

# Algorithmen im Chip-Entwurf 9

## Kanalverdrahtung und Verbesserungen für PathFinder

Andreas Koch  
FG Eingebettete Systeme  
und ihre Anwendungen  
TU Darmstadt

Kanalverdrahtung und PathFinder-Verbesserungen

1

# Überblick

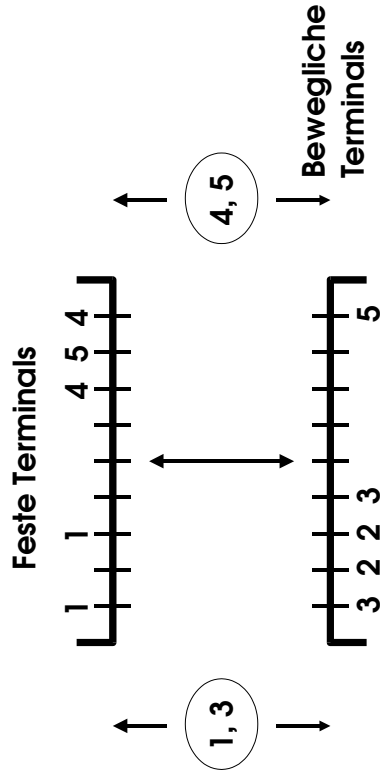
- Wiederholung
- Kanalverdrahtung
  - Yoeli's Robuster Router
  - Beispiel
- Verbesserungen für PathFinder
- Zusammenfassung

Kanalverdrahtung und PathFinder-Verbesserungen

2

## Kanalverdrahtung 1

- Verdrahtung von Netzen in rechteckigem Kanal



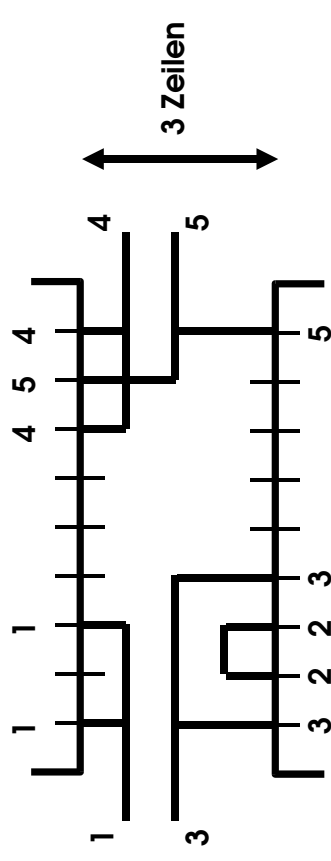
- Ziel: min. Fläche, (min. Länge, min. Vias)

Kanalverdrahtung und PathFinder-Verbesserungen

3

## Kanalverdrahtung 2

- Beispiel gelöst im klassischen Modell

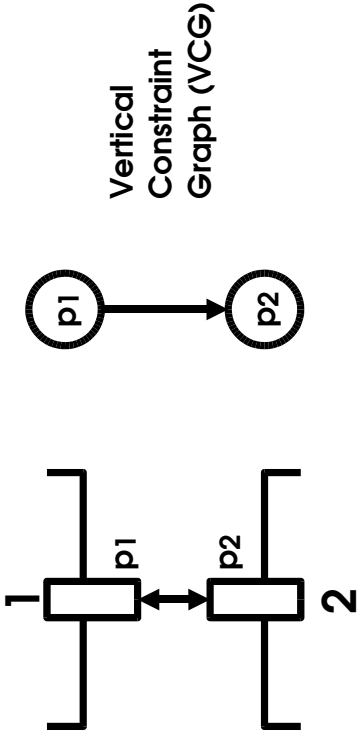


Kanalverdrahtung und PathFinder-Verbesserungen

4

# Vertikale Einschränkungen

- Zwei gegenüberliegende Terminals
  - Oberes Segment in den Kanal muß über unterem Segment in den Kanal liegen
  - ◆ Sonst Kurzschluß



# Horizontale Einschränkungen

- Im klassischen Modell
  - Keine Überlappung zwischen H-Segmenten verschiedener Netze in gleicher Zeile
  - Sonst Kurzschluß
- Horizontale Einschränkung
- Falls keine vertikalen Einschränkungen
  - Keine gegenüberliegenden Terminals
  - Lösung durch Left-Edge Algorithmus (1971)
- Was tun bei H+V Einschränkungen?
  - NP-vollständig!

