Ein Tool zur Verifikation von Standardzell-Layouts

Diplomarbeit

Olav Kahlbaum
Berliner Str 64, 23627 Gr. Grönau

Fachbereich Informatik
Arbeitsbereich Technische Grundlagen der Informatik
Universität Hamburg

Betreuer:
Dr. R. Rauscher
Prof. Dr. K. Kaiser

Mai 1997

Inhaltsverzeichnis

Tabellenverzeichnis

Abbildungsverzeichnis

Kapitel 1

Einführung

Seit vor etwa dreißig Jahren die ersten Mikrochips produziert wurden, werden ständig neue Werkzeuge entwickelt, um die Plazierung und die Verdrahtung (Placement and Routing) zu automatisieren. Je größer der Chip (Anzahl der integrierten Komponenten), desto größer wird die Notwendigkeit zur Automatisierung. Es gibt eine Vielzahl von Veröffentlichungen und Ansätzen zur Entwicklung von Heuristiken zum Placement und Routing. Wichtige Meilensteine bei der Entwicklung von Heuristiken zur Plazierung und Verdrahtung sind z. B. der Lee-Algorithmus [Lee61] und der Timber-Wolf-Algorithmus [Sec85, She89]. Es wird ständig versucht, durch neue Heuristiken (z. B. DKROUTER [Kla94]), die Ergebnisse der Plazierung und Verdrahtung zu verbessern. Die Zielfunktionen sind dabei teilweise widersprüchlich. Zur Verbesserung der Ergebnisse gehört unter anderem die Minimierung der Netzlängen, die Verkleinerung der für die Verdrahtung benötigten Chipfläche und die Minimierung der Anzahl der Durchkontaktierungen (Kontaktlöcher). Solche Verbesserungen führen wiederum zu geringeren Herstellungskosten und auch zu leistungsfähigeren Chips: Es kann mehr Funktionalität auf einer kleineren Fläche integriert werden, und höhere Taktfrequenzen werden möglich. Auch wird versucht, die für die eigentliche Plazierung und Verdrahtung benötigte Zeit möglichst gering zu halten. Um die Verbesserungen bei neu entwickelten Heuristiken überprüfen zu können, ist eine Evaluierung durch Vergleiche der Ergebnisse mit den Ergebnissen bestehender Heuristiken erforderlich.
Die Evaluierung neu entwickelter Heuristiken zum VLSI-Placement und -Routing und insbesondere der Vergleich mit bestehenden Heuristiken ist jedoch oft mit folgenden Problemen verbunden:

- Unterschiedliche Formate der Ein- und Ausgabedaten: Damit arbeitet jeder Algorithmus auf anderen Daten, und es ist keine direkte Vergleichsmöglichkeit dieser Daten gegeben. Für einen sinnvollen und aussagekräftigen Vergleich ist es jedoch unbedingt erforderlich, daß alle Algorithmen auf denselben Daten arbeiten und diese direkt vergleichbar sind. Außerdem sind einheitliche Beispielschaltungen erforderlich.

- Unterschiedliche Technologieabhängigkeiten: Ein Vergleich von Heuristiken, die auf unterschiedlichen Technologien arbeiten, ist schwierig oder sogar unmöglich, da dann die Randbedingungen für die zu vergleichenden Heuristiken unterschiedlich sind. So ist z. B. *Routing over the Cell*, das zu einer Verkleinerung der Chipfläche führen würde, nur bei einigen wenigen Technologien erlaubt, bei den meisten anderen jedoch verboten.

- Abhängigkeiten von verschiedenen Rechner- und Betriebssystemen: Auch durch dieses Problem werden Vergleiche zumindest erschwert.

Um diese Probleme, die die Evaluierung von Heuristiken zum Placement und Routing mit sich bringt, lösen zu können, ist im Rahmen einer Kooperation mit dem Fachbereich Mathematik das System ESPRO (Evaluation System for Placement and ROuting) entstanden. Dieses System wurde im Rahmen einiger Diplomarbeiten [Kla94, Kas95] und einer Studienarbeit [Kah95] entwickelt und implementiert. Die Grundidee dieses Systems ist, daß im Prinzip jeder, der eine neue Heuristik zu

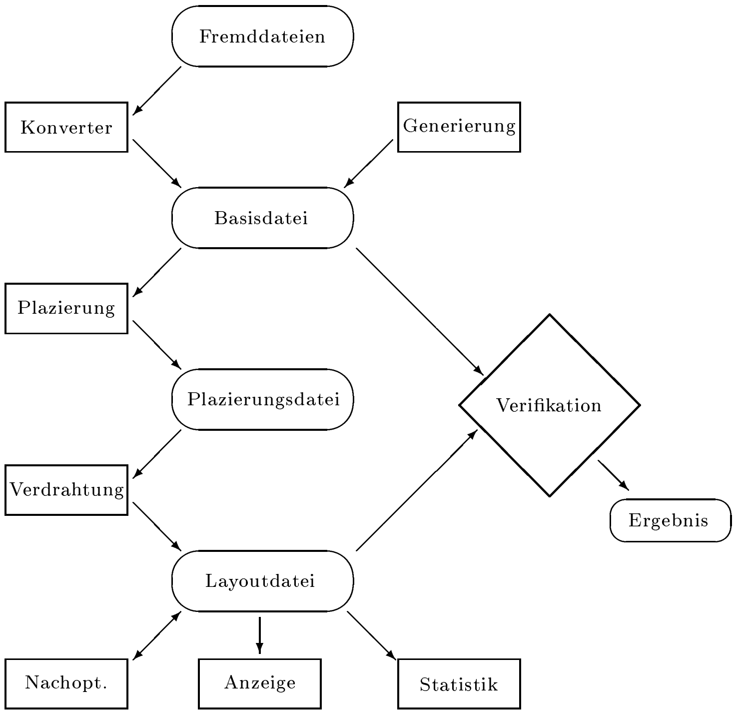

Abbildung 1.1: berblick ber das ESPRO-System mit bisher fehlender Verifikation

Placement und Routing entwickelt hat, diese in das System integrieren und dann mit den bereits zur Verfügung stehenden (Standard-) Heuristiken vergleichen kann.
Im ESPRO-System werden die genannten Probleme, die beim Vergleich von Plazierungs- und Verdrahtungsheuristiken auftreten, auf folgende Weise gelöst:

- Das System besteht aus voneinander unabhängigen Tools, die über Dateien mit genau definierten Formaten miteinander kommunizieren. Damit gibt es einheitliche Formate für die Ein- und Ausgabedaten, und jeder Algorithmus kann auf denselben Daten arbeiten. Auch die direkte Vergleichbarkeit der Ein- und Ausgabedaten ist auf diese Weise gegeben, und für jeden Algorithmus stehen dieselben Beispielschaltungen zur Verfügung. Außerdem entsteht so eine offene Systemarchitektur (Abb. 1.1), die es gestattet, jederzeit und auf einfache Weise neue Tools zu dem bestehenden System hinzufügen zu können. Man muß dazu lediglich die Formate der einzelnen Dateitypen kennen.

- Die Layouts werden rein geometrisch betrachtet, und *Routing over the Cell* ist generell verboten. Dadurch enthält das System keine Technologieabhängigkeiten, so daß die Randbedingungen immer gleich bleiben.

- Das System kann durch portable Programmierung für verschiedene Rechner- und Betriebssysteme verfügbar gemacht werden. Es gibt also auch hier keine Abhängigkeiten.

Damit ist ESPRO ein System, das den Vergleich verschiedenener Plazierungs- und Verdrahtungsheuristiken auf denselben Daten und unter denselben Randbedingungen ermöglicht. Die bisher für dieses System entwickelten Programme sind alle für MS-DOS, einige dieser Tools auch für UNIX verfügbar.
Es kann also im Prinzip jeder, der Zugang zu einem PC hat, mit diesem System arbeiten und auch neue Tools hinzufügen. Es sind keine teuren Workstations und keine langen Einarbeitungszeiten

nötig. ESPRO kann damit sehr gut sowohl in der Forschung als auch in der Lehre eingesetzt werden. Da das gesamte System frei verfügbar ist, kann jeder, der daran interessiert ist, das System auf seinem eigenen PC erproben.
Beispielschaltungen können entweder aus Fremddateien (z. B. dem SOLO-1400-System) konvertiert oder durch manuelle Eingabe oder einen Zufallsgenerator erzeugt werden. Eine Schaltung wird durch eine Basisdatei definiert, die die Netzlisten enthält. Außerdem enthält die Basisdatei Informationen über die Anzahl der Pads und die Anzahl und Größen der Zellen. Für die Plazierung der Zellen sind verschiedene Tools (Placer) verfügbar (Great-Deluge-Placer (Implementierung: [Kla94]), Timber-Wolf-Placer (Implementierung: [Kah95]) sowie genetische Placer). Die Plazierung wird in eine Plazierungsdatei geschrieben. Für die Verdrahtung ist bisher der in [Kla94] entwickelte und implementierte Router DKROUTER verfügbar. Auch wurde in [Kah95] der Lee-Algorithmus implementiert, um den innovativen DKROUTER mit einer Standard-Heuristik vergleichen zu können. Das Ergebnis der Verdrahtung, also das Layout eines Chips, wird in eine Layoutdatei geschrieben. Für die Analyse von Layoutdateien sind Tools verfügbar, die das Layout auf dem Bildschirm anzeigen oder statistisch analysieren und Entwurfswerte ermitteln können. Außerdem wurden in [Kas95] Tools entwickelt und implementiert, mit denen Nachoptimierungen von Layouts durchgeführt werden können.
Als sehr problematisch ist jedoch anzusehen, daß bisher keine Überprüfung stattfindet, ob eine durch die Layoutdatei beschriebene Verdrahtung fehlerfrei ist und genau die Verbindungen implementiert, die durch die in der dazugehörigen Basisdatei gegebenen Netzlisten vorgegeben sind. Sofern dies nicht gewährleistet ist, ist die Analyse der Entwurfswerte (Chipfläche, Fläche für Verdrahtung, Netzlängen) und der Vergleich der verschiedenen Heuristiken mit Hilfe dieser Werte zumindest zweifelhaft. So kann es beispielsweise passieren, daß aufgrund eines Programmfehlers im Router Leitungen, die zu verschiedenen Netzen gehören, übereinandergelegt werden. Damit würde die Chipfläche zwar kleiner werden, es entstehen jedoch Kurzschlüsse zwischen den betroffenen Netzen, so daß die Schaltung nicht mehr funktionieren würde. Die Entwurfswerte eines solchen Layouts (in diesem Beispiel also die Chipfläche) wären damit verfälscht und könnten nicht für Vergleiche benutzt werden. Auch ist vor der Fertigung eines Chips eine Verifikation des Layouts nötig, um sicherstellen zu können, ob die Fertigung dieses Chips überhaupt möglich und der Chip nach der Herstellung auch funktionsfähig ist. Eine solche Verifikation wird z. B. in [Yos86] gefordert und beschrieben.
Im Rahmen dieser Diplomarbeit wurde deshalb ein Tool entwickelt und implementiert, das ein Layout analysiert und dabei auf mögliche Fehler überprüft sowie die Netzlisten extrahiert und mit den Netzlisten aus der dazugehörigen Basisdatei vergleicht.
Dadurch wird ESPRO um das bisher noch fehlende Verifikationstool ergänzt und so nach außen hin abgerundet. Das Verifikationstool ermöglicht die Kontrolle von Layouts und damit der Ergebnisse von Routern und Nachoptimierern. Es ist mehr Sicherheit bei der Analyse der Ergebnisse gegeben, da man die Möglichkeit hat, zu überprüfen, ob Entwurfswerte eines Layouts eventuell durch Layoutfehler verfälscht sind. Die Vergleiche, die mit Hilfe von ESPRO durchgeführt werden können, werden so noch sicherer und aussagekräftiger.

Kapitel 2

Motivation und Aufgabendefinition

In diesem Kapitel wird die Motivation dargelegt, für das ESPRO-System ein Verifikationstool zu entwickeln. Aus dieser Motivation heraus wird dann die eigentliche Aufgabendefinition entwickelt. Außerdem werden alle in dieser Arbeit verwendeten Fachbegriffe definiert.

2.1 Motivation

Wie schon im ersten Kapitel dargestellt, findet im ESPRO-System bisher keine Überprüfung statt, ob eine in einer Layoutdatei enthaltene Verdrahtung fehlerfrei ist und ob diese genau die und nur die Netzlisten implementiert, die in der dazugehörigen Basisdatei enthalten sind. Solange dies nicht gewährleistet werden kann, ist ein Vergleich der Ergebnisse verschiedener Heuristiken aufgrund der Entwurfswerte (Chipfläche, Netzlängen, für die Verdrahtung benötigte Fläche) zumindest zweifelhaft, da Fehler im Layout zu einer Verfälschung der Entwurfswerte führen und Vergleiche dadurch unbrauchbar werden. So können z. B. folgende Verfälschungen auftreten:

- Durch fehlende Verbindungen kann ein Netz kürzer werden, als es eigentlich sein müßte. Auch kann durch die so unzulässigerweise "eingesparten" Segmente die Chipfläche kleiner werden. Fehlende Verbindungen führen auch zu geringerer Kanalauslastung (im Extremfall können Kanalbreiten sogar den Wert Null annehmen) und zu einer Reduktion der kritischen Bereiche. Es werden dadurch also viele Entwurfswerte, unter anderem die Netzlängen und die Chipfläche, verfälscht, so daß diese Werte nicht für einen Vergleich verwendet werden könnten. Würde ein solcher Chip hergestellt werden, wäre dieser nicht funktionsfähig.

- Wenn Segmente, die zu verschiedenen Netzen gehören, übereinandergelegt werden, entstehen dadurch Kurzschlüsse zwischen diesen Netzen. Auch dadurch wird z. B. die Chipfläche verfälscht. Ein solcher Chip ist außerdem nicht funktionsfähig.

Dies sind nur zwei Beispiele, um zu erläutern, weshalb eine Überprüfung eines Layouts nötig ist. Die Liste möglicher Verfälschungen der Entwurfswerte läßt sich beliebig fortsetzen. Verursacht werden solche Layoutfehler durch Programmierfehler in den Programmen, die die Verdrahtungs- bzw. Nachoptimierungsheuristiken implementieren (Router bzw. Nachoptimierer). Da solche Programme groß und komplex sind, ist es kaum möglich, diese direkt zu verifizieren. Es bleibt also nur die Möglichkeit, die Ergebnisse der Router und Nachoptimierer, also die Layouts, zu überprüfen. Eine manuelle Überprüfung eines Layouts (durch Verfolgen der Verbindungen in der grafischen Darstellung am Bildschirm) ist jedoch höchstens bei sehr kleinen Schaltungen (mit bis zu zehn Zellen) möglich und auch dann schon sehr mühsam und zeitaufwendig (s. Abbildung 2.1). Es wird also ein Tool benötigt, das die Überprüfung eines Layouts automatisiert, so daß diese schnell durchführbar ist. Das Verifikationstool sollte alle Fehler, die es findet, möglichst detailliert ausgeben, so daß es mit Hilfe dieser Fehlermeldungen auch möglich ist, die Fehler im Router oder Nachoptimierer einzugrenzen, zu finden und zu beseitigen.

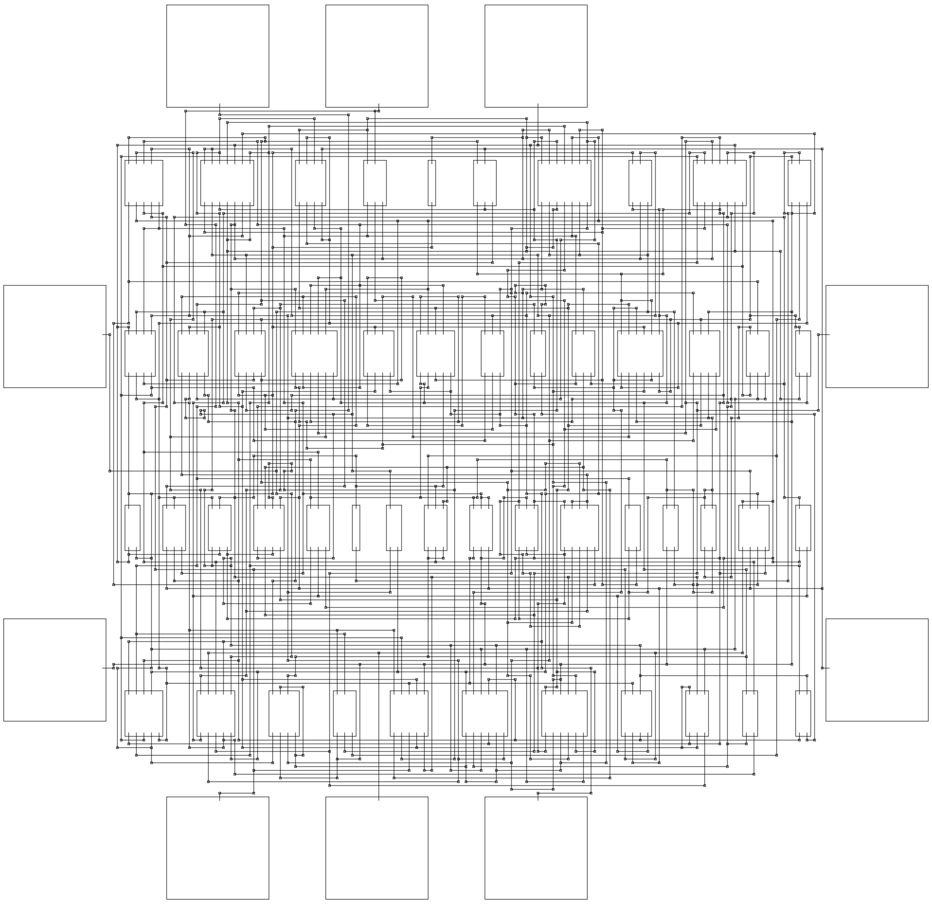

Abbildung 2.1: Beispielchip mit 50 Zellen; eine manuelle Verifikation ist nicht möglich

Verifikation von Chiplayouts ist nicht nur nötig, um sicherzustellen, daß Entwurfswerte nicht verfälscht sind, damit diese für Vergleiche verwendet werden können. Auch und gerade dann, wenn ein Chip hergestellt werden soll, ist als letzter Schritt eine Verifikation des Layouts nötig, um sicherstellen zu können, daß der Chip nach der Herstellung auch funktionsfähig ist. Ein Ansatz hierzu wird z. B. in [Yos86] gefordert und beschrieben. Ein fehlerhaftes Layout bei der Chipherstellung führt zu unvertretbaren Kosten, so daß eine Verifikation auf keinem Fall unterbleiben darf.
Im ESPRO-System geht es jedoch nicht darum, herstellungsfähige Layouts zu liefern, sondern die durch verschiedene Heuristiken entstandenen Layouts sollen untereinander verglichen werden können. Dies ist jedoch nur dann möglich, wenn die Entwurfswerte der Layouts nicht verfälscht sind. Auch hier ist es also dringend erforderlich, die Layouts überprüfen zu können, um mehr Sicherheit beim Vergleich von Entwurfswerten zu erreichen. Die Vergleiche werden erst nach einer Verifikation aussagekräftig.

2.2 Aufgabendefinition

Aus der im vorigen Abschnitt dargestellten Motivation heraus resultiert die Aufgabe, ein Verifikationstool für das ESPRO-System zu entwickeln. Dieses Verifikationstool soll dabei folgendes leisten:

- Extraktion der Netzlisten aus einem Layout: Aus der in einem Layout enthaltenen Verdrahtung sollen die Netzlisten wieder rekonstruiert werden. Dieser Vorgang wird auch als Extraktion bezeichnet. Man kann dies auch als Umkehrung des Routing sehen: Während beim Routing aus den Netzlisten ein Layout erzeugt wird, werden hier aus einem Layout wieder die Netzlisten erzeugt.
- Suche von Fehlern in einem Layout: In einem Layout sollen mögliche Fehler (z. B. Kurzschlüsse), die durch Fehler beim Routing verursacht werden können, erkannt werden.
- Vergleich der Originalnetzlisten mit den aus dem Layout extrahierten Netzlisten: um letztendlich feststellen zu können, ob durch ein Layout genau die Verbindungen implementiert werden, die durch die Originalnetzlisten (Basisdatei) beschrieben sind, ist ein Vergleich der extrahierten Netzlisten mit den Originalnetzlisten notwendig.

