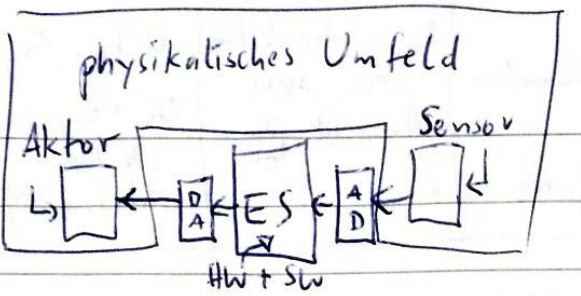
ES I



1. ES

Charakterisierung: ES ≙ Teil eines größeren Systems mit Software auf einen µC, dessen primäre Aufgabe nicht rechenorientiert ist. Er dient zur Erzeugung physikalischer Stellgrößen im Umfeld des Anwenders. Sie werden für spezielle Anwendungsfälle gebaut und es kann dabei personalisierte Hardware zum Einsatz kommen

- kann ein Echtzeitsystem sein

ASIC Application Specific Intigrated Circuit
SoC System on Chip

Anforderungen an ein ES:

- geringe Größe / Gewicht / Kosten / Leistungsaufnahme / Wärmeentwicklung } Mobilität
- umfangreiche Funktionalität } Komfort
- hohe Performance (evtl. Real-Time) } Komfort
- ansprechendes Design
- einfache / intuitive Bedienung
- Robustheit gegen Umwelt / Umfeld
- kurze Time-To-Market

Bedeutung von ES: heute nahezu überall vertreten (IoT)
Gefahr besteht im Versagen von Echtzeitschranken (falls welche)
Hardware & Software muss dabei gut aufeinander abgestimmt sein
⇒ Sicherheit!

Transformierende ES	Interaktive ES	Reaktive ES
A → [ES] → B .h264	Interaktion direkt mit dem User 	Echtzeitanforderung Harte / weiche Zeitschranken

x86 Spielkonsolen Linux	Eigenentwickelte CPUs ARM Android	Modem / Digitalkameras MC + Eigenentwickelte HW meist kein OS	Tastatur, Waschmaschine µC + DSP Kein OS, Assembler
komplexe Aufgabe		spezielle Aufgabe	

Entwurf: HW/SW Partitionierung ↔ braucht Rechnergestützte Entwurfsmethoden & automatisierte Exploration des Entwurfraums (EDA)

- soll günstig & schnell sein
- für Co-Design gibt es wenige Tools
- Entwickler legt HW/SW Teile selber fest und entwickelt danach beides mit getrennten, verschiedenen Tools

Trade-offs: Testbarkeit / Geschwindigkeit / TDP / Temperatur / Chipkosten / Chip-Größe / Chip-Yield / Verpackungskosten / Zuverlässigkeit / Anzahl I-O Pins

1. Analyse der Requirements & Randbedingungen
2. Spezifikation
3. HW/SW Partitionierung ⟵ iterieren (teuer), falls Anpassungen oder Optimierung nötig
4. HW & SW Beschreibung & Entwicklung
5. HW & SW Co-Simulation
6. IC & Bewertung nach Test

Zur Vermeidung einer Iteration eignet sich der Überentwurf

Eierlegende Wollmilch-Sau:

Tool mit: möglichst früher Kostenabschätzung
Simulation für HW/SW-Partitionierung
Formale Verifikation möglich
Kontrolle über Echtzeitanforderungen
Modellierung der physikalischen Systemumgebung
Große Bibliothek für Standardkomponenten

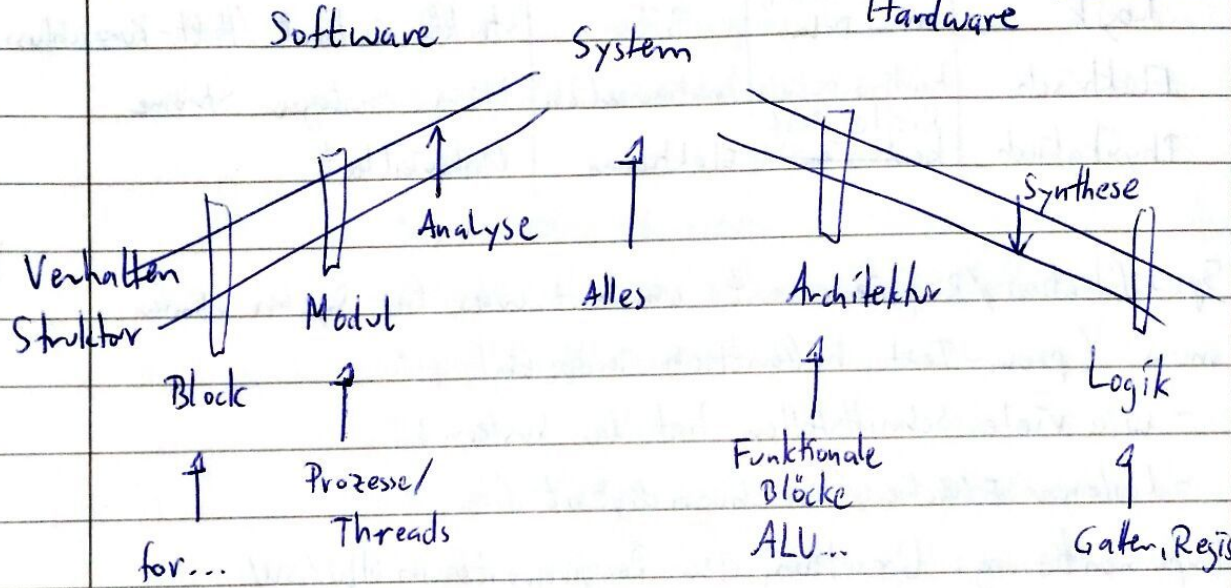
Was es gibt: SW → HW (HLS) C to Silicon
Erkennung von Hotspots, die effizienter in HW gebaut werden können

