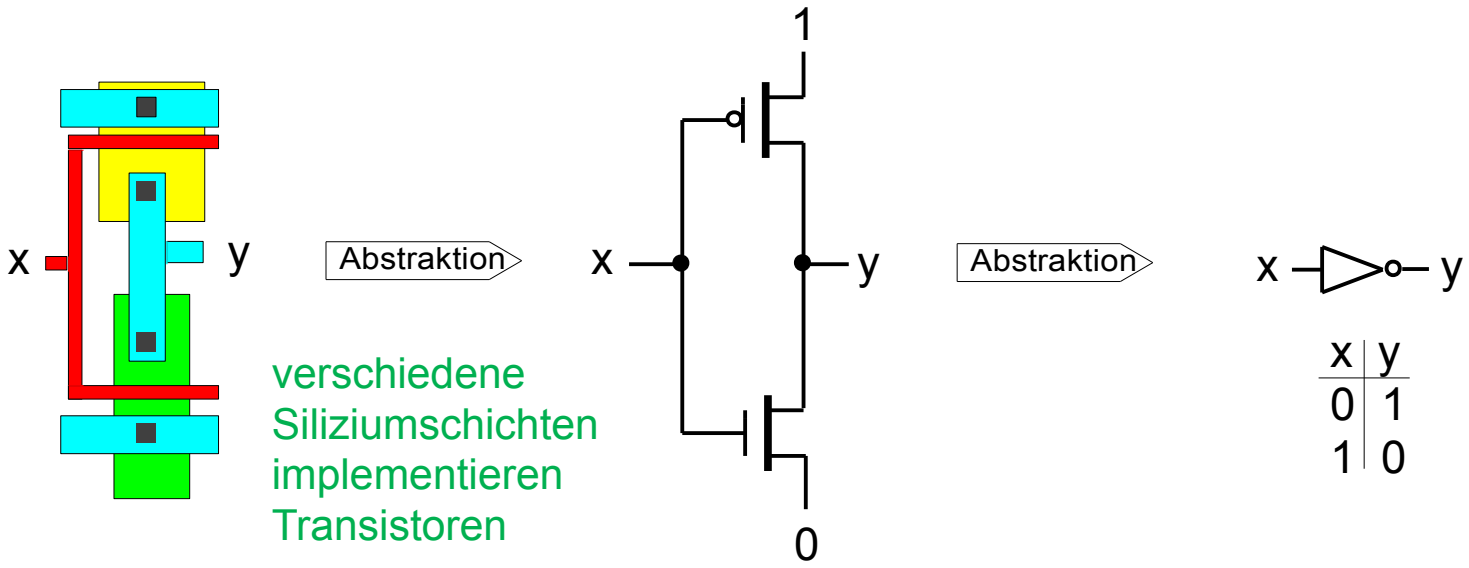
13. Entwurfsautomatisierung

Entwurfsebenen

Dieses Kapitel gibt einen ersten Einblick in den rechnergestützten Entwurf eines mikroelektronischen Systems.

Layoutebene **Transistorebene** **Gatterebene**



Implementierung logischer Funktionen durch MOS-Transistoren

Je nach geplanter Stückzahl und Anwendungsbereich der zu entwerfenden Schaltung wählt man zwischen unterschiedlichen Entwurfsstilen aus.