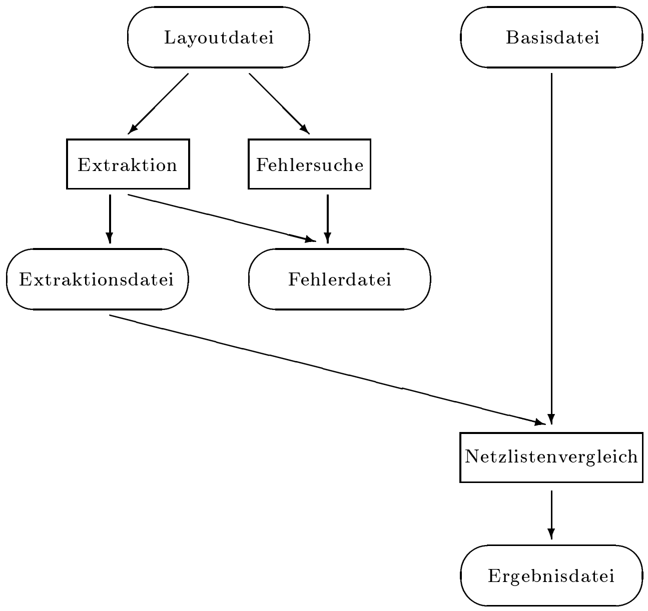

Abbildung 2.2: Überblick über die Struktur des Verifikationstools

Damit ergibt sich für das Verifikationstool eine Struktur, wie in Abbildung 2.2 gezeigt. Ausgehend von den in der Layoutdatei enthaltenen Daten erfolgt zunächst die Extraktion und die Fehlersuche im Layout. Da sowohl die Extraktion als auch die Fehlersuche auf denselben Daten arbeiten, erscheint es sinnvoll, diese in einem gemeinsamen Programm zu implementieren. Das Ergebnis der Extraktion ist eine Extraktionsdatei. In diese Datei sollen die aus dem Layout extrahierten Netzlisten geschrieben werden. Das Ergebnis der Fehlersuche ist eine Datei, in der alle Fehler protokolliert werden. Falls schon bei der Extraktion Fehler auftreten, sollen auch diese in die Fehlerdatei geschrieben werden. Nach der Extraktion und der Fehlersuche erfolgt der Netzlistenvergleich. Hierzu ist ein Tool zu entwickeln, das als Eingabe die Basisdatei und die Extraktionsdatei erhält, um die in diesen Dateien enthaltenen Netzlisten zu vergleichen. Bei diesem Vergleich festgestellte Unterschiede sollen in einer Ergebnisdatei gesammelt werden. Die Fehlerdatei und die Ergebnisdatei

enthalten das Ergebnis einer Verifikation. Diese beiden Dateien werden am Ende vom Benutzer selbst ausgewertet.
Für die Entwicklung des Verifikationstools ergibt sich damit folgende Vorgehensweise:

- Analyse möglicher Fehler: Bevor die Entwicklung der Software beginnen kann, muß zunächst analysiert werden, welche Arten von Fehlern in einem Layout auftreten können, und wie sich solche Fehler auswirken. Bei dieser Analyse sollen die Fehler in Klassen eingeteilt werden. Kriterien für diese Klasseneinteilung sind der Zeitpunkt, an dem der Fehler gefunden wird (während der Fehlersuche im Layout oder während des Netzlistenvergleichs) und die Auswirkungen des Fehlers (ob der Fehler zu Funktionsstörungen der Schaltung führen würde oder nicht). Diese Analyse ist die Grundlage für die weitere Entwicklung. Nach der Analyse der möglichen Fehlerarten sind Algorithmen zu entwickeln, um die einzelnen Fehler entdecken zu können.

- Entwicklung eines *Extraktors*: In dem Extraktor sollen die Extraktion der Netzlisten und die Fehlersuche im Layout implementiert werden. Der Extraktor muß also imstande sein, eine Layoutdatei einzulesen, die Extraktion und die Fehlersuche durchzuführen und das Ergebnis in Form einer Extraktionsdatei und einer Fehlerdatei auszugeben. Die Algorithmen sowohl für die Extraktion als auch für die Fehlersuche sind zu entwickeln. Die bisherigen Veröffentlichungen zum Thema Extraktion gehen größtenteils davon aus, daß aus der elektrischen Ebene Schaltungen auf der Registertransfer-Ebene extrahiert werden sollen (z. B. [Hol86, Neb86, Blö88]), um auf die Funktionalität einer Schaltung zu schließen oder um Simulationen durchführen zu können. Dies wird meist durch eine Art Mustererkennung (z. B. beim Scanline-Algorithmus) durchgeführt. Im ESPRO-System wird jedoch nicht die Funktionalität einer Schaltung betrachtet, sondern nur die Verdrahtung der Zellanschlüsse untereinander. Es sollen hier aus einem Layout die Verbindungen erkannt und daraus Netzlisten rekonstruiert werden. Damit ist hier ein anderer Ansatz nötig. Auch die bisher veröffentlichten Algorithmen zur Fehlersuche (z. B. [Yos86]) wurden für den Design-Rule-Check in bestimmten Technologien entwickelt. ESPRO ist jedoch ein technologieunabhängiges System. Für die Extraktion und Fehlersuche sind also eigene Algorithmen zu entwickeln.

- Entwicklung eines *Netzlistenvergleichers*: Hierfür ist ein Programm zu entwickeln, das eine Basisdatei und eine Extraktionsdatei einliest und diese miteinander vergleicht. Das Ergebnis soll in einer Ergebnisdatei ausgegeben werden. Der Algorithmus für den Vergleich ist zu entwickeln. Aufgrund von in den Layoutdateien enthaltenen redundanten Informationen, die hier ausgenutzt werden können, ist die Zuordnung der Netze bekannt und somit trivial. Das NP-vollständige Isomorphieproblem tritt hier also nicht auf. Fr jedes Netz aus der Basisdatei ist lediglich zu prüfen, ob das dazugehörige Netz in der Extraktionsdatei die gleichen Anschlüsse enthält, was durch einen einfachen Vergleich möglich ist.

- Evaluierung des Tools: Nachdem das Verifikationstool entwickelt worden ist, soll eine Evaluierung erfolgen. Dazu werden möglichst viele Layouts herangezogen, die mit den bisher für das ESPRO-System verfügbaren Routern und Nachoptimierern erstellt worden sind.

Die Analyse der einzelnen Fehlerarten erfolgt im Kapitel 3. Darauf aufbauend werden im Kapitel 4 die Algorithmen zur Extraktion, zur Fehlersuche und zum Netzlistenvergleich entwickelt. Die Implementierung dieser Algorithmen wird im Kapitel 5 beschrieben. Im Kapitel 6 schließlich erfolgt die Evaluierung des Verifikationstools anhand von Beispiellayouts.

2.3 Grundlegende Definitionen

In diesem Abschnitt werden alle Fachbegriffe eingeführt und definiert, die in dieser Arbeit verwendet werden. Die meisten dieser Begriffe werden so auch in anderen Veröffentlichungen zu Placement und Routing verwendet, einige Begriffe sind jedoch spezifisch für das ESPRO-System.

Zelle: Eine Zelle ist eine aktive Funktionseinheit auf einem Chip. In ESPRO werden Zellen als Black Boxes betrachtet, d. h. es kommt auf die Funktion einer Zelle nicht an, die Zelle wird rein geometrisch betrachtet. Eine Zelle ist damit ein Rechteck mit einer festgelegten Höhe und einer Breite, die sich aus der Anzahl der Anschlüsse ergibt. Die Anschlüsse einer Zelle werden auch als **Pins** bezeichnet. Eine Zelle wird in ESPRO definiert durch zwei gegenüberliegende Eckpunkte. Aus der Breite kann die Anzahl der Pins errechnet werden. Die Anschlüsse sind gleichmäßig auf den oberen und den unteren Rand der Zelle verteilt.

Pad: Ein Pad ist ein äußerer Anschluß eines Chips. Pads haben eine feste Größe und sind quadratisch. Die Position wird durch die Koordinaten des Mittelpunktes bestimmt. In einigen Arbeiten zu ESPRO werden Pads auch als Ports bezeichnet, was dazu führt, daß diese Begriffe hier synonym verwendet werden.

Netzliste: Eine Netzliste ist eine Liste, die alle Pads und Pins enthält, die in einem Netz miteinander verbunden werden sollen. Die Elemente einer Netzliste werden auch als **Netznachbarn** oder **Nachbarn** bezeichnet.

Layer: Im ESPRO-System hat jeder Chip zwei voneinander isolierte Verdrahtungsebenen. Diese Verdrahtungsebenen werden als Layer bezeichnet.

Segment: Die Verdrahtungsleitungen werden durch Segmente gebildet. Segmente verlaufen immer entweder in horizontaler oder in vertikaler Richtung, im ersten oder im zweiten Layer und haben eine einheitliche Breite. Ein Segment wird damit definiert durch die Koordinaten des Anfangs- sowie des Endpunktes, den Typ (in horizontaler oder in vertikaler Richtung verlaufend), den Layer, in dem das Segment liegt (Layer 1 oder Layer 2) und den Index des Netzes, zu dem das Segment gehören soll.

Kontaktloch, Via: Ein Kontaktloch oder Via stellt eine Verbindung zwischen zwei Segmenten dar, die in unterschiedlichen Layern liegen. Ein Kontaktloch hat eine feste Größe und wird definiert durch die Koordinaten des Mittelpunktes. Die Begriffe Kontaktloch und Via werden synonym verwendet.

Kapitel 3

Fehlerklassen

In diesem Kapitel wird beschrieben, welche Arten von Fehlern in ESPRO-Standardzell-Layouts auftreten können. Dabei werden die Fehler in eine Klassenhierarchie eingeteilt. Es gibt zwei Oberklassen: Fehler, die beim Extrahieren festgestellt werden, und Fehler, die beim Vergleich der extrahierten Netzlisten mit den Originalnetzlisten (also nach dem Extrahieren) festgestellt werden. Beim Extrahieren gibt es außerdem die Unterklassen „Schwere Fehler“ und „unechte Fehler“. Schwere Fehler würden bei einem realen Chip zu Funktionsstörungen oder Nichtfunktion der Schaltung führen, während unechte Fehler zu einem unschönen und ineffizienten Layout, aber nicht zu Funktionsstörungen führen.
Beim Netzlistenvergleich gibt es diese Unterklassen nicht, da alle hier auftretenden Fehler schwere Fehler sind, also zu Funktionsstörungen führen würden.
Die Klassenhierarchie sieht damit folgendermaßen aus:

1. Fehler, die beim Extrahieren festgestellt werden
   (a) Schwere Fehler
      - Kurzschlüsse zwischen verschiedenen Netzen
      - Unterbrochene Netze
      - Verletzung der Design Rules (Verbot von Routing over the Cell)

   (b) Unechte Fehler
      - Offene Segmentenden
      - Überflüssige Segmente
      - Überlange Segmente
      - Überflüssige Teilnetze
      - Kreise
      - Überflüssige Kontaktlöcher
      - Äquivalente Kontaktlöcher
2. Fehler, die beim Netzlistenvergleich festgestellt werden
   - Fehlende Netznachbarn
   - Überzählige Netznachbarn

Im folgenden werden alle in dieser Hierarchie genannten Fehler näher beschrieben.

3.1 Schwere Fehler beim Extrahieren

Treten in einem Layout schwere Fehler auf, führt dies grundsätzlich zu Funktionsstörungen der Schaltung. Würde ein solcher Chip hergestellt werden, würde dies zu unvertretbaren Kosten führen, da dieser nicht funktionsfähig wäre.

3.1.1 Kurzschlüsse

Kurzschlüsse führen dazu, daß zwei oder mehrere Netze, die eigentlich voneinander getrennt sein müßten, miteinander verbunden werden und damit physikalisch gesehen zu einem Netz verschmelzen. Dies führt grundsätzlich immer zu Funktionsstörungen. Kurzschlüsse können auf drei Arten verursacht werden:

- durch Kontaktlöcher, die zwei Segmente verschiedener Netze miteinander verbinden (in Abbildung 3.1 mit (a) bezeichnet)
- durch Zellanschlüsse (Pins) und Pads, die an zwei Netze angeschlossen worden sind (in Abbildung 3.1 mit (b) bezeichnet)
- durch Segmente verschiedener Netze, die sich im selben Layer überlappen (in Abbildung 3.1 mit (c) bezeichnet)

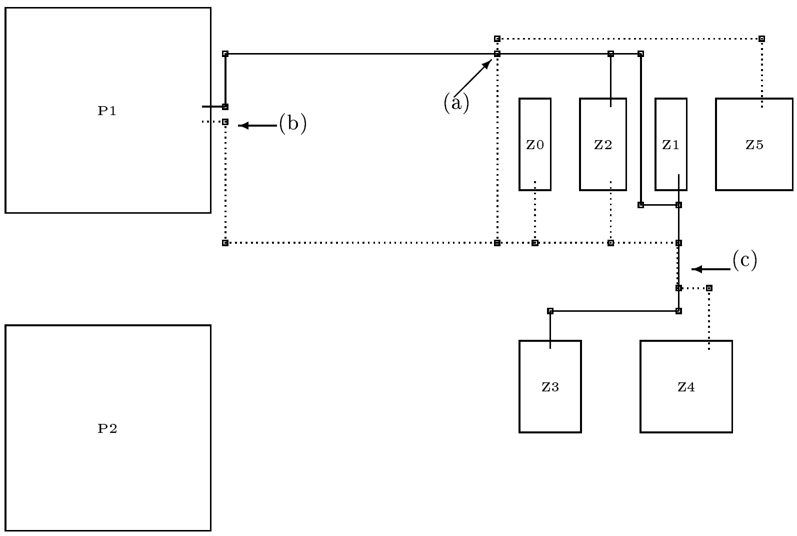

Abbildung 3.1: Beispiele für Kurzschlüsse (durchgezogene Linien: Netz 1, gepunktete Linien: Netz 2; senkrechte Linien: Layer 1, waagerechte Linien: Layer 2)

3.1.2 Unterbrochene Netze

Dieser Fehler führt dazu, daß ein Netz in zwei oder mehrere Teilnetze auseinanderfällt. Diese Teilnetze sind untereinander nicht miteinander verbunden, obwohl dies eigentlich der Fall sein sollte. Auch dieser Fehler führt immer zu Funktionsstörungen und kann durch fehlende Segmente oder fehlende Kontaktlöcher verursacht werden. Ein Beispiel für ein solches Netz ist in Abbildung 3.2 zu sehen.

3.1.3 Verletzung der Design Rules

Die einzige Design Rule in ESPRO ist, daß Segmente nicht über Zellen gelegt werden dürfen (generelles Verbot von Routing over the Cell). Andere Regeln gibt es in ESPRO nicht, da keine Technologieabhängigkeiten vorhanden sind. Eine Verletzung dieser Regel ist ein schwerer Fehler, da die Herstellung des Chips damit unmöglich gemacht werden würde.

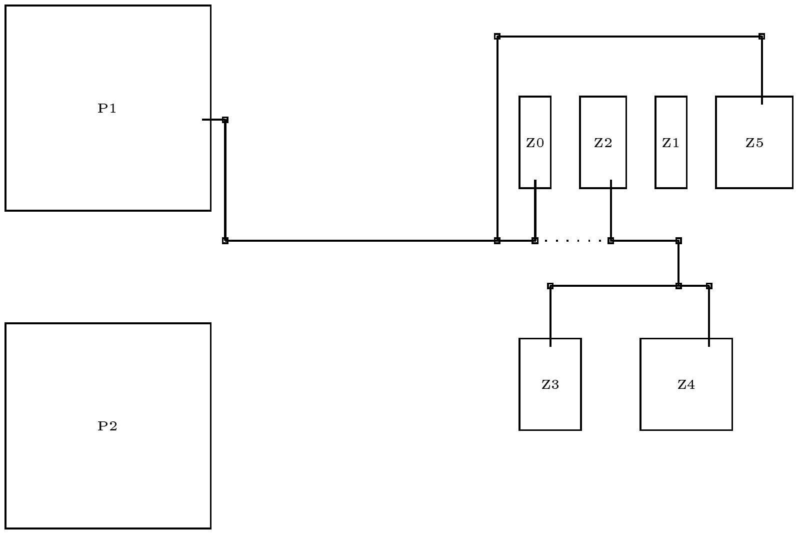

Abbildung 3.2: Beispiel für ein unterbrochenes Netz: Die gepunktete Verbindung fehlt, so daß das Netz in zwei nicht verbundene Teilnetze zerfällt

3.2 Unechte Fehler beim Extrahieren

Treten unechte Fehler allein auf, kann dies nicht zu Funktionsstörungen der Schaltung führen. Unechte Fehler führen jedoch meistens zu Platzverschwendung im Layout und damit zu erhöhten Herstellungskosten und unnötig langen Leitungen bei anderen Netzen, da Flächen blockiert werden, die sonst für andere Leitungen genutzt werden könnten. Außerdem sind solche Fehler immer ein Indiz für Programmierfehler im Router oder im Nachoptimierer.
Unechte Fehler können jedoch auch als Folgefehler von schweren Fehlern (s. 3.1) auftreten.

3.2.1 Offene Segmentenden

Wenn ein Ende eines Segments nicht mit einem anderen Segment, einem Pin oder einem Pad verbunden ist, wird dieses Segmentende als offen bezeichnet. Dieser Fehler allein führt nur zu Platzverschwendung im Layout, kann aber auch ein durch ein unterbrochenes Netz (s. 3.1.2) ausgelöster Folgefehler sein. Außerdem kann es sich hierbei auch um ein überlanges Segment handeln. Beispiele hierfür sind die Segmente (a) und (d) in Abbildung 3.3.

3.2.2 Überflüssige Segmente

Wenn beide Enden eines Segments offen sind und dieses Segment auch nicht an anderen Stellen durch Kontaktlöcher oder Kreuzung im selben Layer mit anderen Segmenten verbunden ist, so wird dieses Segment als überflüssig bezeichnet. Dies führt zu Platzverschwendung im Layout, kann aber auch ein durch ein unterbrochenes Netz (s. 3.1.2) ausgelöster Folgefehler sein. Ein Beispiel für ein solches Segment ist das in Abbildung 3.3 mit (b) bezeichnete Segment.

3.2.3 Überlange Segmente

Ist ein Ende eines Segments offen, das Segment aber noch an anderen Stellen mit anderen Segmenten verbunden, so wird dieses Segment als überlang bezeichnet. Dies führt zu Platzverschwendung im Layout. Außerdem kann es sich um einen durch ein unterbrochenes Netz (s. 3.1.2) ausgelösten Folgefehler handeln. Ein überlanges Segment hat entweder ein (Segment (c) in Abbildung 3.3) oder zwei offene Enden (Segment (d) in Abbildung 3.3).

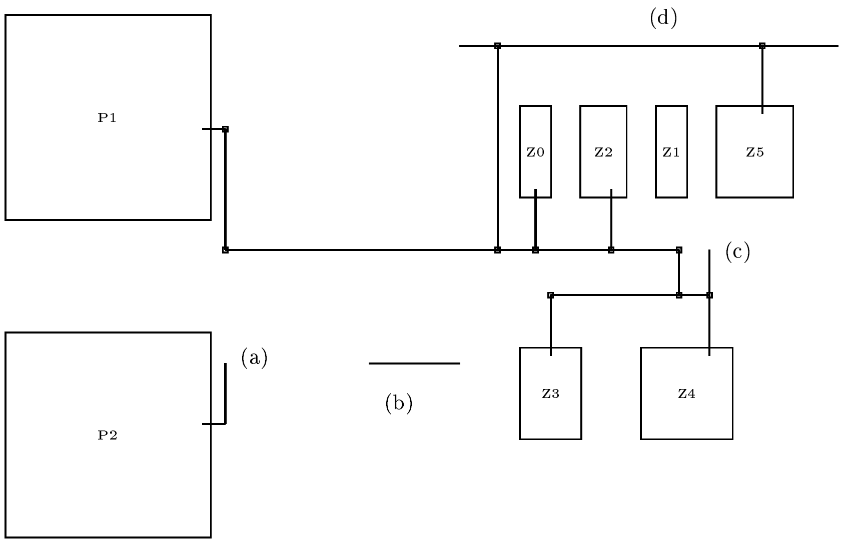

Abbildung 3.3: Beispiele für: Offene Enden (a), überflüssige (b) und überlange Segmente (c, d)

3.2.4 Überflüssige Teilnetze

Überflüssige Teilnetze entstehen, wenn mehrere Segmente eines Netzes zwar miteinander verbunden sind, aber keines dieser Segmente mit einem Pin oder Pad verbunden ist. Dieser Fehler führt zu keinen Funktionsstörungen, verschwendet aber viel Platz im Layout. Ein Beispiel dafür befindet sich in Abbildung 3.4.