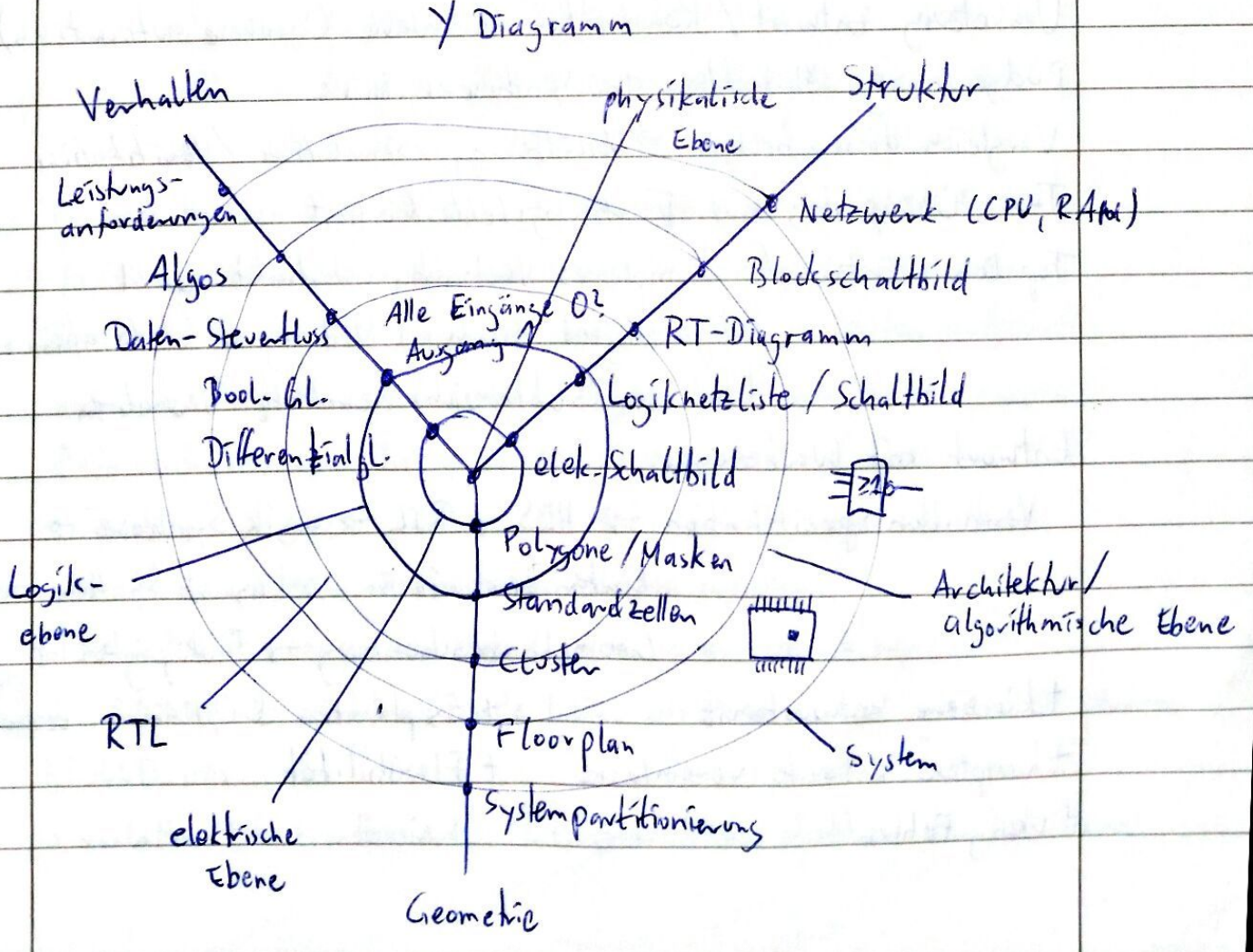
ESI

Hohe Komplexität der ES wird durch Hierarchie und Abstraktion bewältigt

Abstraktionsebenen:

2. ENTWURF





Mit den Ebenen abstrahiert sich auch die Sicht auf Zeit/Daten/Test...

