```

Listing 1.12: VHDL-Code Decoder.vhdl

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.numeric_std.all;

entity Decoder is

  port (
    clk : in std_logic;
    code : in std_logic_vector(3 downto 0);
    decoded : out std_logic_vector(7 downto 0));
end Decoder;

architecture decoder1 of Decoder is

  signal decoded_out : std_logic_vector(7 downto 0) := (others => '0'); -- Signal, dass an den Ausgang
      "decoded" angelegt wird

  begin

  ----------------------------------------------------
  -- Dekodieren eines 4bit breiten Wortes in ein 8bit
  -- breites Wort fuer die 7 Segment Anzeige
  ----------------------------------------------------
  process (clk)
  begin
    if rising_edge(clk) then
      case code is
         when "0000" => decoded_out <= "00000011";  -- 0
         when "0001" => decoded_out <= "10011111"; -- 1
         when "0010" => decoded_out <= "00100101";  -- 2
         when "0011" => decoded_out <= "00001101";  -- 3
         when "0100" => decoded_out <= "10011001";  -- 4
         when "0101" => decoded_out <= "01001001";  -- 5
         when "0110" => decoded_out <= "01000001"; -- 6
         when "0111" => decoded_out <= "00011111"; -- 7
         when "1000" => decoded_out <= "00000001"; -- 8
         when "1001" => decoded_out <= "00001001"; -- 9
```

```
38          when others => decoded_out <= "11111111"; -- error
39        end case;
40      end if;
41    end process;
42
43    decoded <= decoded_out;
44
45  end decoder1;
```

Listing 1.13: VHDL-Code EntprellAutomat.vhdl

```
1   library ieee;
2   use ieee.std_logic_1164.all;
3   use ieee.numeric_std.all;
4
5
6   entity EntprellAutomat is
7
8     port (
9       clk : in std_logic;
10      btn : in std_logic;
11      btnout : out std_logic
12      );
13
14  end EntprellAutomat;
15
16  architecture prell of EntprellAutomat is
17
18    component Zaehler
19      port (
20        clk : in std_logic;
21        zaehler_time : in unsigned(31 downto 0);
22        zaehler_on : in std_logic;
23        peak_out : out std_logic
24      );
25    end component;
26
27    type zustaende is (idle, count); -- 2 Zustaende, idle = button betaetigen moeglich, count = 3ms warten
          (bis 150.000 hochzaehlen bei 50Mhz)
28    attribute enum_encoding : string;
29    attribute enum_encoding of zustaende : type is "1 0";
30    signal z_alt, z_neu : zustaende := idle; -- alter und neuer Zustand des Automaten
31    signal btn_s, btn_output : std_logic := '0'; -- Synchronisiertes (Eingangs)Buttonsignal und
          Ausgangssignal des Automaten
32    signal btn_old : std_logic := '0'; -- Speichern des vorherigen "btn_s" Pegels
33    signal btn2 : std_logic := '1'; -- Synchronisierungssignal fuer den Button
34    signal timer_on : std_logic := '0'; -- Steuerung des externen Zaehler Moduls
35    signal peak : std_logic := '0'; -- Ausgabe des externen Zaehler Moduls
36
37  begin
38
39    -- enthaltener Zaehler, der (falls aktiviert) alle 3ms fuer einen Takt eine eins an das Signal "peak"
          ausgibt
40    timer : Zaehler PORT MAP (clk => clk, zaehler_time => to_unsigned(150000,32), zaehler_on => timer_on,
          peak_out => peak);
41
42    -- Synchronisieren des Eingabesignals (Button) und Speichern des alten "btn_s" Pegels
43    process(clk)
44    begin
45      if rising_edge(clk) then
46        btn2 <= btn;
47        btn_old <= btn_s;
48        btn_s <= not btn2;
49      end if;
50    end process;
51
```

```
52    -- Uebernehmen und berechnen des neuen Zustandes
53    process(clk)
54    begin
55      if rising_edge(clk) then
56        z_alt <= z_neu;
57        case z_alt is
58          when idle =>  if(btn_s /= btn_old) then  -- Wechseln in "count" und starten des Zaehlers, sobald
                sich btn_s aendert
59                  btn_output <= btn_s;
60                  z_neu <= count;
61                  timer_on <= '1';
62                end if;
63          when count => if(peak = '1') then    -- Wechseln zurueck in "idle", sobald der Zaehler eine eins an
                "peak" anlegt (3ms vergangen)
64                  z_neu <= idle;
65                  timer_on <= '0';
66                end if;
67        end case;
68      end if;
69    end process;
70
71    btnout <= btn_output;
72  end prell;
```

Listing 1.14: VHDL-Code Zaehler.vhdl

```
1   library ieee;
2   use ieee.std_logic_1164.all;
3   use ieee.numeric_std.all;
4
5
6   entity Zaehler is
7
8     port (
9       clk : in std_logic;
10      zaehler_time : in unsigned(31 downto 0); -- steuert wieviele Takte der Zaehler zaehlen soll, bis er
            einen peak ausgeben soll. (Zaehler zaehlt mod zaehler_time)
11      zaehler_on : in std_logic; -- Steuersignal, bei 1 laeuft der Zaehler, bei 0 wird der Zaehler gestoppt
            und zurueckgesetzt
12      peak_out : out std_logic -- gibt fuer einen Takt eine eins aus, sobald der durch "zaehler_time"
            angelegte Wert erreicht ist
13      );
14
15  end Zaehler;
16
17  architecture timer of Zaehler is
18
19    signal counter : unsigned(31 downto 0) := (others => '0'); -- Zaehlsignal
20    signal peak : std_logic := '0'; -- Signal, dass an den Ausgang "peak_out" gegeben wird
21
22  begin
23
24    --------------------------------------------------------------------------------
25    -- Liegt "zaehler_on" auf eins, wird das Signal "counter" inkrementiert.
26    -- Wird der durch "zaehler_time" angegebene Maximalwert erreicht wird fuer einen Takt
27    -- eine 1 an das Signal "peak" ausgegeben und "counter" auf 0 zurueckgesetzt
28    -- Liegt "zaehler_on" auf null, wird "counter" auf 0 gesetzt und nicht hochgezaehlt.
29    --------------------------------------------------------------------------------
30    process(clk)
31    begin
32      if rising_edge(clk) then
33        peak <= '0';
34        if(zaehler_on = '1') then
35          if(counter = zaehler_time - 1) then
36            counter <= (others => '0');
37            peak <= '1';
```

```
38          else
39            counter <= counter + 1;
40          end if;
41        else
42          counter <= (others => '0');
43        end if;
44      end if;
45    end process;
46
47    peak_out <= peak;
48
49  end timer;
```

2 Entwurf eingebetteter Systeme Praktikum