Algorithmen für Chip-Entwurfswerkzeuge

Einführung

TECHNISCHE UNIVERSITÄT DARMSTADT

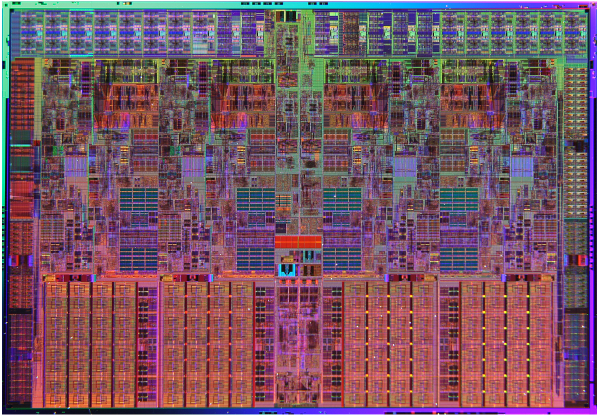
Vorlesung
WS 2012/2013

Florian Stock, Andreas Koch

Eingebette Systeme und Anwendungen
Technische Universität Darmstadt

Embedded Systems & Applications

Physikalischer Schaltkreis



Quelle: Intel

Hardwareentwicklungs-Flow

TECHNISCHE UNIVERSITÄT DARMSTADT

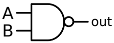
$\cdots \rightarrow$ Entwurf $\rightarrow$ Layoutsynthese $\rightarrow$ Layoutverifikation $\rightarrow$ Fertigung $\rightarrow \cdots$

Zentrum der Vorlesung:

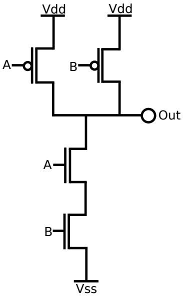
Layoutsynthese $\Rightarrow$

1. Partitionierung
2. Floorplanning
3. Platzierung
4. Verdrahtung
5. Kompaktierung

Sichten
Schematisch und Transistorlayout



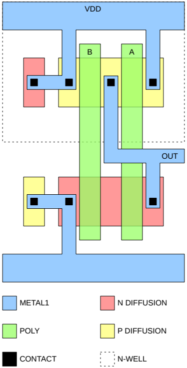
Schematisches Schaltsymbol



Bildquelle: Wikimedia Commons

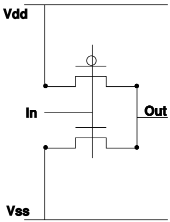
Transistorlayout

Sichten
Physikalisches/Geometrisches/Masken Layout

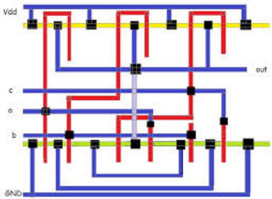


Bildquelle: Wikimedia Commons

Sichten
Symbolisches Layout)



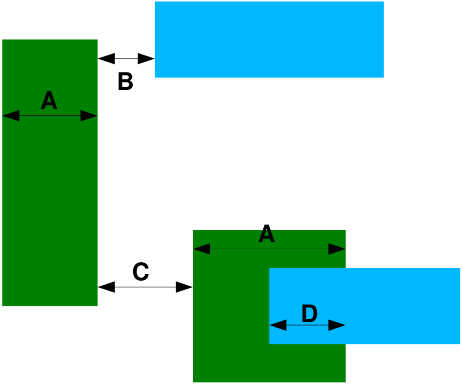
Symbolisches Layout



Stick-Diagramm

- Kein vollständiges Layout
- Keine absoluten geometrischen Angaben
- Nicht notwendige physikalische Angaben fehlen komplett (z.B. n- und p-Wells)
- *Symbole* für Elemente wie Transistoren oder Kontakte
- Länge, Breite, Layer noch variabel