	Zeit	Daten	Test
System	Kausalität	abstrakte Daten	globale Teststrategie
Architektur	abstrakter Takt	", nat. Zahlen	funktionaler Test
RTL	Taktzyklen	Bitvektoren	Testmethoden für Module
Logik	kontinuierlich, zero Delay	Bits	strukturorientiert / Haft-Kurzschlussfehler
Elektrisch	kontinuierlich	kontinuierlich	Messen analogen Ströme
Physikalisch	Reale Zeit ~~kontinuierlich~~	Elektronen	Materialtest

Spezifikation / Requirement umfasst, was das System können muss (prosa-Text, hoffentlich eindeutig)
- wie viele Schnittstellen hat das System?
- Kosten / Fläche / Geschwindigkeit / ...

Implementierung: Umsetzung der Requirements in HW/SW
Umsetzung: Entwurf / Konstruktion des Systems (Synthese automatisch)
Rückgewinnung: Abstraktion der Syntheseergebnisse
Vergleich dieser beiden ⇒ Validierung / Simulation / Verifikation
Test: Überprüfung auf Fertigungsfehlerfreiheit
Top-Down Entwurf: komplexes Verhalten, abstrakt wird abgearbeitet durch Aufteilung in (bald atomare) konkrete, strukturreiche, einfache Verhalten
Entwurf mit Werkzeugen:
Verhaltensspezifikation → HLS → RTL → Logik Synthese → Gater Beschreibung → Layout Synthese → Geometriebeschreibung → Fertigung → Chip

+ kürzere Entwurfszeit
+ komplexe Entwürfe vereinfacht
+ wenig Fehler
+ Exploration & Optimierung effizient ~~einfach~~
+ Flexibilität

ES I

3. VHDL

HW-Beschreibungssprache VHDL für konfigurierbare HW

Was es kann: - Strukturierte Prozedurkonstrukte
- Datentypen: Char, String, Real, Int, Physics, Bezeichner
- Prozeduraufrufe
- Variablen mit Scopus
- Parallelität und Nebenläufigkeit
- Zeit
- Schnittstellen / Entities
- Struct / Process / Behavioral / Dataflow
- Simulation & Synthese
- Operatoren
- If/then/else und case für Verzweigungen
- loop
- Funktionen / Prozeduren
- Generics (auch am Port einer Entity
- Packages & Libraries

Prozesse besitzen eine Sensitivitätsliste, die sie aktiviert.

Top-Down oder Bottom-Up durch struct ermöglicht

Prozesse in sich sind sequentiell: Schaltung real in Stufen aufgeteilt, mehrere Prozesse sind selbständige Schaltungen, daher parallel.

Zeitverhalten per Ereignisliste, sie verzeichnet, wann ein Signal seinen Wert ändert (1→1 möglich) und eventuell damit verbundene Prozesse angestoßen werden.

TRANSPORT: copy des Signals um n ns verschoben

INERTIAL: alle stabilen Signale länger als n ns werden um n ns verschoben.

REJECT: alle " " " " " " " m ns "

Simulation ist eventbasiert 1. Elaboration: Aufbau der Schaltung

2. Initialisation: Variablen setzen, alle Prozesse starten
3. Ausführung: Delta-Zyklen bis zu einem stabilen Zustand.
   3.1. Prozess evaluierung bis END oder WAIT
   3.2. Signale zuweisen, Prozesse anschmeißen (Delta zyklus)
4. Loop 3., 3.1, 3.2 & und erhöhe Simulationszeit bis zum nächsten Eintrag in der Ereignisliste

Testbench für Signalsetzung und Überwachung

4. SYNTHESE

Verhaltensspezifikation → RTL
Kausal → Takt Abstrakte Daten → Bits

High-Level-Synthese: schnell, günstig, einfach, wenn gut entwickt, dann fehlerfrei, optimiert

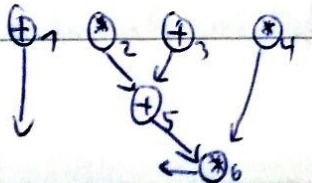
Optimierungen: 1. Konstanten propagieren, mehrfach berechnete Ausdrücke werden nur ein mal berechnet und dann öfter wiederverwendet.
2. Inline-Expandierung: Funktionsaufrufe werden durch ihren Rumpf ersetzt
3. Schleifeninvarianten herausziehen