# Robuster Kanal-Router 1

- Heuristik (Yoeli 1991)
- Algorithmus
  - Iteriert über alle Zeilen im Kanal
  - Verkleinert Problem mit jeder Iteration
  - Wechselt zwischen oberster / unterster Zeile
    - ◆ Arbeitet sich zur Kanalmitte vor
  - Zwei Phasen
    - ◆ Berechnen von Gewichten für Netze
      - ◆ Wie gut wäre aktuelle Zeile für Netz?
    - ◆ Selektion von Untermenge mit maximalem Gewicht
      - ◆ Heuristik bei Verletzung vertikaler Einschränkungen

# Robuster Kanal-Router 2

- Berechnung der Gewichte  $w_i$  für Netz  $i$
- Falls  $i$  Spalten der maximalen Dichte überspannt,
  - ◆ Erwartete Verringerung der max. Dichte, unabhängig von Seite (steepest descent)
    - $w_i \propto B$  (B groß)
- Falls  $i$  ein Terminal auf der aktuellen Seite (oben / unten) auf Spalte  $x$  hat,
  - $w_i \propto d(x)$  (für alle Spalten  $x$ )
- Bevorzuge Netze mit Terminals auf aktueller Seite
- Für alle Spalten  $x$  bei denen eine vertikale Einschränkung verletzt würde,
  - ◆ Bestrafe verletzte Einschränkungen
    - $w_i \propto K d(x)$  ( $5 \leq K \leq 10$ )

## Robuster Kanal-Router 3

- Regeln typisch für Heuristiken
- Robust
  - Unempfindlich gegen kleine Änderungen
- Nach Bestimmung der Gewichte
  - Finde Netz-Untermenge mit maximalem Gewicht, die in selbe Zeile passen
    - ◆ Ohne Verletzung horizontaler Einschränkungen
  - Verwendet Intervallgraph
    - ◆ Kante zwischen Knoten überlappender Intervalle

Kanalverdrahtung und PathFinder-Verbesserungen

9

## Robuster Kanal-Router 4

- Unabhängige Menge
  - Menge unverbundener Knoten
- Also gesucht:
  - Unabhängige Menge maximalen Gewichts
    - ◆ Im allgemeinen NP-vollständig
    - ◆ Aber für Intervallgraphen in P!
- Vorgehensweise
  - Dynamic Programming
  - Konstruiere optimale Lösung aus Teillösungen
    - ◆ Komplexitätsparameter  $\gamma$ :  $1 \leq \gamma \leq \text{Kanallänge}$

Kanalverdrahtung und PathFinder-Verbesserungen

10

## Robuster Kanal-Router 5

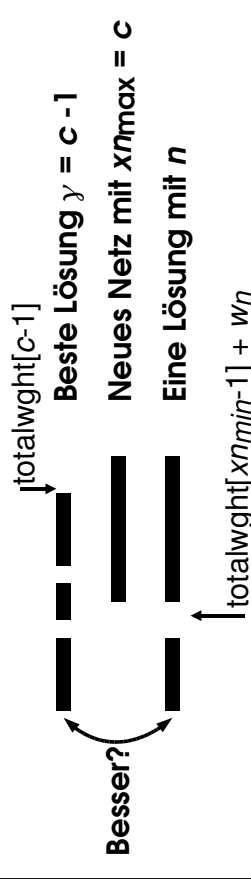
- $\gamma = \text{Spalte } c$
- Betrachte nur Netze mit rechtem Ende  $\leq c$
- Beispiel
  - $i_1 = [1, 4], i_2 = [12, 15], i_3 = [7, 13], i_4 = [3, 8], i_5 = [5, 10], i_6 = [2, 6], i_7 = [9, 14]$
  - $\gamma = 0, \gamma = 1, \gamma = 2, \gamma = 3: \emptyset$
  - $\gamma = 4, \gamma = 5: \{i_1\}$
  - $\gamma = 6, \gamma = 7: \{i_1, i_6\}$
  - $\gamma = 8: \{i_1, i_6, i_4\}$
  - ...