Entwurfsregeln



- Bei ASIC-Layouts
  - Grundlage für erfolgreiche Fertigbarkeit
  - Von *Technologen* erarbeitet
- Üblicherweise:
  - Minimale Breite (A)
  - Minimaler Abstand (B,C)
  - Minimale Überlappung (D)
  - Werte vielfaches von λ

Kompaktierung
Motivation

- Komprimieren/Expandieren von Layouts
  - Unter Beachtung der Designregeln!
- Anwendungsgebiete:

  Layout-Compilierung von symbolischen in geometrische Layouts
  Flächenminimierung von bestehenden Layouts
  Korrektur von Entwurfsregelverletzungen
  Skalierung der Technologie

Kompaktierung
Vorgehensweise

TECHNISCHE UNIVERSITÄT DARMSTADT

Eindimensional (1D)

- Nur eine Richtung bearbeiten

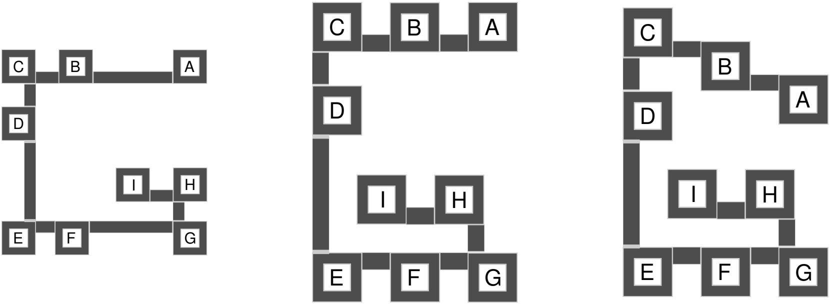
  Operationen: Bewegen, Stauchen
- Oft abwechselnd in X, Y Richtungen
- Problem: Effizent, aber suboptimal

Zweidimensional (2D)

- Beide Richtungen simultan bearbeiten
- Problem: Optimal, aber NP-hart

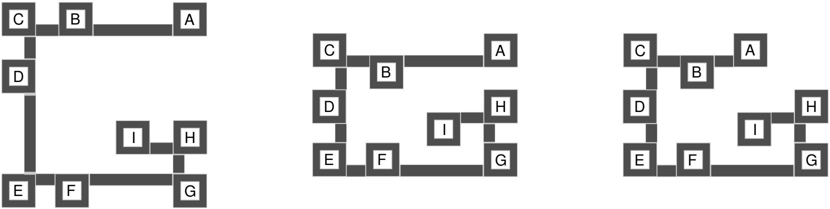
Kompaktierung
Graphisches Beispiel

TECHNISCHE UNIVERSITÄT DARMSTADT



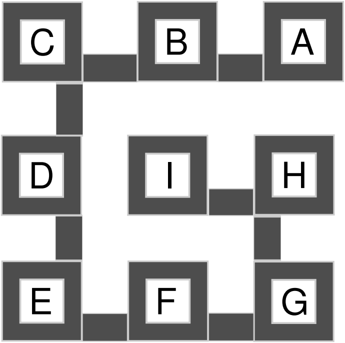
Original ⇒ Horizontal kompaktiert ⇒ Vertikal kompaktiert

Kompaktierung
Graphisches Beispiel



Original ⇒ Vertikal kompaktiert ⇒ Horizonal kompaktiert

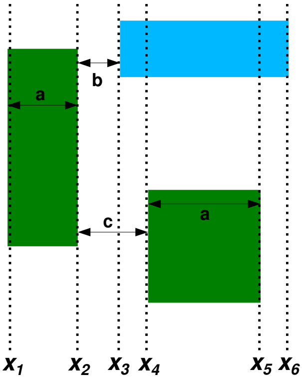
Kompaktierung
Optimum



Optimum

- 2D-Kompaktierung
  - Findet optimale Lösung
  - Problem: NP-Vollständig
- Tatsächliche Vorgehensweise
  - Mehrfache 1D-Kompaktierung
  - Abwechselnd horizontal, vertikal
  - Problem: Nicht optimal