Zwischenformate: 1. Petri-Netze: Kontrollfluss explizit, egal wie komplex, Datenfluss bleibt abstrakt (höchstens als Datentransport durch die Knoten), geeignet für Simulation, Darstellung, Beweise für Erreichbarkeit u.Ä.
2. Datenflussgraph: Knoten ≙ Rechenoperationen
   Kanten ≙ Datentransport
3. Flussgraph: Operatorenknoten mit Operation und Steueranweisung
   Variablenknoten mit Name der Variablen
   Datenflusskante verbindet einen Variablen-

ESI

Knoten mit einem Steuerflussknoten Operator
Steuerflusskante verbindet zwei ~~Steuerflussknoten~~
3. RTL-Struktur: Schema, wie welche HW Bausteine letzten Endes ~~[illegible]~~ verbunden sind und wie welche Daten fließen.
Datenfluss explizit ausgedrückt, Kontrollfluss deutlich, relativ anschaulich, formal schwer darzustellen, Nebenläufigkeit, Zeitverhalten und Pipelining nicht ausdrückbar.

Scheduling & Allokation (Syntheseergebnis: Datenpfad & Steuerpfad)
Ziel: Erzeugen einer Netzliste auf RTL
~~[illegible]~~
Scheduling: Den Operationen werden unter Berücksichtigung ihrer Datenabhängigkeiten eine Reihenfolge zugeordnet.
Allokation: Erzeugung der nötigen Recheneinheiten, um das Schedule umsetzen zu können.
Bindung: Variablen & Operationen an die ~~Allokierten~~ Bausteine binden. Dies wirkt auf die Größe der Multiplexer am Eingang der Bausteine.
Finde möglichst kleine und schnelle Schaltung. Problem: Allokation und Schedule wirken aufeinander, für ein gegebenes Schedule ist die Allokation ablesbar. Umgekehrt: Für eine gegebene Allokation kann ein Schedule unmöglich werden. Exakte Lösung NP-HART !!
Verfahren statisch (da HW): konstruktiv oder iterativ
1. ASAP: exakt lösbar in poly-Zeit
mit Ressourcenbeschränkung: erzeuge Halbordnung:



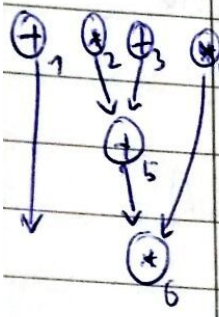
$\{1,2,3,4\} < \{5\} < \{6\}$

Problem: 1 blockiert ALU und ist unkritisch ⇒

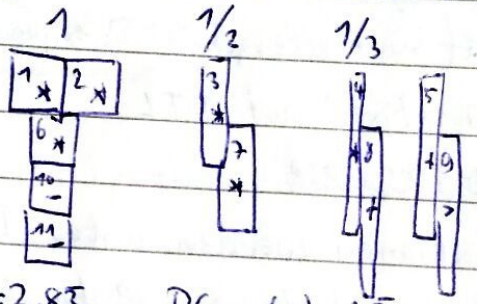
2. ALAP

3. List Scheduling: Erzeuge Prioritäten für Operationen abhängig von ihrer Pfadlänge im DFG: $\{2,3\} < \{4,5\} < \{1,6\}$

4. Force-directed Scheduling: Maß für Auslastung von HW-Bausteinen. Konstruktiver Algorithmus, entscheidet Allokation & Schedule gleichzeitig



Erzeuge Wahrscheinlichkeiten aus ALAP / ASAP Schedule:



Distribution-Graph: Zahl die angibt, wie sehr eine HW-Komponente in einem Kontrollschritt ausgelastet ist.

$DG_{mul}(0) = 2,8\overline{3}$ $DG_{alu}(0) = 0,\overline{3}$

$DG_{mul}(1) = 2,\overline{33}$ $DG_{alu}(1) = 1$

$DG_{mul}(2) = 0,8\overline{3}$ $DG_{alu}(2) = 2$

$DG_{mul}(3) = 0$ $DG_{alu}(3) = 1,\overline{6}$

Berechne für alle möglichen festlegenden Zuweisungen die Total-Force. Sie zieht mit in Betracht, wie Kräfte wirken, wenn implizit dadurch Entscheidungen für die Vorgänger- und Nachfolgeoperationen gefällt werden. Die niedrigste Total-Force führt zur Bindung einer Operation an einen Kontrollschritt und auch eventuell zur Allokation dafür notwendiger HW-Bausteine.

5. → Force-Directed Listscheduling (wähle größte Kräfte)

Berücksichtigung von Bussen, diese beschränken die maximale Anzahl an parallelen Zuweisungen in einem Kontrollschritt (Transfer DG)

Registerminimierung durch Reduzierung von Variablenlebensdauern

$$\text{Storage-DG}(i) = \frac{\text{AVG Life-Overlap}}{\text{MaxLife-Overlap}}$$

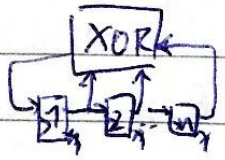
Zeitbedingungen berücksichtigen: Wann kommt (wie viel) Input?
Wann darf geschrieben werden?