Kanalverdrahtung und PathFinder-Verbesserungen

11

## Robuster Kanal-Router 6

- Bestimme Lösung  $\gamma = c$  aus Lösung  $\gamma < c$
- Altes Maximalgewicht *plus* Netz  $n$  mit rechtem Ende in Spalte  $c$ 
  - ◆ Es ex. max. zwei solcher Netze (Term. oben / unten)
- $n$  Teil der optimalen Lösung, falls
  - ◆ Gewicht von  $n$  plus Gewicht bestehender Netze ohne Überlappung mit  $n \geq \text{max. Gewicht ohne } n$



Kanalverdrahtung und PathFinder-Verbesserungen

12

## Robuster Kanal-Router 7

- Für Spalte  $c$  ausgewähltes Netz merken
  - In `selected_net[c]`
  - Kann leer sein ( $=0$ , kein neues dazugekommen)
  - Letztes ( $=$ rechtes) Netz immer in Lösung
  - Dann nach links suchen
    - ◆ Nach nicht-überlappendem Netz
    - Wiederhole bis linker Rand erreicht!
- Beispiel: ...,  $i_2=[5,9]$ ,  $i_3=[4,6]$ , ...,  $i_7=[1,3]$ , ...  
 $c=$  1 2 3 4 5 6 7 8 9  
 $s_n[c]=$  0 0 7 0 0 3 0 0 2
  - $i_2$  in Lösung, überspringe  $i_3$ ,  $i_7$  in Lösung

Kanalverdrahtung und PathFinder-Verbesserungen

13

## Robuster Kanal-Router 8

- Annahme:  $d_{max}$  Durchgänge reichen
  - Wäre dann optimale Lösung
- Iteration
  - ◆ Gewichtsberechnung
  - ◆ Konstruiere
    - ◆ Maximal-gewichtige unabhängige Menge
- Aber:
  - Nur Versuch der Vermeidung von V-Konflikten
    - ◆ Keine Garantie!

Kanalverdrahtung und PathFinder-Verbesserungen

14

## Robuster Kanal-Router 9

- Falls V-Konflikt unvermeidbar
    - Entferne ein oder mehrere Netze
      - ◆ Heuristiki
    - Verdrahte Netz(e) mit Maze-Routing
      - ◆ Gute Umgebung: Viele Hindernisse!
    - Rip-up and Reroute
  - Auch hier: Keine Garantie
- Erneuter Durchlauf mit zusätzlicher Zeile
- $d_{max}$  war nur untere Schranke für Zeilenzahl
- Ggf. auch zusätzliche Spalte

Kanalverdrahtung und PathFinder-Verbesserungen

15

## Robuster Kanal-Router 10

```
robust_router(placed_netlist M) {
  set<int> row;
  seq<set<int>> S;
  int[channel_width+1] totalwght, selected_net;
  bool top;
  int height, c, r, i;
  top := true;
  height := N.dmax();
  for (r:=1; r<= height; ++r) {
    forall "Netze i in netlist N"
      wi := i.compute_weight(N, top);
      totalwght[0] := 0;
      for (c:=1; c<= channel_width; ++c) {
        selected_net[c] := 0;
        totalwght[c] := totalwght[c-1];
        if (n = "Netz mit Term. oben in Spalte c") {
          if (wn + totalwght[xrmin-1] > totalwght[c]) {
            totalwght[c] := wn + totalwght[xrmin-1];
            selected_net[c] := r;
          }
          if (n = "Netz mit Term. unten in Spalte c") {
            if (wn + totalwght[xrmin-1] > totalwght[c]) {
              totalwght[c] := wn + totalwght[xrmin-1];
              selected_net[c] := r;
            }
          }
        }
      }
      S.append(row);
      top := !top;
      N := "Ohne Netze in row";
    }
  }
  "Maze-Routing bei V-Konflikten"
}
```