Modellierung
Abstände $\mapsto$ Ungleichungen

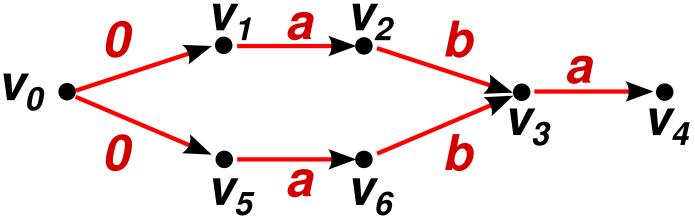


$$x_2 - x_1 \geq a$$
$$x_3 - x_2 \geq b$$
$$x_5 - x_4 \geq a$$
$$x_j - x_i \geq d_{ij}$$

Modellierung
Einschränkungsgraph

Eingschrängungsgraph $G(V, E)$

- Gerichtet von (v_i, v_j)
- Zyklenfrei
- Längster Pfad von v_0 zu v_j = Minimale Koordinate von x_i
- Modelliere x_n durch v_n



Modellierung
Maximale Abstände

- Bisher nur miminmale Abstände
- Maximale Abstände mathematisch: $|x_c - x_w| \leq d$
- $x_j - x_i \leq c_{ij}$ und $x_i - x_j \leq c_{ij}$, $c_{ij} \geq 0$
- Passende Form für unseren Einschränkungsgraph
  Achtung: Jetzt sind aber Zyklen möglich
- Lösung: Berechnung des Längesten Pfades in Graphen mit Zyklen
  Genauer: Einfacher Pfad (d.h. jede Kante max. einmal)

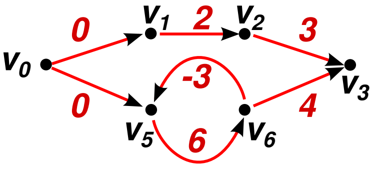
Längster Pfad

Algorithmus abhängig vom Graphen:

Zyklenfreier Graph: OK, ähnlich zu BFS

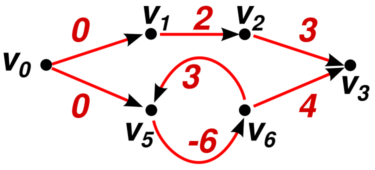
Graph mit Zyklen: Unterscheidung nach Zyklusart

Mit positivem Zyklus



Undefiniert!

Mit negativem Zyklus



OK

Längster Weg
Zyklenfreie Graphen

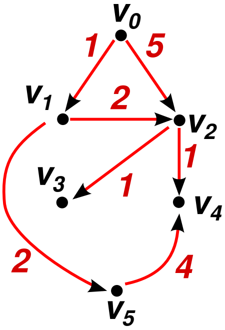
```
main()
begin
    foreach 0 ≤ i < n do
        x_i := 0
    longestPath(G) ;
```

Vorraussetzung:

Graph ist ein DAG
(Directed Acyclic Graph)!

```
longestPath(G):
begin
    foreach v_i in V do
        p_i := v_i.inDegree()
    Set Q := {v_0};
    while (Q ≠ ∅) do
        v_i := Q.pickany();
        Q := Q \ {v_i} ;
        foreach (v_i, v_j) ∈ E do
            x_j := max(x_j, x_i + d_ij);
            p_j := p_j - 1;
            if p_j ≤ 0 then
                Q := Q ∪ {v_j}
```

Längster Pfad
DAG Beispiel



Q	p_1	p_2	p_3	p_4	p_5	x_1	x_2	x_3	x_4	x_5
Init	1	2	1	2	1	0	0	0	0	0
$\{v_0\}$	0	1	1	2	1	1	5	0	0	0
$\{v_1\}$	0	0	1	2	0	1	5	0	0	3
$\{v_2, v_5\}$	0	0	0	1	0	1	5	6	6	3
$\{v_3, v_5\}$	0	0	0	1	0	1	5	6	6	3
$\{v_5\}$	0	0	0	0	0	1	5	6	7	3
$\{v_4\}$	0	0	0	0	0	1	5	6	7	3