Pipelining von Schleifenrümpfen
Zusätzliche Allokationsinformationen: Flächenabschätzungen für Recheneinheiten
Verkettete Operationen in einem Kontrollschritt
Registerallokation: Greedy Ansatz nicht gut, besser:
Erstelle Verträglichkeitsgraph: Kante zwischen 2 Knoten (Variablen) ≙ können das selbe Register zeitlich teilen → erzeuge Cliquen
Alternative: Konfliktgraph + Färbung // NP-Vollständig x.x
Intervallgraphen lassen sich in linearer Zeit färben
Nun müssen Multiplexer vor die Recheneinheiten geschaltet werden. Sie erhalten Steuersignale von einem Steuerwerk (Zustandsautomat)
↑ Path-based-Scheduling: Fasse möglichst viele Knoten des Kontrollflussgraphen zusammen
Neuer Zustand ⇔ Schleifenbeginn v mehrfache Nutzung einer Komponente v mehrfache Datenzuweisung an eine Komponente:
Ziel: Finde cut-Intervall (das erzeugt dann 2 Zustände) für alle möglichen Pfade
⊕ Gute Ergebnisse bei Kontrollflüssen-Problemen
⊖ keine Beachtung gegenüber Bussen, Registern, Schnittstellen, Schleifenausrollen, Pipelining, Instruktionsumordnung

V Logik-Synthese

Zustandsminimierung ⇔ Zwei Zustände S1, S2 sind äquivalent, wenn für alle möglichen Eingaben für S1 und S2 die

gleichen Ausgaben erzeugt werden und die Folgezustände äquivalent sind
Bei – in der Ausgabe kann – so gesetzt werden, dass Zustände kompatibel werden
Finde maximale Kompatibilitätsklasse und fasse kompatible Zustände gemäß dem zusammen
Zustandskodierung in HW: binär #bits = $\lceil \log_2 \#\text{Zustände} \rceil$
#möglicher Kodierungen: $\binom{2^{\#bits}}{\#Zustände} \hat{=} O(n!)$
Umgesetzt: Addierer
Grey-Code: Immer maximal 1 bit ändert sich $r = \lceil \log_2(n) \rceil$
1-hot encoding: Immer 1 bit aktiv für jeden Zustand $r = n$
LFSR: Linear Feedback shift Register



kann $2^n - 1$ Zustände mit n bit kodieren. 0 mit extra Aufwand möglich

Logikminimierung
→ abhängig von Zieltechnologie (PAL, PLA, ROM, ...)
Minimierung kann sein: Reduzieren der #Transistoren, Fläche, #Standardzellen, Verdrahtungsfläche, Blockzahl (FPGA)...
Verfahren (Espresso) EXPAND, REDUCE, ESSENTIAL, IRREDUNDANT
Optimierung mehrstufiger Logik kann sein: kleinere Fläche bei Einhaltung der Zeitbedingungen (oder umgekehrt)
Tranformationen sind: Elimination zerstöre Knoten, wenn zu selten gebraucht $r = a+b$ $p = r+c \Rightarrow p = a+b+c$
Dekomposition wenn Ausdruck zu groß
Extraktion erzeuge Knoten für gemeinsamer Unterausdrücke
Substitution: Verwendung vorhandener Teile
Vereinfachung bei Redundanz
Frequenzoptimierung: Upsizing: größere Transistoren
Load isolation: Buffer / Treiber einfügen (exp. Fläche wächst!)
Load splitting: Gatter verdoppeln und Output verteilen

6. Zieltechnologien

Meet-in-the-Middle: Synthese erzeugt Gatterbeschreibung, diese stammen aus Zellbibliotheken:

Transistoren fertigungen: pMos / nMos ⇒ cMos (FET) } Silizium

IIL / ECL / TTL (bipolar)

BFL / SDFL / DCFL (FET) } Gallium-Arsenid

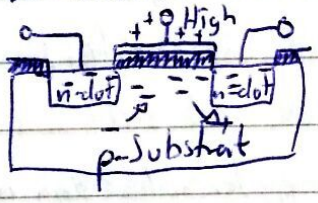
CMOS in der Digitaltechnik: schalte VDD oder GND durch je nach Eingangsbelegungen. Außerhalb der Umschaltzyklen besteht keine (Widerstandslose) Verbindung zwischen VDD & GND

⇒ keine (kaum) statische Verlustleistung

n-Dotierung: Phosphor-Fremdatom im Siliziumgitter

p-Dotierung: Aluminium-Fremdatom im Siliziumgitter

n-Kanal-Transistor



Layout: Genaue Zeichnung der physischen, analogen Komponenten

Probleme: viel Wissen über breite von Teilen, Abstände, Möglichkeiten DRC notwendig um Yield zu maximieren und Kosten zu senken

Layouts sind Herstellerspezifisch ... ⇒ Abstraktionsebene gesucht!