- Ggf. Wiederholung mit
  - Erhöhter Breite
  - Erhöhter Länge

Kanalverdrahtung und PathFinder-Verbesserungen

16

# Robuster Kanal-Router 11

VCG

$B = 1000, K = 5$   
 $w_1 = (0) + (1) + (-5 \cdot 2) = -9$   
 $w_2 = (1000) + (2) + (-5 \cdot (2+3)) = 977$   
 $w_3 = (1000) + (2+2) + (-5 \cdot 0) = 1004$   
 $w_4 = (1000) + (3) + (-5 \cdot 2) = 993$

$totalwght[1]=0$   
 $totalwght[2]=\max(0,0-9)=0$   
 $totalwght[3]=0$   
 $totalwght[4]=\max(0,0+977)=977$   
 $totalwght[5]=\max(977,0+1004,0+993)=1004$

$sel[1]=0$   
 $sel[2]=0$   
 $sel[3]=0$   
 $sel[4]=2$   
 $sel[5]=3$

$x$  1 2 3 4 5  
 1 2 2 3 2  
 $d(x)$  1 2 2 3 2

$i_1=[1,2]$   $i_2=[2,4]$   $i_3=[3,5]$   $i_4=[4,5]$

# Robuster Kanal-Router 12

VCG

$w_1 = (1000) + (2) + (-5 \cdot 0) = 1002$   
 $w_2 = (1000) + (2) + (-5 \cdot 2) = 992$   
 $w_4 = (1000) + (1) + (-5 \cdot 2) = 991$

$totalwght[1]=0$   
 $totalwght[2]=\max(0,0+1002)=1002$   
 $totalwght[3]=1002$   
 $totalwght[4]=\max(1002,0+992)=1002$   
 $totalwght[5]=\max(1002,1002+991)=1993$

$sel[1]=0$   
 $sel[2]=1$   
 $sel[3]=0$   
 $sel[4]=0$   
 $sel[5]=4$

$x$  1 2 3 4 5  
 1 2 3 4 5  
 $d(x)$  1 2 1 2 1

$i_1=[1,2]$   $i_2=[2,4]$   $i_4=[4,5]$

# Robuster Kanal-Router 13

VCG

**Trivial: Netz 2 in Zeile 2**  
**Kombinierte Lösung**

- V-Konflikt:
  - 1. Zeile: Netz 3
  - 2. Zeile: Netz 2
  - 3. Zeile: Netz 1, Netz 4

rip-up & reroute

$i_1=[1,2]$   $i_2=[2,4]$   $i_3=[3,5]$   $i_4=[4,5]$

# Wiederholung VPR/PathFinder

- Verdrahtungsproblem auf FPGAs
- Verdrahtbarkeitsorientierte Verdrahtung
- PathFinder-Algorithmus
  - Gewichteter Maze-Router
  - $p_v, h_v$
- Erweiterung auf Verzögerung
  - Durch Criticality