Graphen mit Zyklen

- Nur mit *negativen* Zyklen
- Erkenne positive Zyklen
  $\Rightarrow$ Überbeschränkte Layouts
- Aber lokalisere sie nicht

Längster Pfad
Liao-Wong

TECHNISCHE UNIVERSITÄT DARMSTADT

foreach $0 \leq i < n$ **do**
  $x_i := -\infty$

$x_0 := 0$; loop_count = 0 ;
repeat
  is_modified := false ;
  longestPath(G_f) ;
  foreach $(v_i, v_j) \in E_b$ **do**
    if $x_j < (x_i + d_{ij})$ **then**
      $x_j := x_i + d_{ij}$;
      is_modified := true ;
  if $++$ *loop_count* $> |E_b|$ *&& is_modified)* **then**
    error(positive cycle!");
until *! is_modified*;

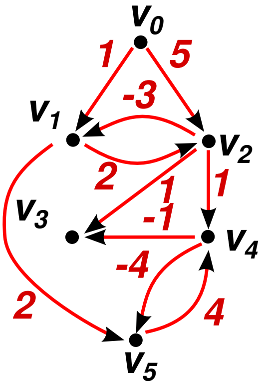
Idee: Zyklen auftrennen

- Kanten E_f: min. Distanz
- Kanten E_b: max. Distanz

$\Rightarrow$ Teilgraph $G_f(V, E_f)$

- Löse LongestPath(G_f)
- Korrigiere für entfernte E_b (Zyklen schließen)
- Jedes $e_b \in E_b$ max. $1\times$ im Pfad
  $\Rightarrow$ stabilisert sich in $|E_b|$
- Wenn nicht
  $\Rightarrow$ überbeschränkt

Längster Pfad
Liao-Wong Beispiel



Schritt	x_1	x_2	x_3	x_4	x_5
Init	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$
Vor 1	1	5	6	7	3
Zurück 2	2	5	6	7	3
Vor 2	2	5	6	8	4
Zurück 2	2	5	7	8	4
Vor 3	2	5	7	8	4
Zurück 3	2	5	7	8	4

- Verbesserung: longestPath(G_f) bemerkt Änderung
- $\mathcal{O}(|E_f| \times |E_b|)$
  d.h. besonders gut, falls $|E_b| \ll |E_f|$

Längster Pfad
Bellman-Ford

TECHNISCHE UNIVERSITÄT DARMSTADT

foreach $0 \leq i < n$ **do**
  $x_i := -\infty$

$x_0 := 0$; loop_count = 0 ;
$S_1 := \{v_0\}$; $S_2 := \emptyset$;
while *loop_count* $\leq n$ && $S_1 \neq \emptyset$ **do**
  foreach $v_i \in S_1$ **do**
    foreach $(v_i, v_j) \in E$ **do**
      if $x_j < (x_i + d_{ij})$ **then**
        $x_j := x_i + d_{ij}$;
        $S_2 := S_2 \cup \{v_j\}$;
  $S_1 := S_2$; $S_2 := \emptyset$;
  ++ loop_count ;
if *loop_count* $> n$ **then** error(positive cycle!");

Idee: Zwei Wellenfronten

S_1 aktuelle

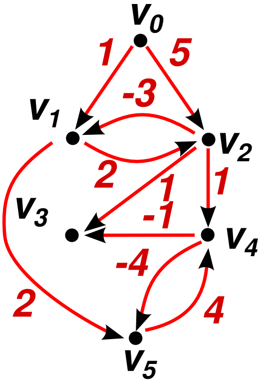
S_2 nächste Iteration

- Vergleichbar azyklischem LP
  aber mehrere Durchläufe
- In k-ter Iteration
  LP durch $k - 1$ Knoten

$\Rightarrow$ Zyklendetektion
LP $> n$ Knoten $\Rightarrow$ Zyklus!