Vollkundenentwurf: alles Individuell & vom Kunden gebastelt: hoch spezifisch, schweineteuer. Er darf sich aus Bibliotheken bedienen ⇒ schneller, weniger Fehleranfälligkeit, automatisiert, Anpassung an neue Technologien einfach, dafür aber wenig für eine Anwendung optimiert

Makro-/Standardzellen / Gate Array / Sea of Gate sparen viel Aufwand durch Vorfertigung der Layouts, Verkabelung notwendig

Standardzellen ≙ Teilschaltung, die vom Kunden plaziert werden kann (NAND, AND, XOR, VA, HA, Vergleicher, DFF, Tristate) mit Standardisierten VDD und GND Anschlüssen für Back-to-Back oder dual-entry Verdrahtung. Doppelzelle möglich. Oft höher optimiert als von Hand. Über diese Zellen ist viel bekannt: Logisches Verhalten (Funktion), Größe, Anschlüsse, temporales Verhalten, elektrisches Verhalten, Layout, Simulationsmodelle

Makrozellen ≙ Standardzelle ohne feste Abmessung, dadurch beliebig komplex (ALU, PLA, ROM, CPU...)
unterbricht die regelmäßige VDD, GND Leitungsstruktur. Dafür gibt es feed-through. Gemeinsame Optimierung gegen Platzverschwendung

Gate-Array ≙ Vorgefertigte Transistoren auf einem Chip, die der Kunde dann verdrahten kann. Strukturen: einfache Streifen, doppelte Streifen, Inseln

Sea-of-Gates ≙ Gate-Array ohne Verdrahtungskanäle. Man nutzt ungenutzte Transistoren als Leiter

Gate-Array vs.	Standardzelle
- feste Chipfläche	+ variabel
- feste Komplexität	+ variabel
- feste Verdrahtungskanäle	+ variabel
+ 4 Prozessschritte	- 9-11 Prozessschritte
+ schnelle Verfügbarkeit von Mustern	+ geringe Kosten bei hoher Stückzahl

Speicher: Horizontal: Word-Lines
Vertikal: Bit-Lines

ROM: Matrix aus fest verbauten und fest verdrahteten NMOS Transistoren, die eine schwache 1 stark auf 0 ziehen

Haben zwar an jeder Kreuzung einen Nmos T., der muss aber nicht verdrahtet sein (→ wie SoG, hohe Integrationsdichte)

PROM, wie ROM, allerdings können 1en geschrieben werden, indem die Verbindungen zu den Nmos T. durchgebrannt werden.

(E)EPROM / Flash - Speicher

Floating Gate auf dem Kanal unter dem herkömmlichen Gate (floating ≙ elektrisch isoliert).

Diese Spannung im Floating Gate kann die Threshold-Voltage des Transistors verschieben, sodass die Gatespannung nicht mehr für einen normalen Betrieb ausreicht. UV-Licht (EPROM) oder eine echt hohe Spannung (18 V+) (EEPROM) kann das wieder rückgängig machen.

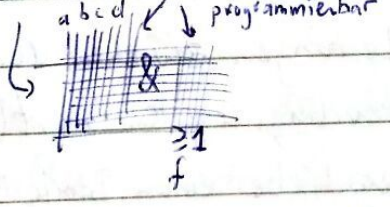
SRAM: Flipflops + kein Auffrischen + schnell - Stromverbrauch - Platz

DRAM: Kapazität und NMOS + sehr klein - muss aufgefrischt werden

Eine Zelle hängt an zwei Bitleitungen, davon gibt es doppelt so viele. Alle Leitungen werden aufgeladen beim Lesen, damit der minimale Strom, der eventuell aus einer Kapazität fließt (oder in sie verschwindet) eine Referenz hat.

Ein Leseverstärker (Doppelinverter) kann diese Differenz amplifizieren, wovon sie dann auch gelesen wird

PLD = { PAL, PLA }



FPGAs: + kurze Entwurfszeiten + anwenderkonfigurierbar + Standardprodukt + hohe Komplexität + flexible Struktur + gute Entwurfssoftware

Kriterien bei der Wahl einer Zieltechnologie:
- Entwurfs- und Fertigungszeiten
- Fertigungsaufwand
- Kosten vs. Stückzahl
- Chipkomplexität (Gatter, I/O)
- Taktrate
- Leistungsaufnahme / Energieverbrauch
- Benötigte Flexibilität und Anpassbarkeit
- Zuverlässigkeit
- Verfügbarkeit
- ...