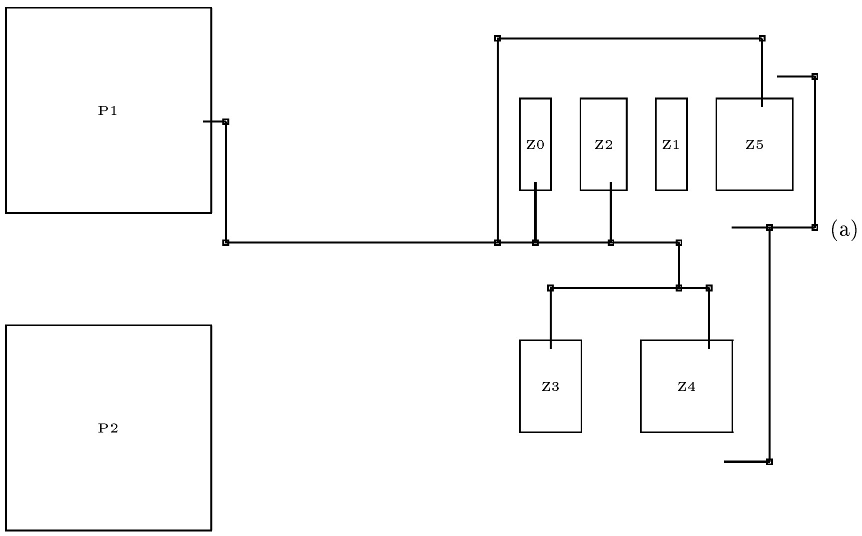

Abbildung 3.4: Das Teilnetz (a) ist überflüssig

3.2.5 Kreise

Ein Kreis ensteht, wenn zwei Netznachbarn auf zwei oder mehreren Wegen miteinander verbunden sind. Auch ein Kreis führt nicht zu Funktionsstörungen der Schaltung, verschwendet jedoch wertvollen Platz im Layout. Es kann sich jedoch nicht um einen Folgefehler handeln. Ein Beispiel für einen Kreis ist in Abbildung 3.5 zu sehen.

3.2.6 Überflüssige Kontaktlöcher

Ein Kontaktloch ist genau dann überflüssig, wenn durch dieses Kontaktloch nicht mindestens zwei in verschiedenen Layern liegende Segmente miteinander verbunden werden. Solche Fehler würden bei der Herstellung eines Chips zu unnötig hohen Kosten führen. Beispiele hierfür sind in Abbildung 3.5 enthalten.

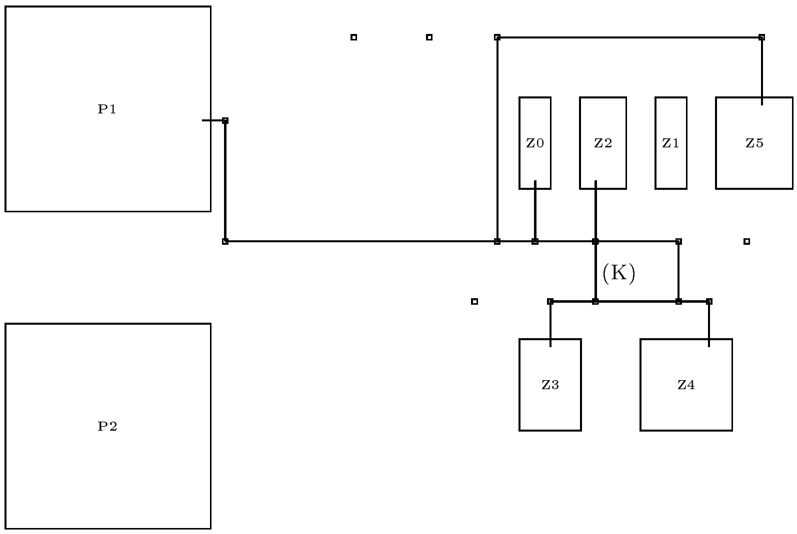

Abbildung 3.5: Kreis (K) sowie einige überflüssige Kontaktlöcher

3.2.7 Äquivalente Kontaktlöcher

Zwei Kontaktlöcher sind genau dann äquivalent, wenn ihre Koordinaten gleich sind. Dieser Fehler ist im Layout nicht sichtbar, kann jedoch auf mögliche Fehler in der Speicherverwaltung oder in der Layout-Ausgaberoutine des Routers oder des Nachoptimierers hindeuten.

3.3 Fehler beim Netzlistenvergleich

Auch wenn beim Extrahieren keiner der in den Abschnitten 3.1 und 3.2 beschriebenen Fehler aufgetreten ist, müssen die extrahierten Netzlisten nicht notwendigerweise mit den Originalnetzlisten übereinstimmen. Hat der Router einen Netznachbarn nicht mit dem Netz verbunden oder einen Netznachbarn mit einem falschen Netz verbunden, jedoch keine der in 3.1 und 3.2 beschriebenen Layoutfehler erzeugt, entsprechen eine oder mehrere der extrahierten Netzlisten nicht den Originalnetzlisten. Solche Fehler können jedoch erst durch einen nach dem Extrahieren durchgeführten Netzlistenvergleich erkannt werden.

3.3.1 Fehlende Nachbarn

Wenn in der Originalnetzliste Netznachbarn vorhanden sind, die in der dazugehörigen extrahierten Netzliste nicht vorhanden sind, so werden diese Netznachbarn als fehlende Nachbarn bezeichnet. Solche Fehler führen in jedem Fall zu Funktionsstörungen oder zur Nichtfunktion der Schaltung.

3.3.2 Überzählige Nachbarn

Sind hingegen in der extrahierten Netzliste Netznachbarn vorhanden, die in der Originalnetzliste nicht vorhanden sind, so werden diese Netznachbarn als überzählige Nachbarn bezeichnet. Auch solche Fehler führen zu Funktionsstörungen oder zur Nichtfunktion der Schaltung.

Kapitel 4

Algorithmen zur Extraktion und Fehlersuche

In diesem Kapitel werden die in der im Rahmen dieser Diplomarbeit entwickelten Software verwendeten Algorithmen beschrieben. Es werden dabei zunächst die bei der Extraktion und dann die bei der anschlieenden Fehlersuche verwendeten Algorithmen beschrieben. Außerdem wird der beim Netzlistenvergleich verwendete Algorithmus beschrieben.

4.1 Aufbereitung des Layouts

Bevor die in diesem Kapitel beschriebenen Algorithmen auf ein Layout angewendet werden, ist eine Aufbereitung des Layouts ntig. Hierbei muß das Layout in eine für die Extraktions- und Fehlersuchalgorithmen günstige Form gebracht werden.
Der wichtigste Vorgang bei der Aufbereitung des Layouts ist das Normalisieren der Segmente. Dies ist im folgenden Abschnitt beschrieben.

4.1.1 Normalisieren der Segmente

In ESPRO muß ein Segment nicht notwendigerweise eine Verbindung zwischen zwei Knoten (Kontaktloch/Zellanschluß/Pad) der Schaltung darstellen. Vielmehr verläuft ein Segment oft durch mehrere Kontaktlcher und verbindet somit mehrere Knoten der Schaltung (Abb. 4.1). Fr die Entwicklung und insbesondere die Implementierung der im folgenden beschriebenen Algorithmen hat es sich jedoch als sehr vorteilhaft erwiesen, wenn jedes Segment hchstens zwei Knoten der Schaltung miteinander verbindet.
Dies wird dadurch erreicht, daß fr jedes Segment, das mehr als zwei Knoten verbindet, Segmente erzeugt werden, die jeweils zwei Knoten verbinden. Dieser Vorgang soll als *Normalisieren der Segmente* und die dabei erzeugten Segmente als *normalisierte Segmente* bezeichnet werden. Die Segmente, aus denen die normalisierten Segmente erzeugt werden, werden im folgenden als *Originalsegmente* bezeichnet. Für ein Originalsegment, das n Knoten (mit $n > 2$) verbindet, werden $n - 1$ normalisierte Segmente erzeugt. Auch für Originalsegmente, die $n \leq 2$ Knoten verbinden, wird je ein normalisiertes Segment erzeugt, das jedoch identisch mit dem Originalsegment ist (Abb. 4.2).
Die meisten der folgenden Algorithmen arbeiten mit den so erzeugten normalisierten Segmenten, einige jedoch auch mit den Originalsegmenten.

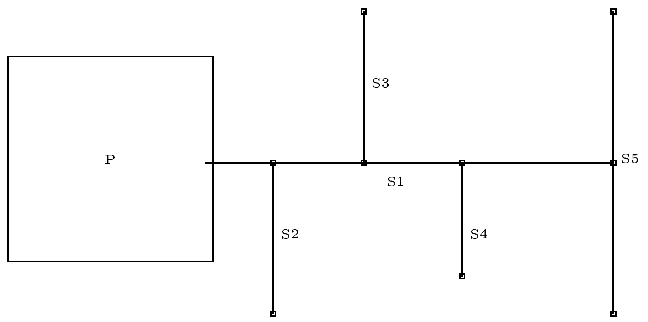

Abbildung 4.1: Vor der Normalisierung: Die Originalsegmente S1 und S5 verbinden jeweils mehr als zwei Knoten.

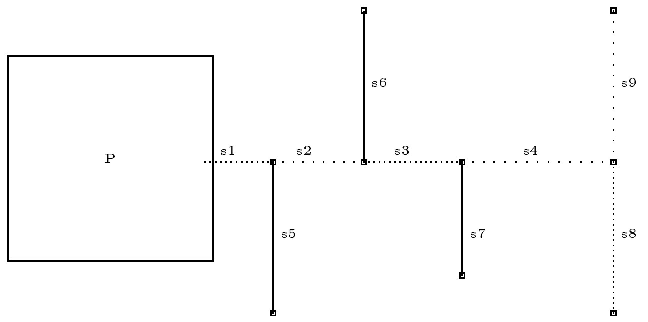

Abbildung 4.2: Nach der Normalisierung: Jedes Segment verbindet höchstens zwei Knoten (unterschiedliche Segmente durch unterschiedliche Strichlierung angedeutet). Das Originalsegment S1 aus Abb. 4.1 wurde in die normalisierten Segmente s1, s2, s3 und s4, das Originalsegment S5 in die normalisierten Segmente s8 und s9 aufgeteilt.

4.2 Extraktion

Nachdem das Layout aufbereitet worden ist, kann die Extraktion der Netzlisten erfolgen. Während des Extrahierens werden außerdem Informationen gewonnen, die für die weiteren Fehlersuchalgorithmen benötigt werden. Nach einigen Vorbemerkungen soll der Extraktionsalgorithmus zunächst grob betrachtet werden. Danach werden einige ausgewählte Schritte verfeinert betrachtet.

4.2.1 Vorbemerkungen

In einer ESPRO-Layoutdatei ist jedes Segment mit der Nummer des Netzes markiert, zu dem das Segment gehören soll. Diese Netznummern werden beim Normalisieren der Segmente (s. 4.1.1) auch auf die normalisierten Segmente übertragen. Dies sind eigentlich redundante Informationen, die jedoch für die Extraktion gut ausgenutzt werden können, da damit bekannt ist, welche Segmente verbunden sein müssen und welche nicht. Damit kann das Problem der Extraktion, welches im allgemeinen als NP-schwer gilt, auf ein Problem reduziert werden, das in P enthalten ist, da lediglich für jedes Netz zu prüfen ist, ob die Segmente, die mit der Nummer dieses Netzes markiert sind, miteinander verbunden sind und welche Anschlüsse an diesen Segmenten hängen. Auch für einen späteren Netzlistenvergleich ist diese Information vorteilhaft, da damit auch die Zuordnung der Originalnetzlisten zu den extrahierten Netzlisten bekannt ist, so daß das sonst bei Netzlistenvergleichen auftretende, NP-vollständige Isomorphieproblem hier nicht auftritt. Der Netzlistenvergleich wird hier ebenfalls auf ein in P enthaltenes, lineares Problem reduziert.

4.2.2 Der Algorithmus (grober Ablauf)

Fr die eigentliche Extraktion erhalten die Segmente (und auch die normalisierten Segmente) noch eine weitere Markierung, die im folgenden Teilnetznummer genannt wird. Außerdem gilt folgende Vereinbarung: Segmente oder normalisierte Segmente mit gleicher Netznummer und gleicher Teilnetznummer sind elektrisch leitend miteinander verbunden.
Die Extraktion kann nun netzweise erfolgen. Dabei werden jeweils nur die Segmente betrachtet, die mit der Netznummer des gerade zu extrahierenden Netzes markiert sind. Um ein Netz zu extrahieren, sind die folgenden Schritte nötig:

1. Initialisieren der Teilnetznummern: Alle zu diesem Netz gehörenden normalisierten Segmente erhalten unterschiedliche Teilnetznummern, beginnend mit der Teilnetznummer 1. Normalisierte Segmente, die zu demselben Originalsegment gehören, erhalten gleiche Teilnetznummern, da für diese normalisierten Segmente bereits bekannt ist, daß sie miteinander verbunden sind. Ein Beispiel ist in Abbildung 4.3 dargestellt.

2. Suchen von Verbindungen von Layer zu Layer: Wenn es in beiden Layern normalisierte Segmente gibt, die zu diesem Netz gehören, wird zunächst nach Verbindungen gesucht, die von einem Segment des einen Layer durch ein Kontaktloch zu einem Segment des anderen Layer verlaufen. Da hierbei nur mit normalisierten Segmenten gearbeitet wird, brauchen nur die Enden der Segmente betrachtet zu werden. Wurde eine Verbindung zwischen normalisierten Segmenten mit unterschiedlicher Teilnetznummer gefunden, werden alle Segmente mit der größeren Teilnetznummer auf die kleinere Teilnetznummer gesetzt. Dies ist in Abbildung 4.4 dargestellt und wird im nächsten Abschnitt im Detail beschrieben.

3. Haben jetzt alle normalisierten Segmente die gleiche Teilnetznummer, sind alle Segmente dieses Netzes miteinander verbunden. Weiter bei Schritt 6.

4. Suchen von Verbindungen innerhalb eines Layers: Ähnlich wie bei Schritt 2 wird für jeden der beiden Layer geprüft, ob es innerhalb dieses Layers Verbindungen zwischen Segmenten dieses Netzes gibt. Wurde eine Verbindung zwischen normalisierten Segmenten mit unterschiedlicher Teilnetznummer gefunden, werden wieder alle normalisierten Segmente mit der größeren Teilnetznummer auf die kleinere Teilnetznummer gesetzt. Dieser Vorgang wird im nächsten Abschnitt detailliert beschrieben.

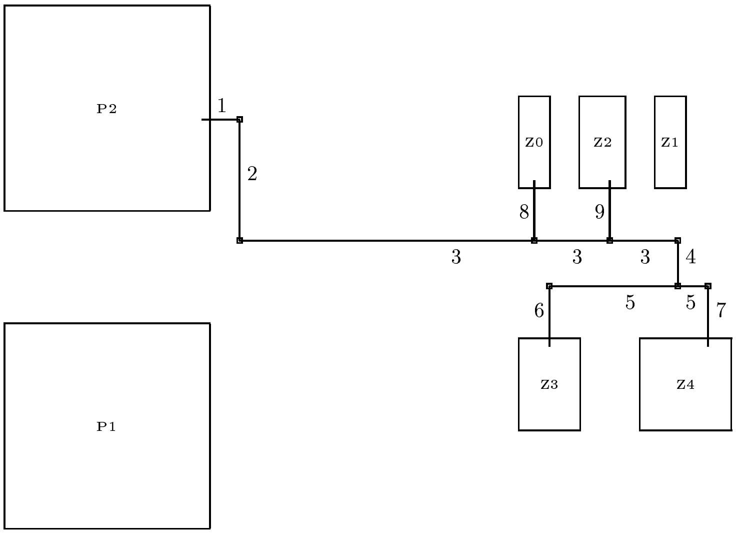

Abbildung 4.3: Die Teilnetznummern nach der Initialisierung

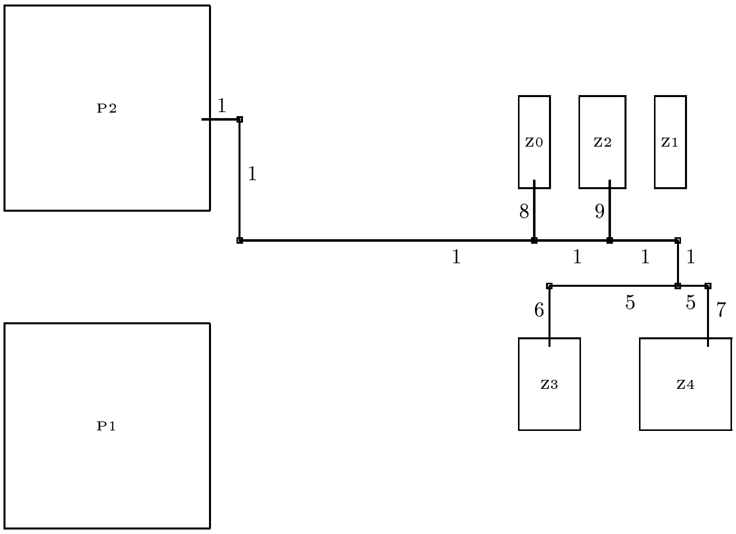

Abbildung 4.4: Eigentliche Extraktion nach vier Schritten

5. Haben jetzt alle normalisierten Segmente die gleiche Teilnetznummer, sind alle Segmente dieses Netzes miteinander verbunden. Andernfalls gibt es mehrere nicht miteinander verbundene Teilnetze.
6. bertragen der Teilnetznummern von den normalisierten Segmenten auf die Originalsegmente
7. Rekonstruieren der Netzliste: Für jedes Originalsegment dieses Netzes wird geprüft, ob dieses in einem Zellanschluß (Pin) oder einem Pad beginnt oder endet. Ist dies der Fall, wird dieser Pin oder Pad in die Netzliste dieses Netzes aufgenommen. Dies ist in Abbildung 4.5 dargestellt.

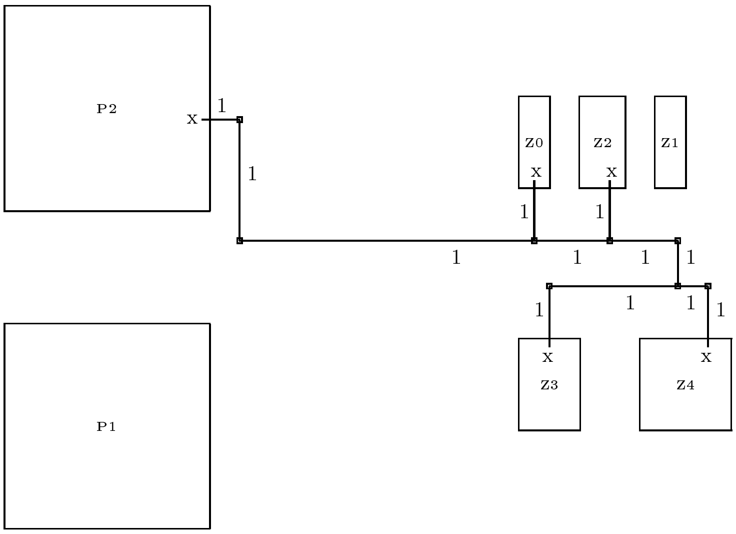

Abbildung 4.5: Vollständig durchgeführte Extraktion. Die mit x markierten Anschlüsse werden in die extrahierte Netzliste aufgenommen.

Diese Schritte müssen für jedes Netz genau einmal ausgeführt werden. Wichtig ist, daß hier jeweils nur nach Verbindungen zwischen Segmenten gesucht wird, die mit der Netznummer des gerade betrachteten Netzes markiert sind. Kurzschlüsse zwischen Segmenten verschiedener Netze werden bei der späteren Fehlersuche gesucht.

4.2.3 Verfeinerte Beschreibung ausgewählter Teile

In diesem Abschnitt sollen die Schritte des Algorithmus, die selbst wieder einen größeren Algorithmus enthalten, näher beschrieben werden. Dies sind insbesondere die Schritte 2 („Verbindungen von Layer zu Layer“) und 4 („Verbindungen innerhalb eines Layers“).

Verbindungen von Layer zu Layer

Bei Verbindungen von Layer zu Layer handelt es sich um Verbindungen, die von einem Segment im ersten Layer über ein Kontaktloch zu einem Segment im zweiten Layer (oder umgekehrt) verlaufen. Da sich bei normalisierten Segmenten nur an deren Anfangs- und Endpunkten Kontaktlöcher befinden können, brauchen auch nur die Anfangs- und Endpunkte der normalisierten Segmente betrachtet zu werden. Der Algorithmus zum Suchen von Verbindungen von Layer zu Layer ist damit folgender:

1. Setze S_{L1} := Menge aller normalisierten Segmente aus Layer 1, die mit der Netznummer des betrachteten Netzes markiert sind.