- $\mathcal{O}(n^3)$, avg. $\mathcal{O}(n^{1.5})$

Längster Pfad
Bellman-Ford Beispiel



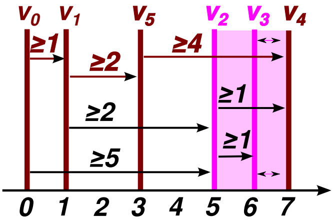
S_1	x_1	x_2	x_3	x_4	x_5
	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$
$\{v_0\}$	1	5	$-\infty$	$-\infty$	$-\infty$
$\{v_1, v_2\}$	2	5	6	6	3
$\{v_1, v_3, v_4, v_5\}$	2	5	6	7	4
$\{v_4, v_5\}$	2	5	6	8	4
$\{v_4\}$	2	5	7	8	4
$\{v_3\}$	2	5	7	8	4

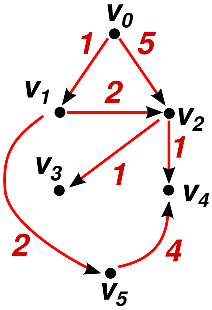
Pfadalgorithmen
Übersicht

- LP $\leftrightarrow$ SP bei Multiplikation der Gewichte mit -1
- Gerichtete zyklenfreie Graphen (DAG)
  SP und LP lösbar in linearer Zeit
- Gerichtete Graphen mit Zyklen
  - Alle Gewichte positiv
    $\Rightarrow$ SP in P, LP ist NP-vollständig
  - Alle Gewichte negativ
    $\Rightarrow$ LP in P, SP ist NP-vollständig
  - Keine positiven Zyklen: LP in P
  - Keine negativen Zyklen: SP in P
  - Sonst: NP-Vollständig

Kompaktierung
Kritische vs. Unkritische Element

- *Kritische Elemente* sind die Knoten entläng des Längsten Pfades
- Unkritisch alle anderen
- Layout-Breite hängt nur von kritischen Elementen ab
- Unkritische Elemente, verschiebbar
  Beeinflussen aber weitere Iterationen

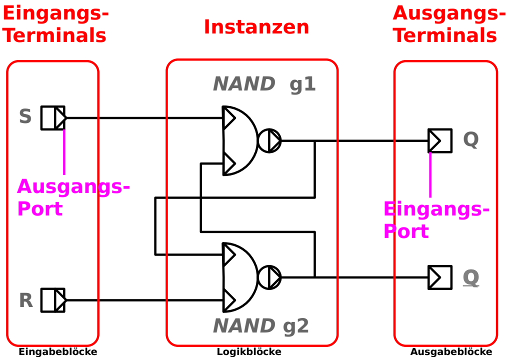




Kompaktierung
Weitergehende Details

- Freie Layoutelemente
  Optimale Lösung ist 2D-Kompaktierung
- Einfügen von Jogs
  Knicke in den Leitungen
- Berechnen der Einschränkungen
  Einfacher n^2-Ansatz: Redundanzen enthalten
- Hierarchisches Vorgehen

Darstellungen von Schaltungen



Zelle und Master-Zelle

```
class cell_master {
   String name;
   truth_table func;
   Rect extent;
   set<port_master> ins, outs;
   ...
}

class cell {
   cell_master master;
   String name;
   set<port> ins, outs;
   ...
}
```

Port und Portmaster

TECHNISCHE UNIVERSITÄT DARMSTADT

```
class port_master {
   String name;
   Point location;
   ...
}

class port {
   port_master master;
   String id;
   cell parent;
   net connects;
   ...
}
```

Netz

```
class net {
   String name;
   set<port> joined;
}
```

Schaltungsdarstellung
Objektzusammenfassung

TECHNISCHE UNIVERSITÄT DARMSTADT

Instanz oder Zelle
- Ein Auftreten einer *Master-Zelle*
- Speichert instanzspezifische Eigenschaften