7 Layoutsynthese

Platzierung der Komponenten auf dem Chip NP-Vollständig
Verfahren: Mead & Conway ↦ Top-Down durch die Hierarchy der Blöcke mit Heuristiken: wenige Überschneidungen, Ø kurze Leitungen, kürzeste Längste Leitung
Flying-Wire: Kabel ↦ Gummibänder mit Kräften, problematische Bänder erhalten große Kraft
Fiduccia-Mattheyses ↦ Divide & Conquer: Chip halbieren und Komponenten der Größe nach balanziert aufteilen
Wünsche: möglichst wenige Leitungen zwischen den Hälften
Random: schnell & schlecht, allerdings gute Startlösung
Force-directed: Random → verschiebe einen Block ins Zentrum derer, mit denen er verbunden ist → Austausch bei Kollision → markiere als unverschiebbar → alle markiert? loop

Simulated Annealing : benötigt Bewertungsfunktion

Verdrahtung: Implizient Abstand der einzelnen Komponenten zueinander → Chipgröße. Große Module bieten Feed-through

Eigene Ebenen für Taktnetz, VDD, GND, Busse } Entflechten möglich bei großer Anzahl an Metallebenen

Simple-Router nutzt Labyrinth-Algorithmus. Naiv, schnell, automatisch

Hierarchischer-Router → Divide & Conquer : unterteile Chip und erzeuge an den Grenzen Kanäle, die mit einer Kostenfunktion an Leitungen vergeben werden, wenn sie da durch müssen / die Grenze schneiden. Klein-ganug → nutze anderen, Greedy, Algo

Pathfinder : Iterativ : zuerst ignoriere Kollisionen und gebe jeder Leitung den kürzesten Kanal. Überbelegte Kanäle werden pro Iteration teuer, die mit der Priorität einer Leitung verglichen wird. (Diese kann für Timing angepasst werden)

Chipherstellung : Wafer = Einkristall aus Silizium, wird in 0,5mm - 1mm dicke scheiben geschnitten

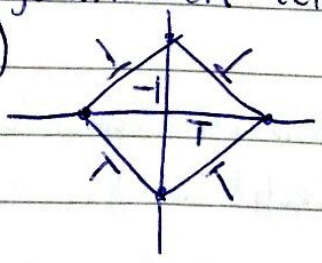
Silizium p-dotiert bei der Herstellung

1. Freilegen der zu dotierenden Flächen und Diffusions-leiterbahnen durch ätzen der Isolationsschicht an den nicht geschützten Stellen
2. Freilegen des Kanals des Verarmungstransistors
3. Aufbringen des Gateoxids (echt kurz)
4. Aufbringen von Poly-Silizium
5. n^+ dotieren an Drain & Source
6. SiO_2 aufbringen, Kontaktlöcher freilegen und Al_2

aufdampfen
7. Oxid aufdampfen und Kontaktlöcher für die oberen Metallebenen ätzen
Multilayer herstellung (Metallebenen)
erstelle Ebenen (doppelseitige) separat, presse sie aufeinander

8. FPGAs

Vorteile programmierbarer Logik + Vorteile Gate Arrays
+ kurze Entwurfszeiten + hohe Komplexität
+ anwenderprogrammierbar + flexibel
+ Standardprodukt + gute Software für Entwurf
→ FPGA ←
Look-Up-table (LUT) realisiert funktionen. Heute gängig mit 6 Eingängen
Mehrere davon, von Registern gehalt, bilden einen Configurable Logic Block (CLB)
Mehrere CLBs bilden einen Slice
Homogene Struktur von CLBs und Programmable Switch Matrices (PSMs)
PSM: Kreuzende Schaltleitungen können mit einem Konfigurationsbit leitend gemacht werden (auch ganze Busse)



n-Transistor

Logikebene, realisiert durch die LUTs und PSMs
Konfigurationsebene, liegt auf der Logikebene, bestimmt

diese aber in ihrer Funktion
Die Konfiguration muss bei Inbetriebnahme neu eingelesen werden (Durch Benutzer oder aus EEPROM/SRAM)
LUTs können auch Shift-Register realisieren
Dedizierte DSP-Blöcke für Multiplikationen o.Ä.
MAC / Akkumulator / ...
Speicherblöcke in LUTs (BRAM) beliebig verteilbar und konfigurierbar
Clock-Netz ≠ Daten-Netz
skew: Zeit, die zwischen zwei (+) Events vergeht, die gleichzeitig hätten eintreten sollen, aber durch Propagationsverzögerungen verzögert wurden.
Jitter: Zeit, die ein kontrolliertes Signal verschoben zu seiner Spezifikation vergeht
Einsatzgebiete: Glue Logic (Protokoll übersetzen ...)
Prototyping (Chip vor Herstellung testen)
Kleinserien (wenn sich Entwurf eines ASIC nicht lohnt)
Vorserienprodukte (um TTM zu kürzen)
Lokales Place and Route: "Packen"
Lege fest, welche LUTs in welche CLBs kommen
Basiert auf Analyse der Netzliste
Nachdem kritische Komponenten untergebracht sind, füllt freie CLBs mit den restlichen LUTs auf
Ziel: einfacheres P&R, schnelle Clocks, wenige CLBs