Standardzellenentwurf

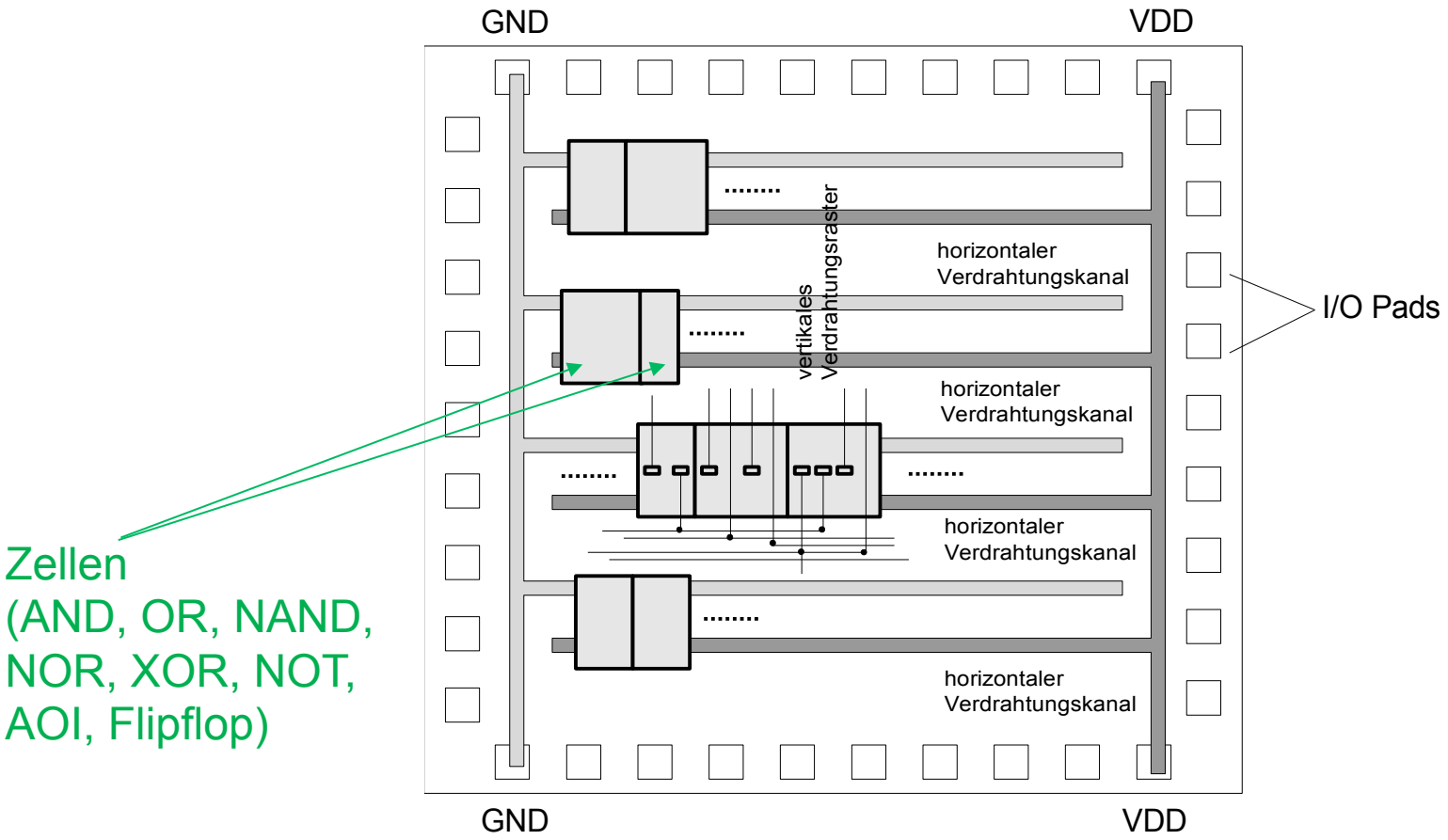
Viele Entwurfsaufgaben können durch Standard-ICs (z.B. Standardbausteine wie Mikrocontroller, Mikroprozessoren), die von zahlreichen Herstellern angeboten werden, gelöst werden.

Oft ist es aber vorteilhaft, für die gegebene Anwendung einen speziellen Chip zu entwickeln. Solche ASICs (application specific integrated circuits) werden mit CAD-Werkzeugen entworfen, die eine Vielzahl von Entwurfsschritten vollautomatisch durchführen.

Ein klassischer Entwurfsstil für ASICs ist der Standardzellenentwurf. Der gesamte Entwurf wird in relativ kleine Einheiten (z.B. Gatter, Flipflops), so genannte "Standardzellen", aufgeteilt, die getrennt entworfen werden können. Der Abstraktionsgrad für die Zellaufteilung ist die logische Ebene (Gatterebene). Die Zellen haben alle die gleiche Höhe, lediglich ihre Breite kann variieren. Die Zellen werden meistens manuell entworfen und in einer Zellbibliothek zusammengefasst.