# Vergleich

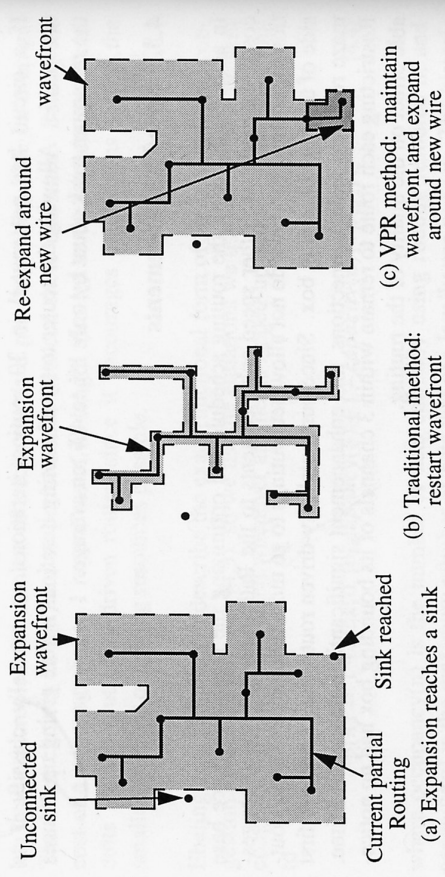
## ■ Pathfinder [McMurchie&Ebeling 1995]

- Zunächst nur verdrahtungsorientiert
- Keine vorgegebene Sink-Reihenfolge
- Wellenausbreitung
  - ◆ Bis alle Sinks erreicht

## ■ Verbesserbar

- Alte Wellenfront in PQ nicht verwerfen
  - ◆ Einfach neue Sink an RT anschliessen
  - ◆ Neue Segmente in PQ übernehmen
- Bei Verzögerungsorientierung
  - ◆ Jetzt steht Sink-Reihenfolge fest
    - ◆ Absteigende  $A_j$  (vergleichbar Criticality)

# Wellenausbreitung



# Verbesserungen

- Swartz, Betz, Rose 1998 (U Toronto)
- Optimierung auf Geschwindigkeit
  - Qualitätsverlust?
- Zwei Kernideen
  - Gezielte Ausbreitung statt breiter Wellenfront
  - Beschränkung auf sinnvolle Startpunkte
- Diverse Detailverbesserungen



# Ausbreitung 1

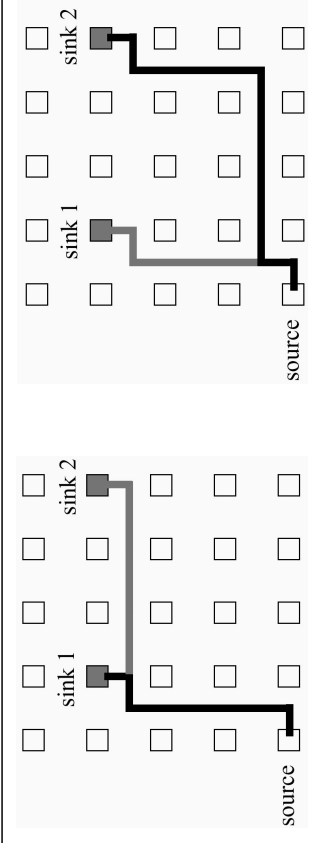
- Gerichtete Tiefensuche DDFS statt BFS
  - Suche bevorzugt in Richtung auf Ziel  $j$  zu

$$\text{Cost}(i, v) = \text{PathCost}(i, u) + C_0 + \alpha \cdot \Delta D$$

- ◆  $\text{PathCost}(j, u)$ : Kosten bis zum Vorgänger  $u$
- ◆  $C_0$ : Verdrahtungsabhängige Basiskosten
  - ◆ Vergleichbar  $c_w$ , wächst aber viel stärker
  - ◆ Weniger Iterationen
- ◆  $\Delta D$ : Manhattan-Distanz von  $v$  zum Ziel  $j$ 
  - ◆  $<0$ :  $v$  liegt näher an  $j$  als  $u$  (= billiger)
  - ◆  $>0$ :  $v$  liegt weiter von  $j$  als  $u$  (= teurer)
- ◆  $\alpha$ : Richtungsfaktor
  - ◆ =0: BFS, keine richtungsabhängigen Komponenten
  - ◆  $>0$ : Nicht mehr verdrahtungsorientiert, Greedy
  - ◆ =1.5: Empfohlen, hohe Beschleunigung, gute Qualität