2. Setze S_{L2} := Menge aller normalisierten Segmente aus Layer 2, die mit der Netznummer des betrachteten Netzes markiert sind.

3. Für jedes Segment $s_1 \in S_{L1}$ führe folgende Schritte aus:

   (a) Setze Menge $M := \emptyset$.

   (b) Suche alle normalisierten Segmente aus S_{L2}, bei denen die Koordinaten des Anfangs- oder Endpunktes mit den Koordinaten des Anfangs- oder Endpunktes von s_1 übereinstimmen, und füge diese in die Menge M ein. Prüfe auch, ob sich an der jeweiligen Stelle ein Kontaktloch befindet.

   (c) Setze t_{min} := die kleinste Teilnetznummer der in M enthaltenen Segmente; ist die Teilnetznummer von s_1 kleiner als t_{min}, setze t_{min} := Teilnetznummer von s_1.

   (d) Für jedes Segment $s_2 \in M$: Wenn t_{min} kleiner ist als die Teilnetznummer von s_2, setze die Teilnetznummern aller Segmente in S_{L2}, die die Teilnetznummer von s_2 haben, auf t_{min}.

   (e) Ist die Teilnetznummer von s_1 nicht t_{min}, setze die Teilnetznummern aller Segmente in S_{L1}, die die Teilnetznummer von s_1 haben, auf t_{min}.

Nun haben alle Segmente dieses Netzes, die über Kontaktlöcher miteinander verbunden sind, die gleiche Teilnetznummer.

Verbindungen innerhalb eines Layers

Das Suchen von Verbindungen innerhalb eines Layers geschieht ähnlich wie das Suchen von Verbindungen zwischen den Layern. Die Segmente sind jedoch nicht durch Kontaktlöcher verbunden. Vielmehr sind zwei Segmente im gleichen Layer genau dann miteinander verbunden, wenn diese entweder an ihren Anfangs- bzw. Endpunkten aneinandertreffen oder sich berlappen, oder aber wenn ein in x-Richtung verlaufendes Segment ein in y-Richtung verlaufendes Segment an beliebiger Stelle kreuzt. Um Verbindungen zu suchen, reicht es also nicht aus, nur die Anfangs- bzw. Endpunkte zu betrachten, sondern es sind verschiedene Fälle zu unterscheiden. Dann ist es am einfachsten, für jeden dieser Fälle eine Bedingung anzugegeben, die genau dann erfüllt ist, wenn eine Verbindung zwischen den beiden Segmenten besteht.
Für die folgende Aufstellung aller möglichen Fälle mit ihrer jeweiligen „Verbindungsbedingung" gelten folgende Vereinbarungen:

S_{fc}	bei x-Segmenten x-Koordinate, bei y-Segmenten y-Koordinate des Anfangspunktes
S_{lc}	bei x-Segmenten x-Koordinate, bei y-Segmenten y-Koordinate des Endpunktes
S_t	bei x-Segmenten gemeinsame y-Koordinate, bei y-Segmenten gemeinsame x-Koordinate

Es gilt immer: $S_{fc} \leq S_{lc}$
Es gibt folgende Fälle:

1. Zwei x-Segmente oder zwei y-Segmente, die an ihren Anfangs- bzw. Endpunkten aneindertreffen oder sich überlappen. Dieser Fall ist der einfachste, die Bedingung hierfür lautet:

$$\begin{aligned} S1_t = S2_t \quad \wedge \quad & ((S2_{fc} \leq S1_{fc} \wedge S1_{fc} \leq S2_{lc}) \vee \\ & (S2_{fc} \leq S1_{lc} \wedge S1_{lc} \leq S2_{lc}) \vee \\ & (S1_{fc} \leq S2_{fc} \wedge S2_{fc} \leq S1_{lc}) \vee \\ & (S1_{fc} \leq S2_{lc} \wedge S2_{lc} \leq S1_{lc})) \end{aligned} \tag{4.1}$$

   Die gemeinsamen x-Koordinaten bzw. die gemeinsamen y-Koordinaten beider Segmente müssen also gleich sein, was durch die Bedingung auf der linken Seite besagt wird. Von den Bedingungen auf der rechten Seite muß nur eine erfüllt sein; diese geben die möglichen Fälle der Segmentüberlappung an, welche in Abbildung 4.6 dargestellt sind.

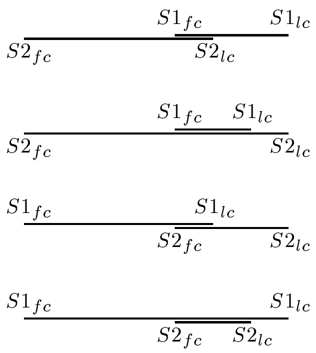

Abbildung 4.6: Mögliche Fälle bei Überlappungen von Segmenten gleichen Typs

2. Ein x-Segment, das ein y-Segment an beliebiger Stelle kreuzt. Hier gibt es verschiedene Unterfälle, die unterschieden werden sollen. Eine Bedingung für alle diese Unterfälle lautet:

$$S1_{fc} \leq S2_t \wedge S2_t \leq S1_{lc} \wedge S2_{fc} \leq S1_t \wedge S1_t \leq S2_{lc} \tag{4.2}$$

Es gibt nun folgende Unterfälle:

(a) Das Segment S1 beginnt an einer beliebigen Stelle des Segments S2. Die Bedingung hierfür lautet:

$$S1_{fc} = S2_t \tag{4.3}$$

(b) Das Segment S1 endet an einer beliebigen Stelle des Segments S2. Die Bedingung hierfür lautet:

$$S1_{lc} = S2_t \tag{4.4}$$

(c) Das Segment S1 kreuzt das Segment S2 an einer beliebigen Stelle. In diesem Fall lautet die Bedingung:

$$S1_{fc} \neq S2_t \wedge S1_{lc} \neq S2_t \tag{4.5}$$

Diese Unterfälle sind in Abbildung 4.7 dargestellt. Die Unterscheidung dieser Unterfälle ist nur notwendig, da die Segmente mit Informationen darber versehen werden sollen, auf welche Weise sie mit anderen Segmenten verbunden sind. Sollten nur Verbindungen gesucht werden, wäre eine Unterscheidung der Unterfälle nicht nötig.

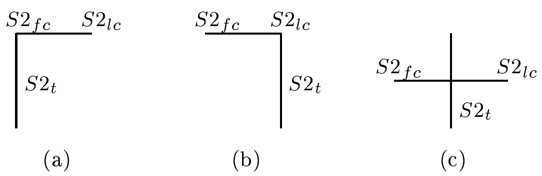

Abbildung 4.7: Mögliche Fälle bei Segmenten verschiedenen Typs

Der eigentliche Algorithmus zum Suchen von Verbindungen innerhalb eines Layers ist nun dem Algorithmus zum Suchen von Verbindungen zwischen den Layern sehr ähnlich. Es wird jedoch nur in einem Layer gesucht. Der Algorithmus wird zweimal durchlaufen, wobei beim ersten Durchlauf der erste und beim zweiten Durchlauf der zweite Layer betrachtet wird. Es wird außerdem vorausgesetzt, daß entweder vorher schon nach Verbindungen zwischen den Layern gesucht wurde oder sich nur in einem Layer Segmente befinden, die mit der Nummer des gerade betrachteten Netzes markiert sind. Der Algorithmus ist also folgender:

1. Setze S_{L1} := Menge aller normalisierten Segmente aus dem betrachteten Layer, die mit der Netznummer des betrachteten Netzes markiert sind.

2. Setze S_{L2} := Menge aller normalisierten Segmente aus dem anderen Layer, die mit der Netznummer des betrachteten Netzes markiert sind.

3. Für jedes Segment $s_1 \in S_{L1}$ führe folgende Schritte aus:

   (a) Setze Menge $M := \emptyset$.

   (b) Suche alle normalisierten Segmente aus S_{L1}, die auf eine der zuvor genannten Arten mit s_1 verbunden sind (also eine der Bedingungen für Überlappung oder Kreuzung erfüllen), und füge diese in die Menge M ein.

   (c) Setze t_{min} := die kleinste Teilnetznummer der in M enthaltenen Segmente; ist die Teilnetznummer von s_1 kleiner als t_{min}, setze t_{min} := Teilnetznummer von s_1.

   (d) Für jedes Segment $s_2 \in M$: Wenn t_{min} kleiner ist als die Teilnetznummer von s_2, setze die Teilnetznummern aller Segmente in S_{L1}, die die Teilnetznummer von s_2 haben, auf t_{min}. Setze außerdem die Teilnetznummern aller Segmente in S_{L2}, die die Teilnetznummer von s_2 haben, auf t_{min}.

   (e) Ist die Teilnetznummer von s_1 nicht t_{min}, setze die Teilnetznummern aller Segmente in S_{L1}, die die Teilnetznummer von s_1 haben, auf t_{min}. Setze außerdem die Teilnetznummern aller Segmente in S_{L2}, die die Teilnetznummer von s_1 haben, auf t_{min}.

Nun haben alle Segmente dieses Netzes, die miteinander verbunden sind, die gleiche Teilnetznummer.

4.3 Fehlersuche

Nachdem aus dem Layout die Netzlisten extrahiert worden sind, erfolgt die Suche von Fehlern im Layout. Einige für die Fehlersuche notwendigen Informationen wurden schon während des Extrahierens gewonnen und müssen nun nur noch von den entsprechenden Fehlersuchroutinen ausgewertet werden. Bei anderen Fehlern wiederum gestaltet sich die Suche aufwendiger. In diesem Abschnitt werden alle Algorithmen beschrieben, die zur Suche nach den im Kapitel 3 beschriebenen Fehlern verwendet werden.

4.3.1 Kurzschlüsse

Kurzschlüsse können, wie schon in 3.1.1 beschrieben, auf drei Arten entstehen. Im folgenden wird für jede dieser drei Arten ein Algorithmus angegeben, um Kurzschlüsse zu finden.

Kurzschlüsse durch Kontaktlöcher

Ein Kontaktloch verursacht genau dann einen Kurzschluß, wenn es zwei Segmente verbindet, die zu verschiedenen Netzen gehören. Um solche Kontaktlöcher zu finden, wird folgender Algorithmus verwendet:

1. Suche zu jedem Kontaktloch ein Segment aus dem ersten Layer und ein Segment aus dem zweiten Layer, das durch dieses Kontaktloch verläuft, und markiere das Kontaktloch mit den Netznummern dieser Segmente.

2. Sind diese beiden Netznummern verschieden, so verursacht das Kontaktloch einen Kurzschluß zwischen den beiden Netzen.

Kurzschlüsse an Pins oder Pads

Ist ein Pin oder ein Pad mit mehreren Netzen verbunden, wird dadurch ein Kurzschlu zwischen diesen Netzen verursacht. Dieser Fehler wird schon während des Extrahierens, bei der Rekonstruktion der Netzlisten (s. Abschnitt 4.2.2), festgestellt: Wenn ein Pin oder ein Pad in die Netzliste eines Netzes aufgenommen wird, wird geprüft, ob dieser Pin oder Pad bereits in der Netzliste eines anderen, vorher extrahierten Netzes enthalten ist. Ist dies der Fall, so ist dieser Pin oder Pad mit mehr als einem Netz verbunden, so daß dadurch ein Kurzschluß zwischen diesen Netzen verursacht wird.

Kurzschlüsse an Segmenten

Falls sich zwei Segmente, die mit verschiedenen Netznummern markiert sind, im selben Layer überlappen oder kreuzen, werden dadurch Kurzschlüsse zwischen den entsprechenden Netzen verursacht. Um derartige Fehler zu finden, wird folgender Algorithmus verwendet:

1. Setze $S_{LN} :=$ Menge aller normalisierten Segmente aus dem betrachteten Layer, die mit der Netznummer des betrachteten Netzes markiert sind.

2. Setze $S_{LX} :=$ Menge aller normalisierten Segmente aus dem betrachteten Layer, die *nicht* mit der Netznummer des betrachteten Netzes markiert sind.

3. Für jedes Segment $s_1 \in S_{LN}$: Suche ein Segment $s_2 \in S_{LX}$, das durch Kreuzung oder Überlappung mit s_1 verbunden ist. Gibt es ein solches Segment, so befindet sich ein Kurzschluß zwischen den Netzen, zu denen s_1 und s_2 gehören.

Dieser Algorithmus wird für jedes Netz zweimal durchlaufen; beim ersten Durchlauf wird Layer 1, beim zweiten Durchlauf Layer 2 betrachtet.

4.3.2 Unterbrochene Netze

Die Informationen, die für die Suche nach unterbrochenen Netzen nötig sind, werden schon während des Extrahierens gewonnen: die Teilnetznummern. Ein Netz ist genau dann unterbrochen, wenn es aus mehreren nicht miteinander verbundenen Teilnetzen besteht. Haben alle Segmente, die mit der Netznummer des betrachteten Netzes markiert sind, die gleiche Teilnetznummer, so besteht das Netz nur aus einem einzigen Teilnetz und ist damit auch nicht unterbrochen. Haben jedoch einige Segmente des betrachteten Netzes andere Teilnetznummern, so gibt es nicht verbundene Teilnetze. In diesem Fall ist zu ermitteln, wieviele verschiedene Teilnetznummern es gibt; dies ist die Anzahl der Teilnetze, in die das Netz zerfallen ist.

4.3.3 Verletzung der Design Rules

In ESPRO gibt es nur eine einzige Design Rule: das Verbot von Routing over the Cell. Diese Design Rule ist einfach zu überprüfen: Es muß für jedes Originalsegment geprüft werden, ob dieses die durch eine Zelle belegte Fläche schneidet. Ist dies der Fall, liegt eine Verletzung der Design Rule vor.

4.3.4 Offene Segmentenden

Die Suche nach offenen Enden besteht im wesentlichen in der Auswertung von Informationen, die schon beim Extrahieren gewonnen worden sind: Jedes Segment wurde mit Informationen darüber versehen, ob dessen Ende offen, mit einem Kontaktloch verbunden, mit anderen Segmenten verbunden oder mit einem Pin oder Pad verbunden ist. Auch wenn ein Segment durch Kreuzung im selben Layer mit einem anderen Segment verbunden ist, wurde dies mit einer entsprechenden Information versehen. Diese Informationen geben nun Aufschluß darüber, ob ein Segment an einem oder beiden Enden offen und damit überlang oder überflüssig ist.

Überflüssige Segmente

Ein Segment ist genau dann überflüssig, wenn es keine Verbindungen zu anderen Segmenten hat (s. 3.2.2). Dies ist genau dann der Fall, wenn folgende Bedingungen erfüllt sind:

1. Beide Segmentenden sind offen.
2. Das Segment ist nicht durch Kreuzung im selben Layer mit anderen Segmenten verbunden.
3. Zu dem Originalsegment gibt es nur genau ein normalisiertes Segment; dies besagt, daß das Segment nicht durch Kontaktlöcher an anderen Stellen als den Segmentenden mit anderen Segmenten verbunden ist.

Überlange Segmente

Ein Segment ist genau dann überlang, wenn es zwar mit anderen Segmenten verbunden, jedoch an mindestens einem Ende offen ist. Es müssen also folgende Bedingungen erfüllt sein:

1. Ein oder beide Segmentenden sind offen
2. Das Segment ist durch Kreuzung im selben Layer mit anderen Segmenten verbunden, *oder / und*
3. Zu dem Originalsegment gibt es mindestens zwei normalisierte Segmente; dies besagt, daß das Segment durch Kontaktlöcher an anderen Stellen als den Segmentenden mit anderen Segmenten verbunden ist.

Diese Bedingungen müssen nun an jedem Segment überprüft werden.

4.3.5 Überflüssige Teilnetze

Ein Teilnetz eines unterbrochenen Netzes ist genau dann überflüssig, wenn kein Segment dieses Teilnetzes mit einem Pin oder Pad verbunden ist. Dies ist mit Hilfe der beim Extrahieren gewonnenen Informationen leicht festzustellen: Wenn ein Netz in mehrere Teilnetze zerfallen ist (s. 4.3.2), ist für jedes Teilnetz zu überprüfen, ob mindestens ein Segment mit einem Pin oder Pad verbunden ist. Ist dies nicht der Fall, ist das Teilnetz überflüssig.

4.3.6 Kreise

Um feststellen zu können, ob ein Netz einen Kreis enthält, der durch beide Layer verläuft, kann folgende Bedingung verwendet werden: Ein Netz enthält einen Kreis, wenn die Anzahl der Kontaktlöcher, die je drei normalisierte Segmente miteinander verbinden, größer ist als die Anzahl der Nachbarn dieses Netzes vermindert um zwei. Dabei wird ein Kontaktloch, das mehr als drei normalisierte Segmente miteinander verbindet, in entsprechend viele Kontaktlcher aufgeteilt, die jeweils drei normalisierte Segmente miteinander verbinden (s. Abb. 4.8).

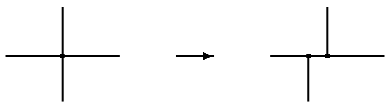

Abbildung 4.8: Aufteilen eines Kontaktloches in zwei Kontaktlöcher

Der Algorithmus ist damit folgender:

1. Setze $n_3 := 0$ (Anzahl der Kontaktlöcher mit drei Segmenten).

2. Für jedes Kontaktloch dieses Netzes: Verbindet dieses Kontaktloch $m \geq 3$ normalisierte Segmente dieses Netzes, so erhöhe n_3 um $m - 2$.

3. Ist nun n_3 größer als die um zwei verminderte Anzahl der Nachbarn dieses Netzes, so enthält das Netz einen Kreis.

Ein Beispiel für ein Netz mit Kreis ist in Abbildung 4.9 dargestellt.

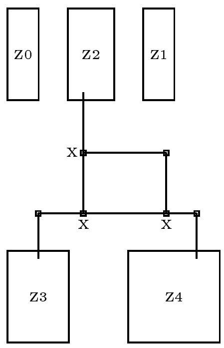

Abbildung 4.9: Das Netz enthält einen Kreis (drei Nachbarn und drei mit x markierte Kontaktlöcher, die je drei Segmente verbinden)

Wenn die Anzahl der Segmente ermittelt wird, die ein Kontaktloch miteinander verbindet, so dürfen dabei offene Segmente nicht mitgezählt werden, da sonst Kreise festgestellt werden, die nicht vorhanden sind. Ein weiterer Nachteil dieses Verfahrens ist, daß damit nur Kreise entdeckt werden können, die durch beide Layer verlaufen. Befindet sich hingegen der Kreis in nur einem Layer, so sind die Segmente, die den Kreis erzeugen, nicht durch Kontaktlöcher verbunden, so daß dieser Algorithmus einen solchen Kreis auch nicht erkennen kann.

4.3.7 Äquivalente Kontaktlöcher

Äquivalente Kontaktlöcher können schon nach dem Einlesen erkannt werden. Nach dem Einlesen der Layoutdatei werden die Kontaktlöcher nach ihren Koordinaten sortiert. Werden dabei äquivalente (also mehrfach mit gleichen Koordinaten vorhandene) Kontaktlöcher erkannt, können diese schon jetzt aus dem Speicher entfernt werden. Äquivalente Kontaktlöcher haben keine weitere Bedeutung für die Verbindungen und könnten sich höchstens störend auf die anderen Algorithmen auswirken.

4.3.8 Überflüssige Kontaktlöcher

Die Informationen, die nötig sind, um überflüssige Kontaktlöcher zu erkennen, werden schon bei der Suche nach Kurzschluß verursachenden Kontaktlöchern geliefert (s. 4.3.1). Ist ein Kontaktloch mit nur einer Netznummer markiert, so gibt es nur in einem Layer ein Segment, das durch dieses Kontaktloch verläuft. Ist ein Kontaktloch mit keiner Netznummer markiert, gibt es in keinem der beiden Layer ein Segment, das durch dieses Kontaktloch verläuft. In beiden Fällen ist das Kontaktloch überflüssig.