Für den Standardzellenentwurf stehen ausgereifte CAD-Werkzeuge zur Verfügung.

Standardzellen-Layout



Struktur eines Standardzellenentwurfs

CAD-basierter Entwurf

Hardwarebeschreibungssprachen (VHDL, Verilog) gestatten die Spezifikation des Entwurfs auf höherer Ebene. Synthesewerkzeuge setzen die Spezifikation automatisch in eine detaillierte Beschreibung des physikalischen Entwurfs um.

Beispiel: Addierer

1. Schritt: Abstrakte Spezifikation durch spezielle Beschreibungssprachen für Hardware (VHDL, Verilog)

```
entity adder is
port(  a, b   : in   integer range 0 to 7;
       s      : out  integer range 0 to 7);
end entity adder;

architecture behaviour of adder is
begin
       s <= a + b;
end architecture behaviour;
```

Logiksynthese

2. Schritt: Logiksynthese

- Übersetzung der Spezifikation in unoptimierte Netzliste aus elementaren logischen Grundfunktionen (UND, ODER, NICHT)

  Je nachdem, welcher Abstraktionsgrad für die Spezifikation gewählt wurde, ist dieser Schritt mehr oder weniger aufwändig. In der heutigen industriellen Praxis ist der Abstraktionsgrad noch relativ nahe an der Struktur der entstehenden Netzliste, so dass dieser Abbildungsschritt relativ einfach ist.

  Andernfalls, wenn die Spezifikation das Verhalten des Systems auf hoher Abstraktionsebene beschreibt, sind Techniken der "High-Level"-Synthese erforderlich.

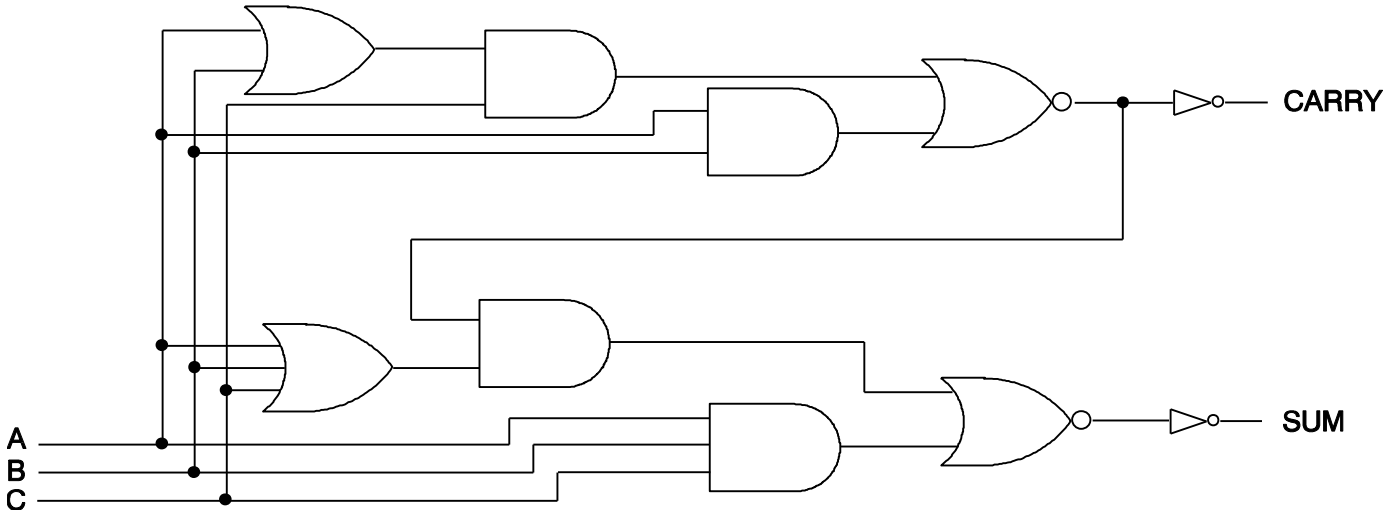
- "Technologieunabhängige" Optimierung der Netzliste

  Information über den konkreten Implementierungsstil, wie beispielsweise die Eigenschaften der Standardzellen einer gegebenen Bibliothek, wird noch nicht benutzt. In der Regel wird nur die Anzahl der Literale in den zugrunde liegenden Booleschen Gleichungen minimiert.