## Ausbreitung 2



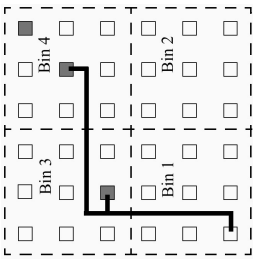
- **Reihenfolge der Sinks**
  - Nächstegelegene zuerst
    - ◆ Besser Anschließbarkeit der Folgenden
- **Reihenfolge der Netze**
  - Viele Terminals zuerst
    - ◆ Vermeidung von Blockaden

Kanalverdrahtung und PathFinder-Verbesserungen

25

## Sinnvolle Startpunkte

- **PathFinder/VPR**
  - Ausbreitung von gesamten RT aus
    - ◆ Übernahme in PQ mit Kosten 0
  - Ineffizient, gerade bei vielen Terminals
- **Idee**
  - Nur Segmente aus RT „nahe“ beim Ziel in PQ
  - Aufteilen der gesamten Fläche in Bins
    - ◆ Hier:
      - ◆ Nur Segmente in Bin 4 expandieren
- **Lohnend bei**
  - Netzen mit vielen Terminals



Kanalverdrahtung und PathFinder-Verbesserungen

26

## Binning Details

- **Bin-Größe**
  - Sollte passen
  - Berechnung pro Netz
    - ◆ Durchschnittliche Fläche pro Sink  $A_s = \text{netzBB} / \#\text{sinks}$
    - ◆ Bewährt: Bin-Größe  $4x A_s$
  - Expandiere
    - ◆ Nur Segmente im gleichen Bin wie nächstes Ziel
      - ◆ Einfache Entfernungsberechnung, kein Bin-Raster
- **Leere Bins**
  - Bin mit Ziel enthält noch keine RT-Segmente
  - Erweitere Suchradius auf 8 Nachbar-Bins
  - Falls immer noch leer
    - ◆ Suche von ganzem RT aus

Kanalverdrahtung und PathFinder-Verbesserungen

27

## Auswirkungen

- **Low-Stress Routing**
  - >10% mehr Tracks als minimal erforderlich
- **15 Beispielschaltungen**
- **Durchschnittliche Rechenzeit**
  - BFS in VPR: 731s
  - DDFS: 14s
  - DDFS+Bins: 7s
- **Durchschnittlicher Qualitätsverlust**
  - BFS in VPR: 15.5 Tracks
  - DDFS: 15.5 Tracks
  - DDFS+Bins: 15.8 Tracks

Kanalverdrahtung und PathFinder-Verbesserungen

28

## Praktikum

- Nicht genau nachprogrammieren
  - Viele Details nicht gezeigt
- Konzepte verstehen
- Inspiration für eigene Ideen
- Sinnvoll
  - Routing Graph
  - Darin nach Verdrahtungen suchen
- Papers auf Web-Site
  - PathFinder, McMurchie & Ebeling 1995
  - Verbesserungen von Swartz et al., 1998
  - Auszüge aus VPR Beschreibung, 1999 [19MB!]

Kanalverdrahtung und PathFinder-Verbesserungen

29

## Zusammenfassung

- Wiederholung Kanalverdrahtung
- Yoeli's Robuster Router
  - Beispiel für komplexere Heuristik
    - ◆ Regeln
  - Ausführliches Beispiel
- Verbesserungen für PathFinder-Algorithmus
  - Schnellere Berechnung
  - Bei minimalem Qualitätsverlust
- Weiteres Vorgehen
  - Donnerstag, 15.12.2005
    - ◆ Kolloquien
  - Freitag, 16.12.2005
    - ◆ Vorträge
  - Dienstag, 20.12.2005
    - ◆ Globale Verdrahtung: Abschnitte 9.4-9.5

Kanalverdrahtung und PathFinder-Verbesserungen

30