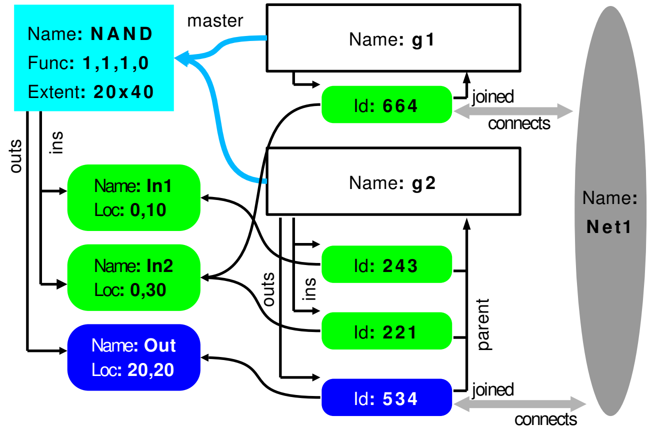
Master-Zelle Speichert Eigenschaften aller Instanzen

Netz Verbindung von mehreren Ports

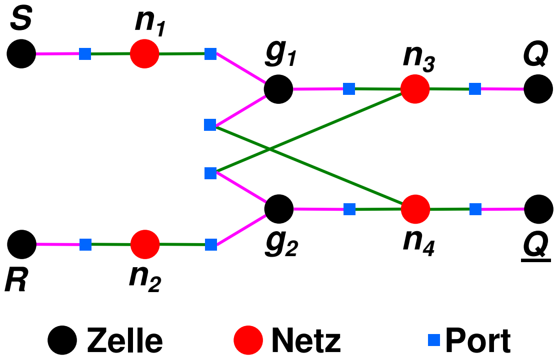
Port
- Anschlusspunkt von Leitung an Zelle
- I.d.R nicht untereinander austauschbar
- Hierarchie: Terminals werden zu Ports

Schaltungsdarstellung
Beispiel

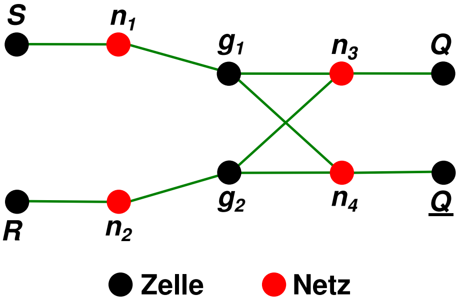
TECHNISCHE UNIVERSITÄT DARMSTADT



Schaltungsdarstellung – Graphmodellierung
Tripartiter Graph

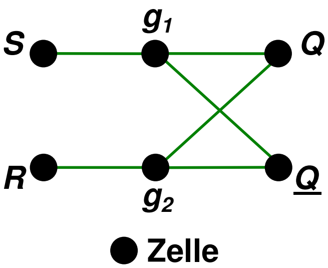


Schaltungsdarstellung – Graphmodellierung
Bipartiter Graph



- Weniger Details
- Verschmelzt Ports mit Zellen
- Äquivalent zu Hypergraph

Schaltungsdarstellung
Graphmodellierung



- Netze nicht mehr explizit modelliert
- Zellen an Netzen bilden jetzt Clique

Schaltungsdarstellungen
Übersicht

- Zelle-Port-Netz-Modell
- Tripartiter Graph
- Bipartiter Graph
- Clique-Modell

$\Downarrow$ ungenauer

Für Problem passendes Modell wählen

- Mehr Daten nicht immer besser

Konvertierungsroutinen bereitstellen

- Nur in ungenauere Darstellung möglich
- Buchführung über Herkunft von Daten

Zusammenfassung

TECHNISCHE UNIVERSITÄT DARMSTADT

- Kompaktierung
- Berechnung der Längsten Pfade
  - Mit und ohne Zyklen
- Modellierung von Schaltungen
  - Graphbasiert
  - Hierarchisch