Logiksynthese

- Anpassung der optimierten technologieunabhängigen Darstellung an Implementierungsstil (technology mapping).

  Im Falle des Standardzellenentwurfs: Darstellung der Netzliste durch Elemente der Zellbibliothek, Optimierungen (Fläche, Laufzeit) unter Ausnutzung der physikalischen Eigenschaften der gegebenen Standardzellen (technologieabhängige Optimierung) .

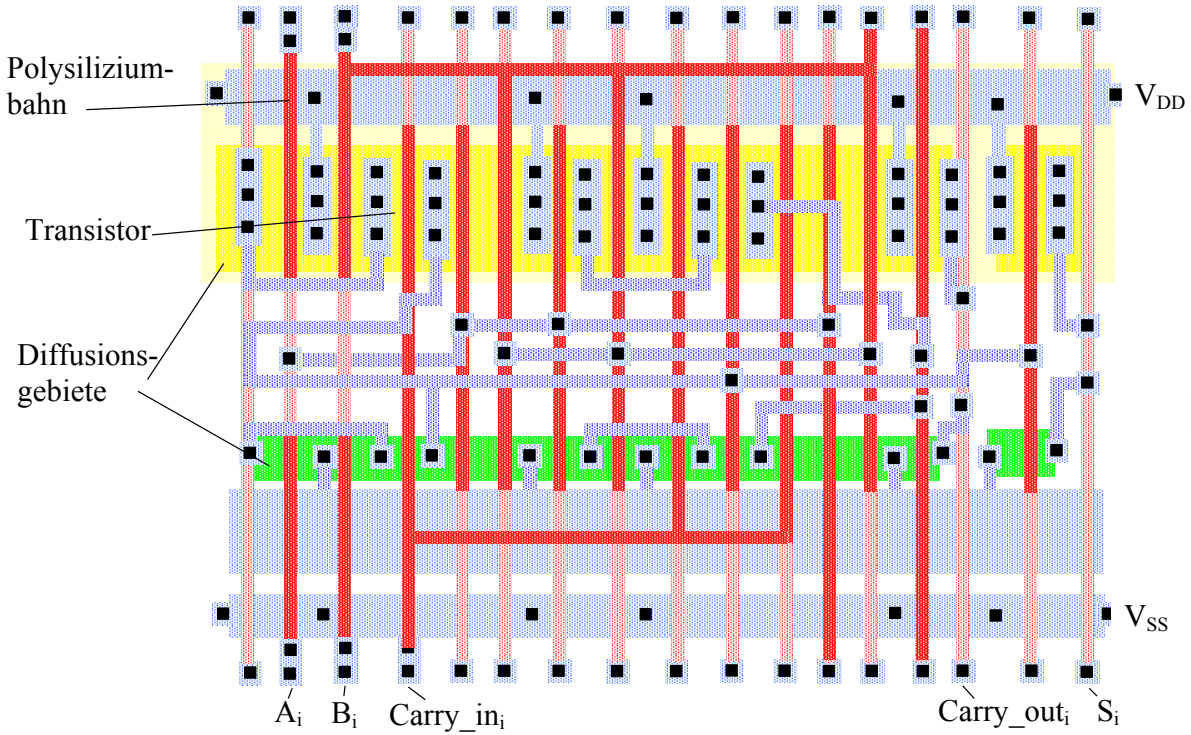


1-bit Addierer auf Gatterebene

Layoutgenerierung

3. Schritt: physikalischer Entwurf (Layoutgenerierung)

- Platzierung der Zellen auf dem Chip
- Verdrahtung der platzierten Zellen
- Dimensionierung der Transistoren zur Optimierung von Fläche und Signallaufzeit, (große Gatter sind schnell, kleine Gatter sind langsam).



1-bit Addierer auf physikalischer Ebene

Simulation

Aus einzelnen kombinatorischen und sequentiellen Schaltungskomponenten werden hochkomplexe Systeme zusammengesetzt. Das Zusammenwirken zwischen den Komponenten ist oft schwer zu durchschauen, so dass „Entwurfsfehler“ (design bugs) niemals auszuschließen sind.