4.4 Netzlistenvergleich

Nach der Extraktion der Netzlisten und der Suche von Fehlern im Layout kann der Netzlistenvergleich erfolgen. Hierzu wird das Ergebnis der Extraktion, also die *extrahierten Netzlisten*, benötigt. Außerdem werden die *Originalnetzlisten*, also die Netzlisten, aus denen das Layout entstanden ist, benötigt. Das Ergebnis der Fehlersuche wird für den Netzlistenvergleich jedoch nicht benötigt. Die in den Netzlisten enthaltenen Elemente (Pins oder Pads) werden auch als *Nachbarn* des jeweiligen Netzes bezeichnet. Sind in der Originalnetzliste eines Netzes Nachbarn enthalten, die nicht in der extrahierten Netzliste dieses Netzes enthalten sind, werden diese Nachbarn als *fehlende Nachbarn* bezeichnet. Sind hingegen in der extrahierten Netzliste eines Netzes Nachbarn enthalten, die nicht in der Originalnetzliste dieses Netzes enthalten sind, werden diese Nachbarn als *überzählige Nachbarn* bezeichnet. Eine Originalnetzliste und die dazugehörige extrahierte Netzliste sind genau dann identisch, wenn alle Elemente der Originalnetzliste auch in der extrahierten Netzliste enthalten sind und alle Elemente der extrahierten Netzliste auch in der Originalnetzliste enthalten sind. Da bei ESPRO die Zuordnung der extrahierten Netzlisten zu den Originalnetzlisten bekannt ist (s. 4.2.1), ist der Algorithmus zum Netzlistenvergleich einfach. Es müssen für jedes Netz folgende Schritte durchlaufen werden:

1. Setze $N_O :=$ die Originalnetzliste des betrachteten Netzes
2. Setze $N_E :=$ die extrahierte Netzliste des betrachteten Netzes
3. Für jedes Element $p_o \in N_O$: Prüfe, ob p_o auch in N_E enthalten ist. Wenn nicht, ist p_o ein fehlender Nachbar des betrachteten Netzes.
4. Für jedes Element $p_e \in N_E$: Prüfe, ob p_e auch in N_O enthalten ist. Wenn nicht, ist p_e ein überzähliger Nachbar des betrachteten Netzes.
5. Gibt es weder fehlende noch überzählige Nachbarn, sind N_O und N_E identisch.

Nachdem dieser Algorithmus für jedes Netz ausgeführt wurde, ist für jedes Netz bekannt, ob dessen extrahierte Netzliste mit dessen Originalnetzliste identisch ist. Ist dies bei jedem Netz der Fall, implementiert das Layout genau die durch die Originalnetzlisten beschriebene Schaltung.

Kapitel 5

Implementierung der Algorithmen

In diesem Kapitel wird die Implementierung der im Kapitel 4 entwickelten Algorithmen beschrieben. Ähnlich wie im Kapitel 4 werden auch hier zunächst die Algorithmen zur Extraktion und Fehlersuche und dann der Algorithmus zum Netzlistenvergleich behandelt. Zu jedem Programm werden zunächst die verwendeten Datenstrukturen und danach die Algorithmen beschrieben. Es wird dabei ein Pseudocode verwendet, der der Programmiersprache C ähnlich ist. Die im Rahmen dieser Diplomarbeit entwickelte Software wurde – wie auch alle andere bisher für das ESPRO-System entstandene Software (s. a. [Kla94], [Kas95] und [Kah95]) – in C geschrieben. Es wäre auch denkbar gewesen, eine beliebige andere Programmiersprache zu verwenden, jedoch ermöglicht C die größte Portierbarkeit auf andere Systeme, da C durch den ANSI-Standard weitgehend genormt und auf nahezu allen Systemen verfügbar ist.

5.1 Das Programm EXTRACT

Die in den Kapiteln 4.1 bis 4.3 beschriebenen Algorithmen zur Extraktion und zur Fehlersuche im Layout wurden in dem Programm EXTRACT implementiert. Es werden nun zunächst die wichtigsten Datenstrukturen und Konstanten angegeben, so daß in der darauf folgenden Beschreibung ausgewählter Prozeduren darauf zurückgegriffen werden kann.

5.1.1 Datenstrukturen

Die Datenstrukturen werden in C-Syntax angegeben. Der Basistyp `int` bedeutet dabei eine vorzeichenbehaftete ganze Zahl mit einer Breite von mindestens 32 Bit. Dies entspricht also einem Bereich von mindestens −2.147.483.648 bis +2.147.483.647. Die verwendeten Konstanten sind in der Tabelle 5.1 zusammengefaßt.
Die aus der Layoutdatei eingelesenen Zellen werden im Array `cell_array` abgelegt. Die Elemente dieses Arrays sind Strukturen des Typs `TCell`, der wie folgt definiert ist:

```
typedef struct
{
  int index;
  int degree;
  int x1, y1, x2, y2;
  int cell_pin_index;
} TCell;
```

Dabei enthält `index` die Ordnungsnummer der Zelle, `degree` die Anzahl der Pins und `x1, y1, x2, y2` die Eckpunkte der Zelle. Dabei gilt immer $x1 < x2$ und $y1 < y2$. Das Element `cell_pin_index`

Name	Wert	Bedeutung	
UNDEF	-1	Wert nicht definiert	
X	0	Segmenttyp:	X-Segment
Y	1		Y-Segment
LAYER_1	0	Segment liegt in:	Layer 1
LAYER_2	1		Layer 2
SE_OPEN	0	Segmentanfang/-ende:	offen
SE_SEG	1		mit Segment verbunden
SE_VIA	2		mit Kontaktloch verbunden
SE_PIN	4		mit Pin oder Pad verbunden
TRUE	1	Wahrheitswerte:	wahr
FALSE	0		falsch
PORTSIZE	28	Größe eines Pads im Layout	

Tabelle 5.1: Konstanten im Programm EXTRACT

enthält den Index des ersten zu dieser Zelle gehörenden Elements im Array `cell_pin_array`.

Im Array `cell_pin_array` wiederum werden Informationen für jeden Pin jeder Zelle abgelegt. Dieses Array enthält Strukturen des Typs `TPinInfo`, die folgendermaßen definiert sind:

```
typedef struct
{
  int net;
  int subnet;
} TPinInfo;
```

Im Element `net` befindet sich die Netznummer und im Element `subnet` die Teilnetznummer des Netzes, an das dieser Anschlu angeschlossen ist. Diese Informationen werden beim Extrahieren gewonnen.

Die in der Layoutdatei enthaltenen Pads werden im Array `port_array` abgelegt. Dieses Array enthält für jeden Pad eine Struktur des Typs `TPort`:

```
typedef struct
{
  int index;
  int net;
  int subnet;
  int x1, y1, x2, y2;
} TPort;
```

Dort befindet sich in `index` die Ordnungsnummer des Pads, und in `x1, y1, x2, y2` die Eckpunkte, wobei auch hier gilt: $x1 < x2$ und $y1 < y2$. In `net` und `subnet` werden während des Extrahierens die Netz- und die Teilnetznummer des Netzes abgelegt, an die dieser Pad angeschlossen ist. Da jeder Pad nur einen Anschluß haben kann, ist es, anders als bei den Anschlüssen der Zellen, nicht nötig, hierfür ein weiteres Array anzulegen.

Die aus der Layoutdatei eingelesenen Segmente werden im Array `segment_array` abgelegt. Dieses Array enthält für jedes Segment eine Struktur vom Typ `TSegment`, die folgendermaßen definiert ist:

```
typedef struct
{
  TSegInfo info;
  int first_coord;
  int last_coord;
  int track_coord;
  int net;
  int mark;
} TSegment;
```

Dabei enthält `first_coord` die Anfangs- und `last_coord` die Endkoordinate des Segments. Das Element `track_coord` enthält bei X-Segmenten die gemeinsame y-Koordinate und bei Y-Segmenten die gemeinsame x-Koordinate. Das Element `net` enthält die Netznummer und `mark` die beim Extrahieren vergebene Teilnetznummer des Segments. Das Element `info` ist ein Bitfeld vom Typ `TSegInfo`, das wie folgt definiert ist:

```
typedef struct
{
  unsigned type:      1;
  unsigned layer:     1;
  unsigned start:     3;
  unsigned end:       3;
  unsigned cross_con: 1;
  unsigned unused:    1
} TSegInfo;
```

Um Speicherplatz zu sparen, wurden hier alle Informationen zusammengefaßt, die jeweils weniger als ein Byte belegen. Dabei enthält `type` (1 Bit) die Richtung des Segments: X oder Y . Das Element `layer` (1 Bit) enthält die Nummer des Layers, in dem sich das Segment befindet. Dies kann LAYER_1 oder LAYER_2 sein. Die Elemente `start` und `end` (jeweils 3 Bits) enthalten Informationen darüber, ob der Anfangspunkt oder der Endpunkt des Segments offen oder mit einem anderen Segment, einem Kontaktloch oder einem Pin verbunden ist. Dies wird durch die Konstanten SE_OPEN, SE_SEG, SE_VIA und SE_PIN ausgedrückt. Das Element `cross_con` gibt an, ob das Segment durch Kreuzung im selben Layer mit einem anderen Segment des gleichen Netzes verbunden ist und wird entsprechend auf TRUE oder FALSE gesetzt. Das Element `unused` wird auf TRUE gesetzt, wenn das Segment überflüssig ist, sonst ist es FALSE.

Die aus den Originalsegmenten erzeugten normalisierten Segmente werden im Array `normsegment_array` abgelegt. Dieses Array enthält für jedes normalisierte Segment eine Struktur vom Typ `TNormSegment`:

```
typedef struct
{
  TSegInfo info;
  int first_coord;
  int last_coord;
  int track_coord;
  int net;
  int mark;
  int seg_idx;
} TNormSegment;
```

Die Bedeutung der einzelnen Elemente dieser Struktur entspricht denen der Struktur `TSegment`. Es kommt jedoch noch das Element `seg_idx` hinzu, das den Index des Originalsegments enthält, aus dem das normalisierte Segment erzeugt worden ist.

Die in der Layoutdatei enthaltenen Kontaktlöcher werden in das Array `via_array` eingelesen. Dieses enthält für jedes Kontaktloch eine Struktur vom Typ `TVia`:

```
typedef struct
{
  int x, y;
  int seg_l1, seg_l2;
} TVia;
```

Die Elemente `x` und `y` enthalten die Koordinaten des Kontaktloches. Die Elemente `seg_l1` und `seg_l2` enthalten die Indizes der beiden durch dieses Kontaktloch verbundenen Segmente aus Layer 1 bzw. Layer 2. Sollte es kein solches Segment geben, wird das entsprechende Element auf den Wert UNDEF gesetzt.

Informationen über die einzelnen Netze werden im Array `net_info_array` abgelegt. Dieses Array enthält für jedes Netz eine Struktur vom Typ `TNetInfo`:

```
typedef struct
{
  int len;
  int neighbours;
  int n_layer1;
  int n_layer2;
} TNetInfo;
```

In dieser Struktur enthält das Element `len` die Länge des Netzes (aus der Layoutdatei), und das Element `neighbours` die beim Extrahieren ermittelte Anzahl der Nachbarn. Die Elemente `n_layer1` und `n_layer2` enthalten jeweils den Index des ersten zu diesem Netz gehörenden normalisierten Segments aus Layer 1 bzw. Layer 2.

5.1.2 Die Grundstruktur des Programms

Nachdem die zugrundeliegenden Datenstrukturen beschrieben wurden, wird in diesem Abschnitt das grobe Ablaufschema des Programms wiedergegeben. Das Programm erhält als Eingabedatei eine ESPRO-Layoutdatei. Die Ausgabedateien dieses Programms sind:

- Die Extraktionsdatei: In diese Datei werden die extrahierten Netzlisten hineingeschrieben. Sie ist im Aufbau einer ESPRO-Basisdatei sehr ähnlich. Näheres zu ihrem Aufbau siehe Anhang A.1.
- Die Fehlerdatei: Dies ist eine Textdatei, in der alle Fehler, die das Programm im Layout findet, protokolliert werden. Eine Beschreibung aller Fehlermeldungen, die in dieser Datei auftreten können, befindet sich im Anhang A.1.

Optional kann außerdem eine Layoutdatei erzeugt werden, in der anstelle der Originalsegmente die normalisierten Segmente verwendet werden. Dies gehört streng genommen jedoch nicht zur eigentlichen Aufgabe und ist deshalb auch nur als Option vorgesehen.
Des weiteren muß vom Benutzer die sog. *Warnstufe* festgelegt werden. Diese bestimmt, ob bei Fehlern alle Fehlermeldungen vollständig ausgegeben werden sollen, oder ob einzelne Meldungen unterdrückt werden sollen. Eine genaue Beschreibung befindet sich im Anhang A.1.

Nachdem der Benutzer alle Parameter festgelegt hat, beginnt das Programm mit dem Einlesen der Dateien. Da in einer ESPRO-Layoutdatei keine Informationen über die Anzahl der Segmente und Kontaktlöcher enthalten sind, muß das Einlesen der Layoutdatei in zwei Durchgängen erfolgen: Im ersten Durchgang werden nur die Segmente und Kontaktlöcher gezählt, dann wird entsprechend viel Speicher für die Datenfelder reserviert, und danach erfolgt das eigentliche Einlesen der Layoutdatei in die dafür vorgesehenen Datenfelder.
Nun ist es nötig, die Datenfelder zu sortieren. Die Daten werden dabei folgendermaßen sortiert:

- Die Segmente (`segment_array`): Die Segmente werden zunächst nach der Zugehörigkeit zum Layer und danach nach ihrer Netzzugehörigkeit sortiert.
- Die Kontaktlöcher (`via_array`): Die Kontaktlöcher werden nach ihren Koordinaten sortiert: zuerst nach der y- und dann nach der x-Koordinate
- Die Zellen (`cell_array`): Die Zellen werden, ähnlich wie die Kontaktlöcher, nach den Koordinaten ihres linken oberen Eckpunktes sortiert.
- Die Pads (`port_array`): Auch die Pads werden, wie die Zellen, nach den Koordinaten ihres linken oberen Eckpunktes sortiert.

Durch diese Sortierung wird die Anzahl der Vergleiche, die bei den folgenden Prozeduren nötig sind, auf ein Minimum reduziert, was zu einer erheblichen Beschleunigung und auch zu einer einfacheren Implementierung der Algorithmen führt.
Sollte während des Sortierens der Kontaktlöcher festgestellt werden, daß es äquivalente Kontaktlöcher gibt, so werden diese jetzt aus dem Speicher entfernt, da sich diese störend auf die nachfolgenden Prozeduren auswirken können.
Damit sieht das grobe Ablaufschema des Hauptprogramms (`main()`) in Pseudo-C wie folgt aus. Dabei sind Prozeduren, die im folgenden genauer behandelt werden, *kursiv* dargestellt, und es ist jeweils die Nummer des Abschnitts angegeben, in dem die Prozedur behandelt wird.

```
main()
{
  Initialisieren der Variablen;
  Abfragen der Dateinamen (Benutzereingabe);
  Abfragen der Warnstufe (Benutzereingabe);
  Analysieren der Layoutdatei;
  Anlegen der Datenfelder (Speicherplatz reservieren);
  Einlesen der Layoutdatei;
  Sortieren der Daten;
  Entfernen äquivalenter Kontaktlöcher;

  /*--- Aufbereitung des Layouts ---*/
  break_segments();             //Normalisieren der Segmente (s. 5.1.3)

  /*--- Eigentliches Extrahieren ---*/
  do_extraction();              //Extrahieren der Netzlisten (s. 5.1.4)

  /*--- Suchen von Fehlern im Layout ---*/
  find_unused_vias();           //überflüssige Kontaktlöcher suchen (s. 5.1.6)
  find_shorcircuit_vias();      //Kurzschlüsse an Vias suchen (s. 5.1.7)
  check_shortcircuit_segments(); //Segmentkurzschlüsse suchen (s. 5.1.8)
  check_open_segments();        //offenene Segmentenden suchen (s. 5.1.9)
  check_design_rules();         //Design-Rule-Verletzungen suchen (s. 5.1.10)
  check_circles();              //Kreise suchen (s. 5.1.11)
}
```

In den nun folgenden Abschnitten werden grundsätzliche, der Implementierung zugrunde liegende, Ideen erläutert und die aus programmtechnischer Sicht interessanten Prozeduren beschrieben.

5.1.3 Aufbereitung des Layouts: break_segments()

Die Prozedur `break_segments()` bereitet das Layout für die nachfolgenden Prozeduren auf. Diese Aufbereitung besteht darin, daß die Segmente normalisiert werden. Dies bedeutet, daß Segmente, die mehr als zwei Kontaktlöcher verbinden, in mehrere normalisierte Segmente heruntergebrochen werden, die jeweils zwei Kontaktlöcher miteinander verbinden.
Um ein Segment zu normalisieren, muß für jeden Punkt, der auf diesem Segment liegt, geprüft werden, ob sich dort ein Kontaktloch befindet. Ist dies der Fall, wird ein normalisiertes Segment erzeugt, das das letzte, auf dem betrachteten Segment liegende Kontaktloch (oder den Anfangspunkt des Segments) mit diesem Kontaktloch verbindet.
Treten Segmente auf, die die Länge Null haben, werden dafür trotzdem normalisierte Segmente erzeugt. Außerdem wird dann eine Fehlermeldung in die Fehlerdatei geschrieben.
Vorbedingung für diese Prozedur ist die richtige Sortierung der Segmente und Kontaktlöcher.
In Pseudo-C sieht diese Prozedur folgendermaßen aus (`segment_array` enthält die Originalsegmente, die normalisierten Segmente werden in `normsegment_array` abgelegt, `n_o_normsegments` ist eine globale Variable, die die Anzahl der normalisierten Segmente enthält):

```
break_segments()
{
  n_o_normsegments=0;

  //Alle Originalsegmente abarbeiten
  for(s=0; s<n_o_segments; s++)
  {
    if(Segment in segment_array[s] hat Länge 0)
    {
      Fehlermeldung('Segment der Länge Null');
      Erzeuge normalisiertes Segment der Länge Null (norm_seg);
      normsegment_array[n_o_normsegments++]=norm_seg;
      continue; //Weiter mit nchstem Segment (Rest der Schleife überspringen)
    }
    start_via_flag=FALSE;
    first_normseg_flag=TRUE;
    k_old=segment_array[s].first_coord;

    //Für alle Punkte auf diesem Segment
    for(k=segment_array[s].first_coord; k<=segment_array[s].last_coord; k++)
    {
      Suche ein Kontaktloch an dieser Stelle des Segments (binäre Suche);
      Gibt es ein solches Kontaktloch, setze found_via auf dessen Index;
      if(found_via!=UNDEF)
      {
        //Kontaktloch mit dem Index dieses Segmentes markieren
        if(segment_array[s].info.layer==LAYER_1) via_array[found_via].seg_l1=s;
        else via_aray[found_via].seg_l2=s;

        if(k!=k_old)
        {
          Erzeuge norm. Segment norm_seg mit Anfangskoordinate k_old und
          Endkoordinate k;

          if(start_via_flag)
          {
            norm_seg.info.start=SE_VIA;
            if(!first_normseg_flag) norm_seg.info.start|=SE_SEG;
          }
          else norm_seg.info.start=SE_OPEN;
```

```
          if(k<segment_array[s].last_coord) norm_seg.info.end=SE_SEG|SE_VIA;
          else norm_seg.info.end=SE_VIA;
          normsegment_array[n_o_normsegments++]=norm_seg;

          k_old=k;
          start_via_flag=TRUE;
          first_normseg_flag=FALSE;
        }
        else start_via_flag=TRUE;
      }
    }
    if(k_old<segment_array[s].last_coord)
    {
      Erzeuge norm. Segment norm_seg mit Anfangskoordinate k_old und
      Endkoordinate segment_array[s].last_coord;

      if(start_via_flag) norm_seg.info.start=SE_VIA;
      else norm_seg.info.start=SE_OPEN;
      if(!first_normseg_flag) norm_seg.info.start|=SE_SEG;

      normsegment_array[n_o_normsegments++]=norm_seg;
    }
  }
}
```

Am Ende dieser Prozedur stehen die normalisierten Segmente im Array `normsegment_array` und die Variable `n_o_normsegments` enthält deren Anzahl. Da die Originalsegmente der Reihe nach abgearbeitet werden, haben die normalisierten Segmente die gleiche Sortierung wie die Originalsegmente. Außerdem ist jedes Kontaktloch mit dem Index des Segments aus dem ersten Layer und dem Index des Segments aus dem zweiten Layer versehen, die es miteinander verbindet.