9. µC & DSP

Programmierbar (mit Spezialisierung)
µC eher zur Steuerung von Geräten (Anzeigen, Tastaturen..)
DSP eher zur Verarbeitung von Datenströmen (Filter, Kodieren, Dekodieren → Multimedia / Telekommunikation)
µC Komponenten: Rechenkern + SRAM + Flash + Peripherie auf einem Chip
Einsatzgebiet: Waschmaschine, Spülmaschine, Armbanduhr, Motorsteuerung, ABS, Tacho, Eisenbahnsteuerung
Peripheriebausteine: SRAM, EEPROM, ..., I/O Ports, DA/AD, PWM, I²C, SPI, (S)UART, CAN, USB, One-Wire, Ethernet, Timer, Zähler-Zeitgeber-Einheit, Watchdog
Serial Peripheral Interconnect
1 Master, n Slaves, n Slave Select Leitungen, synchrone serielle Übertragung, hohe Datenraten (MHz-Bereich)
I²C (Inter Integrated Circuit)
Nur 2 Leitungen SCL, SDA, eher langsam (100-400kHz)
1-Wire: Nur eine Leitung (+Masse) "DQ" für Versorgungsspannung & Daten - Einsatz zur Kommunikation zwischen Komponenten (Akkoüberwachung, Thermometer auslesen..)
U(S)ART Universal Synchronous/Asynchronous Receiver/Transmitter: vollduplex-Übertragung, anynchron oder synchnen. Datenübertragungsbefehle:
TxD, RxD Handshake: TxRDY, RxRDY
Modemsteuerung: CTS, DSR, DTR, RTS
Universal serial Bus (USB) verpolungssicher, Daten über zwei verdrillte Adern (+4 Stück bei USB 3.0)

Da USB beliebt, liegt oft ein Hardwareblock vor, Software auch einfach

Zähler / Zeitgeber: können Events zählen (Interrupt / Clock / ...)
können selber Interrupts auslösen, z.B. beim Überlauf oder erreichen eines bestimmten Zählerstandes
Einsatz: Impulsgeneratoren, Taktgeber, Zeitmessung, Referenz-zeitgeber für Ereigniszähler, ...

PWM (Pulsweitenmodulation) ist ein Rechteck-Signal mit variabler Breite, „quasi-analog". Wird genutzt zum Dimmen von LEDs oder zur Lautstärkeregelung
Verwendet Timer mit einstellbarem Überlauf (Periode T) und einstellbarem Umschaltzeitpunkt (t_1)

Watchdog zählt bis zu einem Wert hoch und löst dann Reset oder Interrupt aus. Er will regelmäßig von einem Programm zurückgesetzt werden.

LED Ansteuerung: n LEDs aber nur m << n Leitungen.
Daten Shiften [Ser] → m → [Par]
Drawback: Updaterate sinkt mit steigender LED Anzahl
Zeile / Spalte Multiplexing mit zeitmultiplexing
Charlieplexing: n bit → n (n-1) LEDs einzeln ansteuerbar mithilfe des Tri-States des µC und des Spannungsabfalls über eine Diode

Entwicklungsumgebung:
Interpreter (oft Basic)
Compiler (C, Pascal, ...)
Assembler (prozessorspezifisch, auch cross-compiler möglich)
Simulator für µC vorhanden, zeige alle Register, ...
In-Circuit-Emulator / Debugger ersetzt zur Entwurfs-

zeit den µC, um Peripherie oder Timing-Probleme aufzuzeigen

Datenformat für die Übertragung der Anwendung in den (EE)PROM:

Binär / Intel-Hex-Format

: 04 0000 00 74 0C 24 05 53

Start Länge Adresse 1. Byte Datenkennung Daten Prüfsumme

DSP gut für Datenverarbeitung, Pipeline, parallel arbeitende Arithmetikeinheiten, schnelle Multiplizierer, MAC, oft nur FP, Havvard-Architektur ⇒ Programmbus, Datenbus oft zwei Datenbusse, Ybus Xbus

Anwendung zur Filterung, Frequenzanalyse ... in Smartphones, Radar, Sonar, Messtechnik, Musikerzeugung, MP3/MP4 Encoder/Decoder, Spracherkennung, Bildverarbeitung

Compiler: C / C++ (Crosscompiler für DSP-spezifischen Assembler

Editor / Debugger / Linker / Simulator / Evaluierer und Analysierer, alles da!

Umsetzung von Finite Impuls Response (Hochpass, Tiefpass, Bandpass) oder rekursiven Filtern Infinite Impuls Response IIR

Hier kommt die MAC Einheit zum Tragen