Daher ist die „Verifikation“ der spezifizierten Komponenten und des Gesamtsystems sehr wichtig.

Verifikation durch Simulation:

Erstelle ein Modell des Systems. Lege Stimuli an die Eingänge des System(modell)s an und untersuche das Verhalten des Modells für diese Eingaben.

Simulation auf allen Entwurfsebenen zur Validierung der Korrektheit des Entwurfs.

Formale Verifikation

Simulation macht nur Stichproben ⇒ unvollständig!

"Formale" Verifikation: mathematisch exakt, vollständiger Beweis der Korrektheit, gewinnt industriell stark an Bedeutung.

Verifikation durch formale Verifikation:
Erstelle ein Modell des Systems. Formuliere eine wünschenswerte Eigenschaft des Systems in einer Sprache wie Prädikatenlogik. Untersuche, ob das System(modell) die Eigenschaft (ausgedrückt als prädikatenlogische Formel) erfüllt.

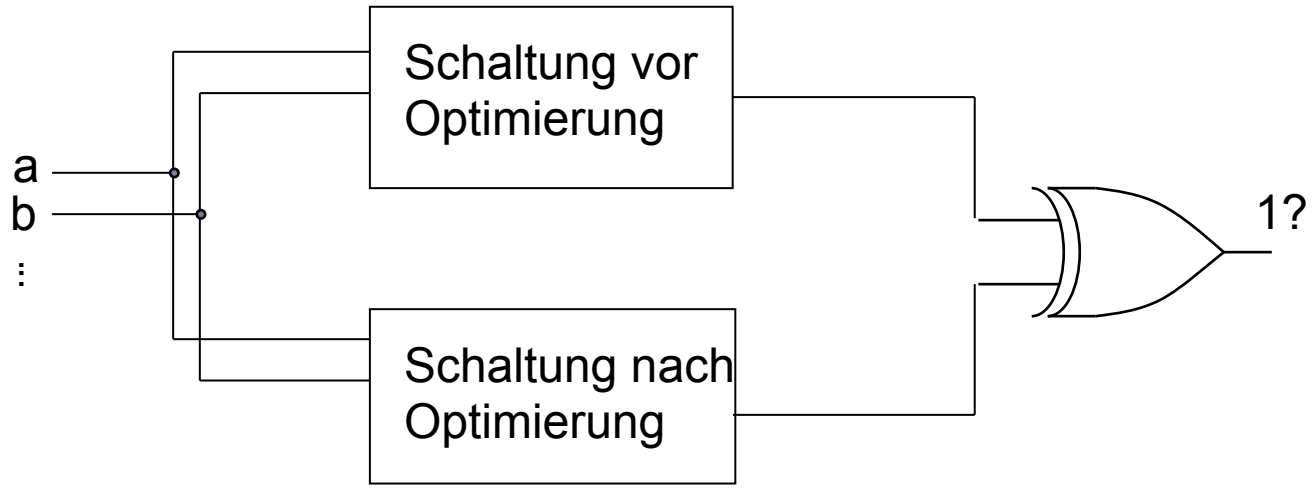
Praxis:
Die Verifikation großer Systeme durch Prädikatenlogik ist meist nichthandhabbar. (Man bedenke die grundsätzliche Nicht-Entscheidbarkeit!) Deswegen beschränkt man sich in der Praxis auf "einfachere" Logiken wie bestimmte temporale Logiken oder Aussagenlogik.

Diese Techniken sind Gegenstand der Vorlesung „Hardware Verification and Quality Assessment“.

Formale Verifikation

Beispiel: Äquivalenzvergleich

Formaler Äquivalenzvergleich zur Verifikation der Optimierungsschritte: Bugs in Synthesesoftware und manuelle Eingriffe des Designers können zu Fehlern in der Implementierung führen.



Schaltungen sind äquivalent und Implementierung korrekt, wenn Ausgangssignal nicht erfüllbar ist. Werkzeuge zum Äquivalenzvergleich beruhen auf Verfahren zur Überprüfung der Erfüllbarkeit einer Booleschen Funktion („SAT-solving“).