5.1.4 Extrahieren der Netzlisten: do_extraction()

Diese Prozedur führt das eigentliche Extrahieren durch. Das Extrahieren erfolgt dabei netzweise. Für jedes Netz wird zunächst mit Hilfe der Prozedur *check_connectivity()* geprüft, ob die zu diesem Netz gehörenden Segmente miteinander verbunden sind. Die Prozedur *check_connectivity()* wird später im Detail beschrieben.
Nach dem Aufruf von *check_connectivity()* für ein Netz wird die Netzliste dieses Netzes rekonstruiert. Dazu wird für jedes Segment, das zu dem aktuellen Netz gehört, geprüft, ob es an einem Pin oder an einem Pad beginnt oder endet. Dies erfolgt durch binäre Suche in den Arrays `cell_array` und `port_array`. Wurde ein solcher Pin oder Pad gefunden, wird dieser mit der Netznummer und der Teilnetznummer des Segments markiert. Außerdem wird das Segment mit einer entsprechenden Information versehen. Sollte der Pin oder Pad schon mit der Netznummer eines anderen Netzes markiert sein, handelt es sich um einen Kurzschluß an einem Pin oder Pad, und es wird eine entsprechende Fehlermeldung ausgegeben. Durch die Sortierung der Segmente stehen für jeden Layer alle Segmente eines Netzes hintereinander im Array, so daß die Segmente nicht gesucht werden müssen. Es muß nur bekannt sein, wo der zweite Layer beginnt. Dadurch wird sowohl die Implementierung vereinfacht als auch der Algorithmus beschleunigt.
Nachdem die Netzliste des aktuellen Netzes rekonstruiert ist, wird die Netzliste in die Extraktionsdatei geschrieben. Dazu müssen alle Pins und alle Pads, die mit der Netznummer des aktuellen Netzes markiert sind, in die Extraktionsdatei geschrieben werden.
Die Zugehörigkeit eines Pins oder Pads zu einer bestimmten Netzliste wird also dadurch ausgedrückt, daß der Pin oder Pad mit der Nummer des entsprechenden Netzes markiert wird. Damit werden keine aufwendigen Mengen- oder Listenoperationen benötigt, die in C nicht und auch in anderen Programmiersprachen oft nur sehr eingeschränkt vorhanden sind. Auch kann bei dieser

Methode leicht überprüft werden, ob ein Element bereits zu einer Liste gehört, da nur die Markierung des Elements abgefragt werden muß und nicht alle Listen nach dem Element durchsucht werden müssen.
Die Prozedur `do_extraction()` hat folgende Vorbedingungen:

- Sortierung der Daten wie in 5.1.2 beschrieben
- Die Netznummern der Pins und Pads müssen mit dem Wert UNDEF (zu keinem Netz gehörig) initialisiert sein

In Pseudo-C sieht diese Prozedur wie folgt aus (die globale Variable `start_layer_2` enthält den Index des ersten Originalsegments im zweiten Layer):

```
do_extraction()
{
  int layer1, layer2, act_net, n_o_subnets, i, s;
  //Indizes des ersten Segments im ersten bzw. zweiten Layer
  layer1=0;
  layer2=start_layer_2;
  //Alle Segmente abarbeiten
  while(layer1<start_layer_2 || layer2<n_o_segments)
  {
    //Aktuelle Netznummer ermitteln
    if(layer1<start_layer_2 && layer2<n_o_segments)
      act_net=min(segment_array[layer1].net, segment_array[layer2].net);
    else if(layer1<start_layer_2) act_net=segment_array[layer1].net;
    else if(layer2<n_o_segments) act_net=segment_array[layer2].net;
    //Verbindungen des aktuellen Netzes prüfen
    n_o_subnets=check_connectivity(act_net);
    //An Segmente im Layer 1 angeschlossene Pins/Pads suchen
    while(layer1<start_layer_2)
    {
      //Schon nächstes Netz erreicht?
      if(segment_array[layer1].net!=act_net) break;
      if(Segment segment_array[layer1] beginnt oder endet an Pin/Pad)
      {
        if(Netznummer des Pin/Pad!=UNDEF) Fehlermeldung('Kurzschluß an Pin/Pad');
        Setze Netznummer und Teilnetznummer des Pin/Pad auf die des Segments;
      }
      layer1++;
    }
    //An Segmente im Layer 2 angeschlossene Pins/Pads suchen
    while(layer2<n_o_segments)
    {
      //Schon nächstes Netz erreicht?
      if(segment_array[layer2].net!=act_net) break;
      if(Segment segment_array[layer2] beginnt oder endet an Pin/Pad)
      {
        if(Netznummer des Pin/Pad!=UNDEF) Fehlermeldung('Kurzschluß an Pin/Pad');
        Setze Netznummer und Teilnetznummer des Pin/Pad auf die des Segments;
      }
      layer2++;
    }
```

```
    Schreibe alle Pins und Pads mit der Netznummer act_net in die
    Extraktionsdatei (Netzliste des Netzes act_net);
    //Suche nach überflüssigen Teilnetzen, falls es mehrere Teilnetze gibt
    if(n_o_subnets>1)
      check_subnet_use(act_net, n_o_subnets); //(s. 5.1.5)
  }
  //Übertragen der Anschlußinformationen auf die norm. Segmente
  for(i=0; i<n_o_normsegments; i++)
  {
    s=normsegment_array[i].seg_idx;
    if(segment_array[s].first_coord==normsegment_array[i].first_coord)
      normsegment_array[i].info.start|=segment_array[s].info.start;
    if(segment_array[s].last_coord==normsegment_array[i].last_coord)
      normsegment_array[i].info.end|=segment_array[s].info.end;
  }
}
```

Am Ende dieser Prozedur sind die Netzlisten aller Netze rekonstruiert und stehen in der Extraktionsdatei. Außerdem sind alle Segmente jedes Netzes, die zu dem gleichen Teilnetz gehören, mit der gleichen Teilnetznummer markiert. Des weiteren enthält jedes Originalsegment und jedes normalisierte Segment Informationen darüber, ob es an seinem Anfangs- bzw. Endpunkt offen ist oder an einem Via, einem Pin oder Pad oder einem anderen Segment beginnt oder endet.

Prüfen der Verbindungen: check_connectivity()

Diese Prozedur prft, ob alle Segmente eines Netzes miteinander verbunden sind. Die Nummer des zu prüfenden Netzes wird als Parameter übergeben.
Zunächst werden die Teilnetznummern initialisiert. Dabei erhält jedes normalisierte Segment des Netzes eine andere Teilnetznummer, beginnend mit der Teilnetznummer 1. Nur normalisierte Segmente, die zu demselben Originalsegment gehören, erhalten gleiche Teilnetznummen.
Danach wird geprüft, ob beide Layer Segmente dieses Netzes enthalten. Ist dies der Fall, wird mit Hilfe der Prozedur *check_between_layers()* nach Verbindungen zwischen den beiden Layern gesucht. Sind nun die kleinste und die größte in diesem Netz auftretende Teilnetznummer gleich, so sind alle normalisierten Segmente des Netzes miteinander verbunden. Andernfalls wird mit Hilfe der Prozedur *check_same_layer()* nach Verbindungen innerhalb jedes Layers gesucht.
Enthält nur ein Layer Segmente des zu prüfenden Netzes, werden nur mit Hilfe der Prozedur *check_same_layer()* Verbindungen innerhalb dieses Layers gesucht.
Zum Schluß werden die Teilnetznummern der normalisierten Segmente auf die Originalsegmente übertragen. Außerdem wird die Anzahl der Teilnetze ermittelt und als Rückgabewert zurückgegeben.
Vorbedingung für diese Prozedur ist, daß die Segmente richtig sortiert und die normalisierten Segmente erzeugt worden sind.
In Pseudo-C sieht die Prozedur folgendermaßen aus:

```
int check_connectivity(int net_no)
{
  int norm_layer1, norm_layer2, norm_layer1_end, norm_layer2_end;
  int min_mark, max_mark;
  //Indizes des ersten und letzten norm. Segments aus Layer 1 bzw. 2
  norm_layer1=net_info_array[net_no].n_layer1;
  norm_layer2=net_info_array[net_no].n_layer2;
  norm_layer1_end=net_info_array[net_no+1].n_layer1;
```

```
  norm_layer2_end=net_info_array[net_no+1].n_layer2;

  Initialisiere Teilnetznummern;

  //Enthalten beide Layer Segmente des Netzes?
  if(norm_layer1<norm_layer1_end && norm_layer2<norm_layer2_end)
  {
    //Verbindungen zwischen den Layern suchen
    check_between_layers(norm_layer1, norm_layer1_end, norm_layer2, norm_layer2_end);

    min_mark=kleinste Teilnetznummer;
    max_mark=größte Teilnetznummer;

    //Gibt es noch mehr als ein Teilnetz?
    if(min_mark!=max_mark)
    {
      //Verbindungen innerhalb der beiden Layer suchen
      check_same_layer(LAYER_1, norm_layer1, norm_layer1_end,
            norm_layer2, norm_layer2_end);
      check_same_layer(LAYER_2, norm_layer1, norm_layer1_end,
            norm_layer2, norm_layer2_end);

      min_mark=kleinste Teilnetznummer;
      max_mark=größte Teilnetznummer;
    }
  }
  else
  {
    //Nur Layer 1 enthält Segmente dieses Netzes
    if(norm_layer1<norm_layer1_end)
      check_same_layer(LAYER_1, norm_layer1, norm_layer1_end,
            norm_layer2, norm_layer2_end);

    //Nur Layer 2 enthält Segmente dieses Netzes
    if(norm_layer2<norm_layer2_end)
      check_same_layer(LAYER_2, norm_layer1, norm_layer1_end,
            norm_layer2, norm_layer2_end);

    min_mark=kleinste Teilnetznummer;
    max_mark=größte Teilnetznummer;
  }
  Übertrage die Teilnetznummern der normalisierten Segmente
  auf die Originalsegmente;

  return Anzahl der Teilnetze;
}
```

Am Ende dieser Prozedur sind alle Segmente des zu prüfenden Netzes, die zum gleichen Teilnetz gehören, mit der gleichen Teilnetznummer markiert. Gibt es nur ein Teilnetz, sind alle Segmente dieses Netzes miteinander verbunden.

Verbindungen zwischen Layern: check_between_layers()

Diese Prozedur sucht nach Verbindungen von normalisierten Segmenten eines Netzes zwischen den beiden Layern durch Kontaktlcher.
Dabei werden die Anfangs- und Endpunkte jedes normalisierten Segments aus Layer 1, das zum Netz gehört und in einem Kontaktloch beginnt und/oder endet, mit den Anfangs- und Endpunkten jedes normalisierten Segments aus Layer 2 verglichen. Wurde eine solche Übereinstimmung

gefunden, wird festgestellt, welches der beiden normalisierten Segmente die kleinere und welches die größere Teilnetznummer hat. Dann werden in beiden Layern alle normalisierten Segmente dieses Netzes, die die größere Teilnetznummer haben, auf die kleinere Teilnetznummer gesetzt. Die betreffenden normalisierten Segmente werden außerdem mit der Information versehen, daß sie jeweils mit einem anderen Segment verbunden sind.
Auch bei dieser Prozedur ist die Sortierung der normalisierten Segmente vorteilhaft, da alle normalisierten Segmente jedes Layers, die zu einem Netz gehören, hintereinander in dem Array `normsegment_array` stehen. Dadurch ist es einfach, die Suchräume für die Segmente eines Netzes einzuschränken, was zu einer erheblichen Beschleunigung des Vorgangs führt. Auch müssen die zu einem bestimmten Netz gehörenden Segmente nicht extra gesucht werden, was zu einer Vereinfachung der Implementierung führt. Die Grenzen der Suchräume für jeden der beiden Layer werden dabei als Parameter übergeben.
Die Prozedur hat folgende Vorbedingungen:

- Sortierung der Daten wie in 5.1.2 beschrieben
- In beiden Layern mu sich mindestens ein zu diesem Netz gehrendes Segment befinden
- Die Teilnetznummern mssen richtig initialisiert worden sein (beginnend mit 1; normalisierte Segmente, die zum gleichen Originalsegment gehren, haben gleiche Teilnetznummern)

In Pseudo-C sieht die Prozedur `check_between_layers()` folgendermaßen aus:

```
check_between_layers(int norm_layer1, int norm_layer1_end,
            int norm_layer2, int norm_layer2_end)
{
  int i, s;
  int old_mark, new_mark;
  for(i=norm_layer1; i<norm_layer1_end; i++) //Alle norm. Seg. des Netzes in Layer 1
  {
    //Kein Kontaktloch an Anfang und Ende?
    if((normsegment_array[i].info.start & SE_VIA)==0 &&
    (normsegment_array[i].info.end & SE_VIA)==0) continue;
    for(s=norm_layer2; s<norm_layer2_end; s++) //Alle norm. Seg. des Netzes in Layer 2
    {
      old_mark=max(normsegment_array[i].mark, normsegment_array[s].mark);
      new_mark=min(normsegment_array[i].mark, normsegment_array[s].mark);
      //Kontaktloch am Anfangspunkt?
      if((normsegment_array[i].info.start & SE_VIA)!=0)
      {
        if(Anfangspunkt von Segment i == Anfangspunkt von Segment s)
        {
          Setze in beiden Layern alle norm. Segmente dieses Netzes
          mit der Teilnetznummer old_mark auf die Teilnetznummer new_mark;
          normsegment_array[i].info.start|=SE_SEG;
          normsegment_array[s].info.start|=SE_SEG;
        }
        else if(Anfangspunkt von Segment i == Endpunkt von Segment s)
        {
          Setze in beiden Layern alle norm. Segmente dieses Netzes
          mit der Teilnetznummer old_mark auf die Teilnetznummer new_mark;
          normsegment_array[i].info.start|=SE_SEG;
          normsegment_array[s].info.end|=SE_SEG;
        }
      }
```

```
      //Kontaktloch am Endpunkt?
      if((normsegment_array[i].info.end & SE_VIA)!=0)
      {
        if(Endpunkt von Segment i == Anfangspunkt von Segment s)
        {
          Setze in beiden Layern alle norm. Segmente dieses Netzes
          mit der Teilnetznummer old_mark auf die Teilnetznummer new_mark;
          normsegment_array[i].info.end|=SE_SEG;
          normsegment_array[s].info.start|=SE_SEG;
        }
        else if(Endpunkt von Segment i == Endpunkt von Segment s)
        {
          Setze in beiden Layern alle norm. Segmente dieses Netzes
          mit der Teilnetznummer old_mark auf die Teilnetznummer new_mark;
          normsegment_array[i].info.end|=SE_SEG;
          normsegment_array[s].info.end|=SE_SEG;
        }
      }
    }
  }
}
```

Am Ende dieser Prozedur sind alle normalisierten Segmente des durch den Suchraum gegebenen Netzes, die miteinander verbunden sind, mit derselben Teilnetznummer markiert.

Verbindungen in einem Layer: check_same_layer()

Diese Prozedur sucht nach Verbindungen von normalisierten Segmenten eines Netzes im gleichen Layer.
Dabei werden die Koordinaten jedes normalisierten Segmentes eines Layers, die zu dem zu prüfenden Netz gehören, miteinander verglichen. Gibt es eine Kreuzung oder eine Überlappung zweier solcher Segmente, so handelt es sich um eine Verbindung. Ähnlich wie in der Prozedur `check_between_layers()` wird dann festgestellt, welches der beiden normalisierten Segmente die kleinere und welches die größere Teilnetznummer hat. Dann werden in beiden Layern alle normalisierten Segmente dieses Netzes, die die größere Teilnetznummer haben, auf die kleinere Teilnetznummer gesetzt. Außerdem werden die betreffenden normalisierten Segmente mit der Information versehen, daß diese jeweils mit einem anderen Segment verbunden sind.
Auch hier wird der Suchraum für die Segmente durch deren Sortierung stark eingeschränkt. Als Parameter werden der zu prüfende Layer und die Suchraumgrenzen des Netzes übergeben.
Die Prozedur hat folgende Vorbedingungen:

- Sortierung der Daten wie in 5.1.2 beschrieben
- Die Teilnetznummern mssen richtig initialisiert worden sein (beginnend mit 1; normalisierte Segmente, die zum gleichen Originalsegment gehren, haben gleiche Teilnetznummern), oder `check_between_layers()` mu vorher ausgeführt worden sein.

In Pseudo-C sieht diese Prozedur wie folgt aus:

```
check_same_layer(int which, int norm_layer1, int norm_layer1_end,
              int norm_layer2, int norm_layer2_end)
{
  int layer_start, layer_end;
  int i, s, min_mark, max_mark, old_mark, new_mark;
```

```
  if(which==LAYER_1) //Indizes des ersten und letzten norm. Segments des Layers
  {
    layer_start=norm_layer1;
    layer_end=norm_layer1_end;
  }
  else if(which==LAYER_2)
  {
    layer_start=norm_layer2;
    layer_end=norm_layer2_end;
  }
  else return;

  min_mark=kleinste Teilnetznummer;
  max_mark=größte Teilnetznummer;

  if(min_mark==max_mark) return; //Es gibt nur ein Teilnetz, also Abbruch

  for(i=layer_start; i<layer_end; i++) //Alle norm. Seg. des Netzes in diesem Layer
  {
    if(normsegment_array[i].mark==min_mark) continue;

    for(s=layer_start; s<layer_end; s++) //Alle norm. Seg. des Netzes in diesem Layer
    {
      //Gleicher Index oder gleiche Teilnetznummer?
      if(s==i || normsegment_array[i].mark==normsegment_array[s].mark) continue;

      if(Überlappung der norm. Segmente i und s)
      {
        old_mark=max(normsegment_array[i].mark, normsegment_array[s].mark);
        new_mark=min(normsegment_array[i].mark, normsegment_array[s].mark);

        Setze in beiden Layern alle norm. Segmente dieses Netzes
        mit der Teilnetznummer old_mark auf die Teilnetznummer new_mark;

        if(i<s) //Anschlußinformationen je nach Art der Überlappung
        {
          normsegment_array[i].info.end|=SE_SEG;
          normsegment_array[s].info.start|=SE_SEG;
        }
        else
        {
          normsegment_array[s].info.end|=SE_SEG;
          normsegment_array[i].info.start|=SE_SEG;
        }
      }
      else if(Kreuzung der norm. Segmente i und s)
      {
        old_mark=max(normsegment_array[i].mark, normsegment_array[s].mark);
        new_mark=min(normsegment_array[i].mark, normsegment_array[s].mark);

        Setze in beiden Layern alle norm. Segmente dieses Netzes
        mit der Teilnetznummer old_mark auf die Teilnetznummer new_mark;

        normsegment_array[i].info.cross_con=TRUE;
        normsegment_array[s].info.cross_con=TRUE;
      }
    }
  }
}
```

Am Ende sind alle normalisierten Segmente des durch den Suchraum gegebenen Netzes, die innerhalb eines Layers miteinander verbunden sind, mit derselben Teilnetznummer markiert.

5.1.5 Überflüssige Teilnetze: check_subnet_use()

Diese Prozedur prft, ob ein Netz überflüssige Teilnetze enthält. Da jedes Segment beim Extrahieren mit Informationen versehen worden ist, ob dieses offen oder mit einem anderen Segment, einem Via oder einem Netznachbarn verbunden ist, muß hier lediglich eine Auswertung dieser Informationen durchgeführt werden. Ist kein Segment eines Teilnetzes mit einem Netznachbarn verbunden, so ist dieses Teilnetz überflüssig. Als Parameter werden die Nummer des zu prüfenden Netzes und die Anzahl der Teilnetze dieses Netzes übergeben. Vorbedingung für diese Prozedur ist, daß die Extraktion durchgeführt wurde.
In Pseudo-C sieht die Prozedur wie folgt aus:

```
check_subnet_use(int net_no, int subnets)
{
  int i, s, layer1, layer2, used;

  //Indizes des ersten Originalsegments des Netzes in Layer 1 und 2
  layer1=normsegment_array[net_info_array[net_no].n_layer1].seg_idx;
  layer2=normsegment_array[net_info_array[net_no].n_layer2].seg_idx;

  //Alle Teilnetze des Netzes
  for(i=1; i<=subnets; i++)
  {
    used=FALSE;

    //In Layer 1 mit einem Pin/Pad verbundene Segmente suchen
    s=layer1;
    while(!used && segment_array[s].net==net_no)
    {
      if(segment_array[s].mark==i &&
      (segment_array[s].info.start & SE_PIN ||
      segment_array[s].info.end & SE_PIN)) used=TRUE;

      s++;
    }
    //In Layer 2 mit einem Pin/Pad verbundene Segmente suchen
    s=layer2;
    while(!used && segment_array[s].net==net_no)
    {
      if(segment_array[s].mark==i &&
      (segment_array[s].info.start & SE_PIN ||
      segment_array[s].info.end & SE_PIN)) used=TRUE;

      s++;
    }
    //Keine mit einem Pin/Pad verbundenen Segmente gefunden
    if(!used) Warnung('Überflüssiges Teilnetz');
  }
}
```

Die Prozedur gibt eine Warnung aus, wenn das Netz ein oder mehrere überflüssige Teilnetze enthält. Es werden keine Daten verändert.

5.1.6 Überflüssige Kontaktlöcher: find_unused_vias()

Um festzustellen, ob ein Kontaktloch überflüssig ist, müssen die Segmentindizes ausgewertet werden, mit denen die Kontaktlöcher schon während des Normalisierens versehen wurden. Haben ein Index oder beide Indizes den Wert UNDEF, so liegt das Kontaktloch entweder auf nur einem oder gar keinem Segment, ist also überflüssig. Es ist also nur eine Auswertung schon vorhandener Informationen notwendig. Dies ist trivial und sieht folgendermaßen aus (die globale Variable `n_o_vias` enthält die Anzahl aller Kontaktlöcher im Array `via_array`):

```
find_unused_vias()
{
  int i;

  for(i=0; i<n_o_vias; i++)
  {
    if(via_array[i].seg_l1==UNDEF || via_array[i].seg_l2==UNDEF)
    {
      Warnung('Überflüssiges Kontaktloch');
    }
  }
}
```

5.1.7 Kurzschlüsse an Vias: find_shortcircuit_vias()

Um Kurzschlüsse zu finden, die durch Kontaktlöcher verursacht werden, ist – ähnlich wie bei der Suche nach überflüssigen Kontaktlöchern – nur eine Auswertung vorhandener Informationen nötig. Auch hier werden die Segmentindizes ausgewertet: Gehören die Segmente, die zu diesen Indizes gehören, zu unterschiedlichen Netzen, so werden durch dieses Kontaktloch zwei verschiedene Netze verbunden, es liegt also ein Kurzschluß vor. Dies wird durch die Prozedur `find_shortcircuit_vias()` überprüft:

```
find_shortcircuit_vias()
{
  int i, l1seg, l2seg;

  for(i=0; i<n_o_vias; i++)
  {
    //Segment-Indizes des Kontaktloches
    l1seg=via_array[i].seg_l1;
    l2seg=via_array[i].seg_l2;

    //Das Kontaktloch darf nicht überflüssig sein
    if(l1seg!=UNDEF && l2seg!=UNDEF)
    {
      //Kurzschluß, wenn Netznummern der Segmente ungleich
      if(segment_array[l1seg].net!=segment_array[l2seg].net)
      {
        Fehlermeldung('Kurzschluß an Kontaktloch');
      }
    }
  }
}
```

5.1.8 Segmentkurzschlüsse: check_shortcircuit_segments()

Die Prozedur `check_shortcircuit_segments()` sucht nach Segmentkurzschlüssen. Dabei wird zunächst im ersten, danach im zweiten Layer gesucht. Es muß in jedem Layer geprüft werden, ob es Segmente gibt, die sich mit Segmenten anderer Netze überlappen oder kreuzen. Die Überprüfung erfolgt netzweise. Die Segmente des ersten Netzes müssen jeweils mit den Segmenten aller anderen Netze verglichen werden, die Segmente des n-ten Netzes mit den Segmenten aller anderen Netze ab Netz $n+1$. Damit sieht die Prozedur `check_shortcircuit_segments()` folgendermaßen aus (die globale Variable `n_o_nets` enthält die Anzahl der Netze, die globale Variable `start_norm_layer_2` enthält den Index des ersten normalisierten Segments des zweiten Layers):

```
check_shortcircuit_segments()
{
  int net, nseg_1, nseg_2;
  TNormSegment check_nseg;

  for(net=0; net<n_o_nets-1; net++) //Layer 1
  {
    //Erstes normalisiertes Segment des Netzes net im ersten Layer:
    nseg_1=net_info_array[net].n_layer1;

    while(normsegment_array[nseg_1].net==net) //Für alle Segmente des Netzes net
    {
      check_nseg=normsegment_array[nseg_1];

      //Alle norm. Segmente aller anderen Netze ab dem Netz net+1
      for(nseg_2=net_info_array[net+1].n_layer1; nseg_2<start_norm_layer_2;nseg_2++)
      {
        if(Kreuzung oder Überlappung von check_nseg und normsegment_array[nseg_2])
        {
          Fehlermeldung('Segmentkurzschluß in Layer 1');
        }
      }
      nseg_1++;
    }
  }
  for(net=0; net<n_o_nets-1; net++) //Layer 2
  {
    //Erstes normalisiertes Segment des Netzes net im zweiten Layer:
    nseg_1=net_info_array[net].n_layer2;

    while(normsegment_array[nseg_1].net==net) //Für alle Segmente des Netzes net
    {
      check_nseg=normsegment_array[nseg_1];

      //Alle norm. Segmente aller anderen Netze ab dem Netz net+1
      for(nseg_2=net_info_array[net+1].n_layer2; nseg_2<n_o_normsegments; nseg_2++)
      {
        if(Kreuzung oder Überlappung von check_nseg und normsegment_array[nseg_2])
        {
          Fehlermeldung('Segmentkurzschluß in Layer 2');
        }
      }
      nseg_1++;
    }
  }
}
```

5.1.9 Offene Segmentenden: check_open_segments()

Auch bei der Suche nach offenen Segmenten handelt es sich um eine Auswertung bereits vorhandener, beim Extrahieren gewonnener Informationen. Jedes normalisierte Segment wurde beim Extrahieren mit Informationen versehen, ob dessen Enden offen sind oder ob es durch Kreuzung mit einem anderen normalisierten Segment verbunden ist. Diese Informationen werden in der folgenden Prozedur ausgewertet:

```
check_open_segments()
{
  int nseg, nseg_e, seg_idx, start, end, no_con;

  for(nseg=0; nseg<n_o_normsegments; nseg++)
  {
    //Informationen über Anfangs- und Endpunkt: SE_OPEN, SE_SEG, SE_VIA, SE_PIN
    start=normsegment_array[nseg].info.start;
    end=normsegment_array[nseg].info.end;

    seg_idx=normsegment_array[nseg].seg_idx;
    if(normsegment_array[nseg].first_coord==segment_array[seg_idx].first_coord &&
      normsegment_array[nseg].last_coord==segment_array[seg_idx].last_coord &&
      !normsegment_array[nseg].info.cross_con)
        no_con=TRUE; //Segment hat keine Verbindungen durch Kreuzung
    else
        no_con=FALSE; //Segment hat Verbindungen durch Kreuzung

    if(!no_con)
    {
      //Letztes normalisiertes Segment ermitteln, das zu dem
      //Originalsegment gehört
      nseg_e=nseg;
      while(nseg_e<n_o_normsegments &&
      normsegment_array[nseg_e].seg_idx==seg_idx) nseg_e++;
      if(normsegment_array[nseg_e].seg_idx!=seg_idx) nseg_e--;
      end=normsegment_array[nseg_e].info.end;
      nseg=nseg_e;
    }

    if(start==SE_OPEN && end==SE_OPEN)
    {
      Warnung('Segment offen am Anfang und Ende');
      if(no_con) Warnung('Segment überflüssig');
    }
    else if(start==SE_OPEN)
    {
      Warnung('Segment offen am Anfang');
      if(no_con) Warnung('Segment überflüssig');
    }
    else if(end==SE_OPEN)
    {
      Warnung('Segment offen am Ende');
      if(no_con) Warnung('Segment überflüssig');
    }

    //Bei überfl. norm. Seg. wird das Flag unused auf TRUE gesetzt
    normsegment_array[nseg].info.unused=no_con;
  }
}
```

5.1.10 Design-Rule-Verletzungen: check_design_rules()

Die Prozedur `check_design_rules` überprüft die Einhaltung der einzigen Design Rule von ESPRO: Das Verbot von "Routing over the Cell". Dazu wird zu jedem Segment geprüft, ob das Segment mindestens eine Zelle schneidet. Schneidet ein Segment mindestens eine Zelle, so handelt es sich um eine Verletzung der Design Rule. Da die Zellen nach Koordinaten geordnet im Array `cell_array` stehen, kann binäre Suche verwendet werden. Die Prozedur sieht damit folgendermaßen aus (die globale Variable `n_o_segments` enthält die Anzahl aller Segmente):

```
check_design_rules()
{
  int i;

  for(i=0; i<n_o_segments; i++)
  {
    Suche eine Zelle, durch die das Segment aus segment_array[i]
    hindurchläuft (mittels binärer Suche im cell_array);

    if(eine solche Zelle existiert)
    {
      Fehlermeldung('Segment verläuft durch Zelle');
    }
  }
}
```

5.1.11 Kreise: check_circles()

Die Prozedur `check_circles()` sucht nach Kreisen in Netzen, die durch Segmente aus beiden Layern gebildet werden. Es wird fr jedes Netz die Anzahl der Kontaktlöcher ermittelt, an denen drei oder mehr normalisierte Segmente hängen. (Kontaktlöcher mit mehr als drei normalisierten Segmenten werden als entsprechend viele mit drei normalisierten Segmenten gezählt). Dazu werden für jedes Kontaktloch, das zu dem gerade betrachteten Netz gehört, alle normalisierten Segmente gesucht und gezählt, die in dem betreffenden Kontaktloch beginnen oder enden und zum betrachteten Netz gehören. Ist die Anzahl der Kontaktlöcher, die drei normalisierte Segmente miteinander verbinden, größer als die um zwei verminderte Anzahl der Nachbarn dieses Netzes, so enthält das Netz einen Kreis. Dabei dürfen überflüssige Segmente nicht mitgezählt werden, da es dann zu Falschmeldungen kommt. Deshalb ist eine Vorbedingung für diese Prozedur, daß die Prozedur `check_open_segments()` vorher ausgeführt worden ist.

```
check_circles()
{
  int i, s1, s2, net, n_o_con, via_count_3;

  for(net=0; net<n_o_nets; net++)
  {
    via_count_3=0;
    for(i=0; i<n_o_vias; i++)
    {
      //Segment-Indizes des Kontaktloches
      s1=via_array[i].seg_l1;
      s2=via_array[i].seg_l2;

      if(s1==UNDEF || s2==UNDEF) continue; //Kontaktloch überflüssig?

      //Wenn das Kontaktloch nur Segmente dieses Netzes verbindet
      if(segment_array[s1].net==segment_array[s2].net && segment_array[s1].net==net)
      {
        n_o_con=Anzahl der Segmente an diesem Kontaktloch;
```

```
        while(n_o_con>2)
        {
          //3 oder mehr: entsprechend viele mit 3 Segmenten
          via_count_3++;
          n_o_con--;
        }
      }
    }
    if(via_count_3>net_info_array[net].neighbours-2)
    {
      Fehlermeldung('Netz enthält Kreis');
    }
  }
}
```

5.2 Das Programm COMPARE

Der im Kapitel 4.4 beschriebene Algorithmus zum Netzlistenvergleich ist in dem Programm COMPARE implementiert. Die Struktur des Programms wird in diesem Abschnitt beschrieben.

5.2.1 Die Grundstruktur des Programms

Das Programm COMPARE erhält als Eingabe entweder eine Basisdatei und eine Extraktionsdatei oder zwei Extraktionsdateien. Das Ergebnis des Programms wird in eine Textdatei geschrieben, die im folgenden als Ergebnisdatei bezeichnet wird. Die Namen der Dateien werden durch den Benutzer festgelegt.

Nachdem der Benutzer die Dateinamen eingegeben hat, liest das Programm zunächst die Eingabedateien ein. Danach wird die Anzahl der Zellen, der Pads und der Netze miteinander verglichen. Gibt es hier Abweichungen, ist ein weiterer Vergleich nicht sinnvoll, da es sich dann wahrscheinlich um verschiedene Schaltungen handelt.

Danach werden die Breiten der Zellen (also die Anzahl der Anschlüsse jeder Zelle) verglichen, und Abweichungen in die Ergebnisdatei geschrieben.

Dann erfolgt der eigentliche Netzlistenvergleich. Jedes Netz aus der ersten Eingabedatei wird mit dem Netz aus der zweiten Eingabedatei verglichen, das den gleichen Index hat. Auch hier werden Unterschiede in die Ergebnisdatei geschrieben. Sollte ein Netz aus mehr als einem Teilnetz bestehen, so wird auch hierfür eine Fehlermeldung in die Ergebnisdatei geschrieben. Damit ist das Programm beendet.

Das Ablaufschema des Hauptprogramms (`main()`) sieht in Pseudo-C wie folgt aus:

```
main()
{
  Initialisieren der Variablen;
  Abfragen der Dateinamen (Benutzereingabe);
  Anlegen der Datenfelder (Speicherplatz reservieren);
  Einlesen der Dateien;

  Vergleich der Anzahl der Zellen, Pads und Netze;
  if(Unterschiede festgestellt)
  {
    //Verschiedene Schaltungen?
    Fehlermeldung ausgeben;
    exit;
  }
```

```
  Vergleich der Zellbreiten;
  if(Unterschiede festgestellt)
  {
    Unterschiede in Ergebnisdatei schreiben;
  }
  //Netzlistenvergleich
  for(net=0; net<Anzahl_Netze; net++)
  {
    Vergleiche Netzliste des Netzes net aus der ersten Eingabedatei
    mit der Netzliste des Netzes net aus der zweiten Eingabedatei;
    if(Unterschiede festgestellt)
    {
      Unterschiede in Ergebnisdatei schreiben;
    }
    if(Netz enthält mehrere Teilnetze)
    {
      Fehlermeldung in Ergebnisdatei schreiben;
    }
  }
}
```

Das Programm COMPARE wertet also die Informationen aus, die EXTRACT in der Extraktionsdatei liefert. Durch Vergleich mit der dazugehörigen Basisdatei (Originalnetzlisten) werden Informationen über fehlende und überzählige Netznachbarn gewonnen und in die Ergebnisdatei geschrieben. In der Extraktionsdatei befinden sich auch Informationen darüber, wieviele Teilnetze jedes Netz beinhaltet. Besteht ein Netz aus mehreren Teilnetzen, kann es möglicherweise unterbrochen sein, und ein entsprechender Hinweis wird in die Ergebnisdatei geschrieben.

Kapitel 6

Evaluierung des Tools

In diesem Kapitel wird das im Rahmen der Diplomarbeit entwickelte Verifikationstool evaluiert. Dazu wurden für verschiedene Beispielchips zunächst mit dem Router DKROUTER ([Kla94]) Layouts erzeugt. Aus den so enstandenen Layouts wurden dann mit dem Programm EXTRACT die Netzlisten extrahiert, und es erfolgte mit Hilfe des Programms COMPARE ein Vergleich der extrahierten Netzlisten mit den Originalnetzlisten. Fehler, die beim Extrahieren oder beim Netzlistenvergleich aufgetreten sind, werden näher analysiert und diskutiert. Es wird versucht, Hinweise zu geben, wo Fehler im Router DKROUTER sein könnten. Dies ist zwar nicht primäres Ziel der Arbeit, aber ein mögliches Anwendungsgebiet des Verifikationstools.
Danach wurden die Layouts der Beispielchips mit den Nachoptimierern NACHOPT und NETZMIN aus [Kas95] bearbeitet. Aus den auf diese Weise enstandenen, nachoptimierten Layouts wurden wieder die Netzlisten extrahiert, und der Netzlistenvergleich mit den Originalnetzlisten wurde durchgeführt. Auch hier werden Fehler, die beim Extrahieren oder beim Netzlistenvergleich aufgetreten sind, näher analysiert und diskutiert, und es soll zumindest versucht werden, Hinweise auf Fehler in den Nachoptimierern NACHOPT und NETZMIN zu geben.
Zur Evaluierung wurden die Beispielchips T10, T20, T50, T100, T200, T500 und T1000 verwendet. Dies sind mit dem Programm RND2CHP aus [Kla94] zufällig generierte Chips mit 10, 20, 50, 100, 200, 500 und 1000 Zellen. Außerdem wurden die Chips P45, P276 und P690 verwendet. Hierbei handelt es sich um Teile von am Fachbereich Informatik entwickelten Chips, die mit dem Programm EXL2CHP aus [Kla94] vom Solo-1400- in das ESPRO-Format konvertiert wurden, mit 45, 276 und 690 Zellen. Dieselben Beispielchips wurden auch in [Kla94] und [Kas95] zur Evaluierung der dort entstandenen Software verwendet.

6.1 Von DKROUTER erzeugte Layouts

In diesem Abschnitt werden die vom Router DKROUTER erzeugten Layouts bearbeitet. Zunächst wurden die Layouts mit dem Programm EXTRACT bearbeitet, um die Netzlisten zu extrahieren und Fehler im Layout zu finden. Danach erfolgte für jeden Beispielchip ein Netzlistenvergleich mit Hilfe des Programms COMPARE, um Unterschiede zwischen den Originalnetzlisten und den extrahierten Netzlisten feststellen zu können.

6.1.1 Extraktion und Fehlersuche

Zur Evaluierung des Programms EXTRACT wurde zuerst dessen Zeitbedarf für jeden Beispielchip ermittelt, außerdem die Anzahl der Originalsegmente und der daraus erzeugten nomalisierten Segmente, um aus diesen Werten auf den Speicherbedarf des Programms EXTRACT schließen zu können. Diese Werte sind in der Tabelle 6.1 zusammengefaßt. In diese Tabelle wurden außerdem die Anzahl der Netze und die Anzahl der Kontaktlöcher mit aufgenommen. Der Zeitbedarf für die einzelnen Beispiele wurde dabei auf einem PC 486/DX66 unter MS-DOS ermittelt.

Chip	Zeit (sec)	Netze	Segmente	Vias	norm. Segmente
T10	0.38	17	174	132	202
T20	1.88	32	381	299	437
T50	2.58	53	990	840	1176
T100	5.99	75	1980	1687	2360
T200	16.21	103	4155	3676	5028
T500	83.13	233	12080	11020	14395
T1000	330.66	241	23570	21734	28479
P45	1.43	67	571	429	635
P276	10.00	299	2710	2090	2901
P690	45.77	646	7616	6173	8158

Tabelle 6.1: Zeit- und Speicherbedarf des Programms EXTRACT mit vom DKROUTER erzeugten Layouts

Man sieht, daß der Zeitbedarf hauptsächlich von der Anzahl der Segmente und nicht von der Anzahl der Netze abhängig ist. Dies fällt insbesondere dann auf, wenn man die Beispiele T500 und P276 miteinander vergleicht. Der Chip P276 hat mehr Netze, jedoch deutlich weniger Segmente als der Chip T500, so daß für den Chip P276 auch deutlich weniger Zeit benötigt wird. Auch bei einem Vergleich der Beispielchips T1000 und P690 fällt dies deutlich auf. Dies wurde auch so erwartet, da sowohl bei der Extraktion als auch bei den meisten Fehlersuchprozeduren jedes Segment bzw. jedes normalisierte Segment betrachtet wird, wobei es unerheblich ist, wieviele Segmente zu wievielen Netzen gehören.
Der krasse Unterschied der Verhältnisse der Anzahl der Netze zur Anzahl der Segmente bei den zufällig generierten Chips einerseits und bei den der Praxis entnommenen Chips andererseits kommt daher, daß die Netze der zufällig generierten Chips meistens sehr viele Nachbarn miteinander verbinden, während die Netze der aus der Praxis entnommenen Chips oft nur sehr wenige Nachbarn verbinden. Das Verhältnis der Anzahl der Netze zur Anzahl der Segmente liegt im Durchschnitt bei den zufällig generierten Chips bei 0.04 und bei den der Praxis entnommenen Chips bei 0.1.
Die Anzahl der aus den Originalsegmenten erzeugten normalisierten Segmente ist bei den zufällig generierten Chips um 15-20 Prozent, bei den aus der Praxis stammenden Chips um 7-10 Prozent höher als die Anzahl der Originalsegmente. Dieser Wert ist jedoch hauptsächlich vom verwendeten Routing-Algorithmus abhängig und soll deshalb hier nicht weiter diskutiert werden. Bei Chips, deren Layout mit dem DKROUTER erzeugt und nicht nachoptimiert wurde, wird vom Programm EXTRACT also ca. 20 Prozent mehr Speicher für die normalisierten Segmente benötigt als für die Originalsegmente.

6.1.2 Ergebnisse der Fehlersuche im Layout

Nachdem der Zeit- und Speicherbedarf des Programms EXTRACT betrachtet wurde, soll nun das Ergebnis der Fehlersuche analysiert werden. In der Tabelle 6.2 sind dazu alle Fehler, die in den Beispiellayouts aufgetreten sind, zusammengefaßt. Es ist jeweils die Anzahl der Fehler angegeben. Fehler, die bei keinem der Beispielchips aufgetreten sind, wurden nicht in die Tabelle aufgenommen. Die einzelnen Spalten der Tabelle haben dabei folgende Bedeutungen:

- Spalte „Segmentkurzschl. (Layer 1/Layer 2)“: Anzahl der Kurzschlüsse zwischen Segmenten, für den ersten und den zweiten Layer
- Spalte „Kurzschlüsse an Vias“: Anzahl der durch Kontaktlöcher verursachten Kurzschlüsse
- Spalte „überflüssige Teilnetze“: Anzahl der überflüssigen Teilnetze

- Spalte „offene Seg. (A/E/AE)“: Anzahl der Segmente, die am Anfang, am Ende und am Anfang und Ende offen sind
- Spalte „DRC (x/y)“: Anzahl der x-Segmente und der y-Segmente, die die Design Rules verletzen (durch Zellen verlaufen)

Chip	Segmentkurzschl. (Layer 1/Layer 2)	Kurzschlüsse an Vias	überflüssige Teilnetze	offene Seg. (A/E/AE)	DRC (x/y)
T10	0/0	0	0	0/0/0	0/0
T20	0/0	0	0	0/0/0	0/0
T50	0/0	0	0	0/0/0	0/0
T100	0/0	0	0	0/0/0	0/0
T200	21/0	1	1	0/1/1	0/1
T500	0/0	0	3	1/1/3	0/0
T1000	0/0	0	10	2/2/10	0/0
P45	0/0	0	0	0/0/0	0/0
P276	0/0	0	0	0/0/0	0/0
P690	0/0	0	0	0/0/0	0/0

Tabelle 6.2: Ergebnisse des Programms EXTRACT (vom DKROUTER erzeugte LAYOUTS)

Bei der Betrachtung der Tabelle stellt man zunächst fest, daß die zufällig generierten Chips T10, T20, T50 und T100 frei von Fehlern im Layout sind. Dies muß jedoch nicht bedeuten, daß die extrahierten Netzlisten identisch mit den Originalnetzlisten sind, da bisher kein Netzlistenvergleich durchgeführt wurde. Auch die Beispielchips aus der Praxis (P45, P276 und P690) sind frei von Fehlern im Layout.
Es treten jedoch bei den größeren, zufällig generierten Chips T200, T500 und T1000 einzelne Fehler im Layout auf. Beim Chip T200 sind dies sogar schwere Fehler: 21 Segmentkurzschlüsse im ersten Layer, außerdem ein durch ein Kontaktloch verursachter Kurzschluß zwischen den Netzen 21 und 3. Auch gibt es ein y-Segment, das den durch eine Zelle belegten Bereich schneidet. Des weiteren treten noch einige unechte Fehler auf: ein überflüssiges Teilnetz, ein Segment, das am Anfang offen ist und ein Segment, das am Anfang und am Ende offen ist.
Bei den Chips T500 und T1000 treten unechte Fehler auf. Dabei handelt es sich um überflüssige Teilnetze und um offene Segmente.
Die Layoutfehler in den Chips T200, T500 und T1000 wurden dann näher untersucht. Dazu wurden zunächst beim Chip T200 die betroffenen Stellen mit dem Programm DISPLAY aus [Kla94] betrachtet.
Bei dem Chip T200 wurde dabei festgestellt, daß es ein Segment gibt, das einen Kurzschluß zwischen 21 Netzen verursacht, und außerdem den durch die Zelle 107 belegten Bereich schneidet. Dieses Segment ist ein y-Segment, das fälschlicherweise im ersten Layer liegt, obwohl der DKROUTER y-Segmente grundsätzlich in den zweiten Layer legt. Damit sind 21 x-Segment, die im ersten Layer liegen und alle zu verschiedenen Netzen gehören, kurzgeschlossen. Das fehlerhafte Segment gehört zu Netz 21, ist jedoch mit keinem anderen Segment aus diesem Netz verbunden, weshalb auch ein überflüssiges Teilnetz gemeldet wird. Außerdem wird dieses Segment als offen am Anfang und Ende gemeldet. Des weiteren gibt es ein Kontaktloch, das dieses Segment mit einem anderen, zu Netz 3 gehörigen Segment verbindet, weshalb es zu einem Kurzschluß an einem Kontaktloch kommt. Der Ausschnitt des Chips, in dem sich diese Fehler befinden, ist in Abbildung 6.1 zu sehen. Dabei sind Segmente, die sich im ersten Layer befinden, durchgezogen gezeichnet, und Segmente, die zum zweiten Layer gehören, gestrichelt gezeichnet. Man sieht deutlich das y-Segment, das im ersten Layer mehrere Kurzschlüsse verursacht und außerdem den durch die Zelle 107 belegten Bereich schneidet. Damit sind, bis auf ein offenes Segment, alle Layoutfehler im Layout des Chips T200 erklärt. Später erfolgt noch ein Netzlistenvergleich.

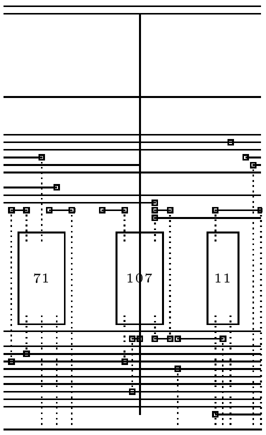

Abbildung 6.1: Beispielchip T200: Kurzschluß und Design-Rule-Fehler

Um die bei dem Chip T500 gemeldeten Fehler näher zu untersuchen, wurden in der Extraktionsdatei zunächst die Netze näher betrachtet, die in mehrere Teilnetze zerfallen sind. Dies sind die Netze 16 (2 Teilnetze) und 61 (4 Teilnetze). In der Netzliste des Netzes 16 gehören jedoch alle Nachbarn zum Teilnetz 1, das zweite Teilnetz ist also überflüssig und gehört damit zu den drei überflüssigen Teilnetzen. In der Netzliste des Netzes 61 gibt es hingegen einen Netznachbarn, der zu Teilnetz 4 gehört. Alle anderen Nachbarn gehören jedoch zu Teilnetz 1. Damit sind in diesem Netz die Teilnetze 2 und 3 überflüssig, womit auch die anderen beiden überflüssigen Teilnetze erklärt sind. Der Anschluß 3 der Zelle 245 hat keine Verbindung zu den anderen Nachbarn des Netzes 61. Das Netz 61 ist also unterbrochen, was einen schweren Fehler darstellt. Um den Grund für diese Netzunterbrechung festzustellen, wurde das Netz mit Hilfe des Programms DISPLAY betrachtet. Dabei wurde festgestellt, daß die Segmente, die die fehlenden Verbindungen herstellen sollen, zwar vorhanden sind, jedoch zwei Kontaktlöcher, die diese Segmente miteinander verbinden sollen, fehlen. Deshalb werden die betreffenden Segmente auch als offen gemeldet.
Beim Chip T1000 treten ähnliche Fehler auf wie beim Chip T500. Auch hier gibt es zehn überflüssige Teilnetze und die Betrachtung der Extraktionsdatei zeigte, daß es zwei Netze gibt, die mehr als ein Teilnetz enthalten. Dies sind die Netze 16 (6 Teilnetze) und 120 (8 Teilnetze). Ein Nachbar des Netzes 16, nämlich der Anschluß 1 der Zelle 580, ist mit dem Teilnetz 6 verbunden. Beim Netz 120 ist ebenfalls ein Nachbar, nämlich der Anschluß 2 der Zelle 639, mit dem Teilnetz 6 verbunden. Alle anderen Teilnetze sind überflüssig. Die Netze 16 und 120 sind also unterbrochen. Um die Ursache für diese Fehler festzustellen, wurde das Layout des Chips betrachtet, und zwar die Umgebungen der Zellen 580 und 639. Dabei wurde festgestellt, daß die Segmente, die die fehlenden Verbindungen herstellen sollen, zwar vorhanden sind, jedoch die Kontaktlöcher, die diese Segmente mit den anderen Segmenten verbinden sollen, fehlen. Deshalb werden auch die betreffenden Segmente als offen am Anfang und am Ende gemeldet.
Man sieht also, daß es bei Chips mit sehr vielen Zellen und sehr komplexen Netzen, wie die Chips T200, T500 und T1000, vereinzelt zu Layoutfehlern kommt, die mitunter schwere Fehler zur Folge haben. Aus einer Ursache resultieren oft viele Folgefehler.

6.1.3 Ergebnisse des Netzlistenvergleichs

Nach der Extraktion und Fehlersuche in den Beispiellayouts erfolgte der Netzlistenvergleich. Dazu wurden mit Hilfe des Programms COMPARE die bei der Extraktion erzeugten extrahierten Netzlisten mit den Originalnetzlisten der Beispielchips verglichen. Nur dann, wenn sowohl bei der Extraktion keine Layoutfehler und auch beim Netzlistenvergleich keine Unterschiede auftreten, kann man sicher sein, daß ein Layout genau die gegebenen Netzlisten implementiert.

Die Ergebnisse der Netzlistenvergleiche sind in der Tabelle 6.3 dargestellt.

Chip	Zeit (sec)	Übereinstimmungen	Fehlende Nachbarn	überzählige Nachbarn	unterbrochene Netze
T10	0.01	77 (100%)	0	0	0
T20	0.01	144 (100%)	0	0	0
T50	0.03	346 (100%)	0	0	0
T100	0.04	684 (100%)	0	0	0
T200	0.08	1358 (99.9%)	1 (0.1%)	0	1 (1.0%)
T500	0.18	3378 (100%)	0	0	2 (0.9%)
T1000	0.39	6745 (100%)	0	0	2 (0.8%)
P45	0.02	215 (100%)	0	0	0
P276	0.10	836 (100%)	0	0	0
P690	0.26	2024 (100%)	0	0	0

Tabelle 6.3: Ergebnisse des Programms COMPARE mit vom DKROUTER erzeugten Layouts

Dabei wurden für jeden Chip folgende Werte in die Tabelle aufgenommen:

- Spalte „Zeit (sec)“: Die Laufzeit des Programms COMPARE, in Sekunden
- Spalte „Übereinstimmungen“: Die Anzahl der Übereinstimmungen (alle Netze)
- Spalte „Fehlende Nachbarn“: Die Anzahl fehlender Netznachbarn (alle Netze)
- Spalte „Überzählige Nachbarn“: Die Anzahl überzähliger Netznachbarn (alle Netze)
- Spalte „unterbrochene Netze:“ Die Anzahl der Netze, die aus mehr als einem Teilnetz bestehen und damit möglicherweise unterbrochen sind.

Die Laufzeit des Programms ist in allen Fällen sehr kurz und durch die Anzahl der Zellen und der Netze bestimmt. Bei den Chips T10, T20, T50 und T100 treten keine Abweichungen von den Originalnetzlisten auf. Da bei diesen Chips auch keine Layoutfehler aufgetreten sind, kann man davon ausgehen, daß die Layouts genau die gegebenen Netzlisten fehlerfrei implementieren, der DKROUTER also in diesen Fällen fehlerfrei gearbeitet hat. Auch bei den der Praxis entnommenen Beispielchips (P45, P276 und P690) ist dies der Fall; es treten keine Abweichungen von den Originalnetzlisten und auch keine Layoutfehler auf, so daß auch diese Layouts fehlerfrei sind.
Bei den Chips T200, T500 und T1000 treten jedoch Fehler auf. Beim Chip T200 ist dies ein fehlender Nachbar und ein möglicherweise unterbrochenes Netz. Zunächst wurde die Ausgabedatei des Programms COMPARE betrachtet und festgestellt, daß es sich bei dem fehlenden Nachbarn um Anschluß 3 der Zelle 107 handelt, der eigentlich mit Netz 30 verbunden sein müßte. Um dies zu überprüfen, wurde Netz 30 mit dem Programm DISPLAY betrachtet. Dabei wurde festgestellt, daß im Layout ein Segment fehlt. Es handelt sich dabei um das Segment, das den Anschluß der Zelle mit einem anderen Segment des Netzes 30 verbinden müßte (s. Abbildung 6.2). Das Netz, das als möglicherweise unterbrochen gemeldet wird, ist Netz 21. Hier ist jedoch schon von der Extraktion her bekannt, daß das Netz ein überflüssiges Teilnetz enthält. Betrachtet man zusätzlich die Extraktionsdatei des Chips, stellt man außerdem fest, daß alle Nachbarn an einem Teilnetz hängen, das Netz also nicht unterbrochen sein kann. Es handelt sich bei dem überflüssigen Teilnetz also um das schon in Abbildung 6.1 gezeigte, überflüssige Segment. Die Aussage, daß ein Netz aus mehreren Teilnetzen besteht, kann nur dann dahingehend interpretiert werden, daß das Netz unterbrochen ist, wenn bekannt ist, daß die Teilnetze nicht überflüssig sind. Damit gibt es im Layout T200 also nicht nur mehrere schwere Layoutfehler, sondern auch ein unvollständiges Netz.
Beim Chip T500 treten zwei Netze auf, die aus mehr als einem Teilnetz bestehen und damit möglicherweise unterbrochen sind. Es handelt sich dabei um die Netze 16 und 61. Im Netz 16

handelt es sich um ein überflüssiges Teilnetz. Bei der Betrachtung der extrahierten Netzliste dieses Netzes wurde außerdem festgestellt, daß alle Nachbarn zu einem Teilnetz gehören, das Netz also nicht unterbrochen sein kann. Es handelt sich hier um ein überflüssiges Segment. Netz 61 ist jedoch tatsächlich unterbrochen. In der extrahierten Netzliste dieses Netzes gehört der Anschluß 3 der Zelle 245 zu einem anderen Teilnetz als die übrigen Nachbarn dieses Netzes. Damit enthält das Netz zwar alle Nachbarn, ist aber unterbrochen. Wie bei der Auswertung des Layouts bereits festgestellt wurde, ist das Netz aufgrund zweier fehlender Kontaktlöcher unterbrochen, so daß dieser Nachbar keine Verbindung zum übrigen Netz hat.
Im Layout des Chips T1000 treten ähnliche Fehler auf wie im Layout des Chips T500. Auch hier gibt es zwei Netze, die möglicherweise unterbrochen sind. Es handelt sich dabei um die Netze 16 und 120. Hier sind beide Netze tatsächlich unterbrochen. Die extrahierten Netzlisten beider Netze enthalten jeweils einen Nachbarn, der zu einem anderen Teilnetz gehört als die übrigen Nachbarn. Im Netz 16 ist dies der Anschluß 1 der Zelle 580, während es sich im Netz 120 um den Anschluß 2 der Zelle 639 handelt. Dies sind die Nachbarn, die – wie bereits festgestellt – aufgrund fehlender Kontaktlöcher nicht mit ihren Netzen verbunden sind.

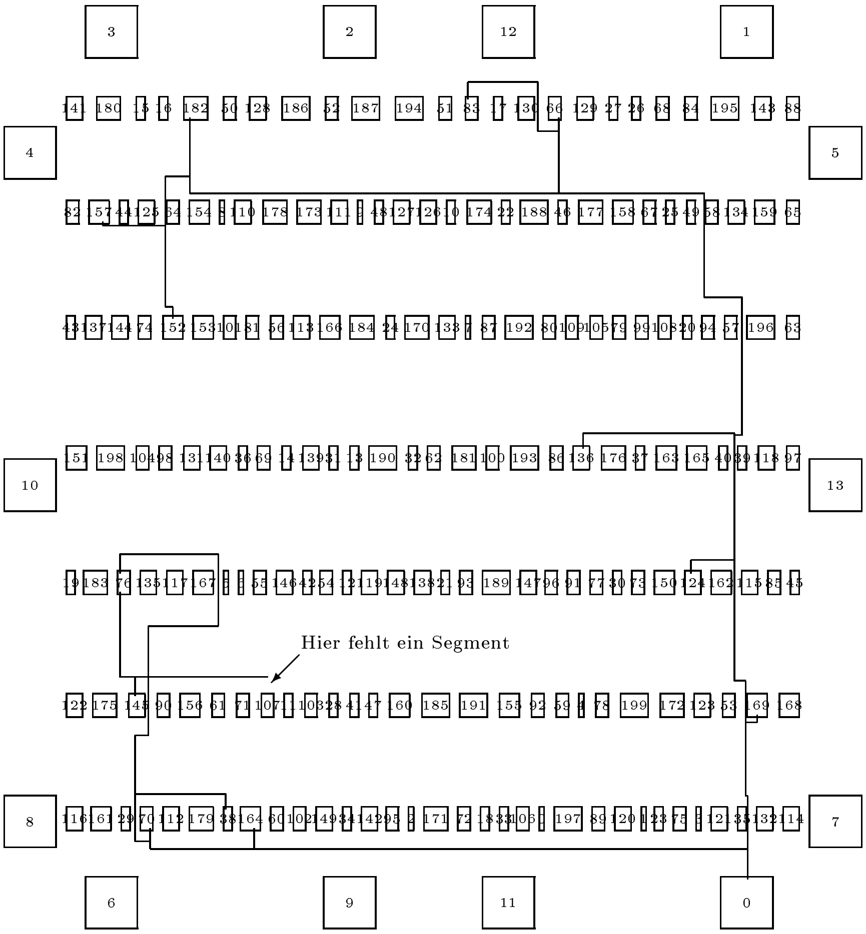

Abbildung 6.2: Beispielchip T200: Fehlender Nachbar im Netz 30 (Layer nicht unterschieden, Kontaktlöcher nicht gezeichnet; siehe auch Abbildung 6.8)

6.2 Mit NACHOPT nachoptimierte Layouts

In diesem Abschnitt werden nachoptimierte Layouts bearbeitet. Dazu wurden zunächst die vom DKROUTER erzeugten Layouts mit dem Programm NACHOPT aus [Kas95] einer Nachoptimierung unterzogen. Danach erfolgte wieder die Extraktion und Fehlersuche sowie der Netzlistenvergleich mit Hilfe der Programme EXTRACT und COMPARE.

6.2.1 Extraktion und Fehlersuche

Auch bei der Evaluierung des Programms EXTRACT mit nachoptimierten Layouts wurde wieder zuerst der Zeit- und Speicherbedarf des Programms untersucht. Die Ergebnisse sind in der Tabelle 6.4 dargestellt. Die Tabelle enthält die benötigte Rechenzeit und die Anzahl der Netze, Segmente, Vias und der normalisierten Segmente. Die Chips T500, T1000 und P690 konnten in diese Evaluierung nicht einbezogen werden, da der Nachoptimierer diese aufgrund interner Grenzen (maximal 300 Zellen) nicht verarbeiten konnte. Der Chip T200* ist ein Chip, bei dem das fehlerhafte Layout manuell korrigiert wurde (Entfernen des Segments, das mehrere Kurzschlüsse verursacht und Einfügen eines Segments und eines Kontaktloches, um Netz 30 zu vervollständigen).

Chip	Zeit (sec)	Netze	Segmente	Vias	norm. Segmente
T10	0.38	17	199	86	199
T20	0.83	32	435	196	435
T50	2.74	53	1170	651	1170
T100	6.48	75	2355	1378	2355
T200	17.47	103	5012	3050	5013
T200*	17.42	103	5012	3050	5012
P45	1.38	67	625	277	625
P276	10.00	299	2864	1541	2864

Tabelle 6.4: Zeit- und Speicherbedarf des Programms EXTRACT mit nachoptimierten Layouts

Auch hier sieht man, daß die Rechenzeit hauptsächlich von der Anzahl der Segmente abhängig ist. Ein deutlicher Unterschied gegenüber den nicht nachoptimierten Layouts ist jedoch, daß hier die Anzahl der normalisierten Segmente stets der Anzahl der Originalsegmente entspricht (die einzige Ausnahme ist der Chip T200, bei dem jedoch schon das nicht nachoptimierte Layout Fehler enthält). Dies bedeutet also, daß der Nachoptimierer NACHOPT ausschließlich Segmente erzeugt, die je zwei Knoten miteinander verbinden, also normalisierten Segmenten entsprechen. (Eine Normalisierung der Segmente, wie sie vom Programm EXTRACT durchgeführt wird, wird auch in [Kas95] gefordert.) Bei mit NACHOPT nachoptimierten Layouts wird also für die normalisierten Segmente genau so viel Speicher benötigt wie für die Originalsegmente. Wenn man sich außerdem die von EXTRACT erzeugten normalisierten Segmente ausgeben läßt, um diese dann mit den Originalsegmenten zu vergleichen, stellt man fest, daß diese den Originalsegmenten entsprechen. Dies hängt jedoch vom verwendeten Nachoptimierer ab und soll hier nicht weiter untersucht werden.

6.2.2 Ergebnisse der Fehlersuche im Layout

Das Ergebnis der Fehlersuche im Layout ist in der Tabelle 6.5 dargestellt. Es ist jeweils die Anzahl der Fehler jeder Fehlerklasse angegeben. Fehlerklassen, die bei keinem der Beispielchips aufgetreten sind, wurden nicht in die Tabelle aufgenommen. Damit enthält die Tabelle folgende Fehlerarten:

- Spalte „Segmentkurzschl. (Layer 1/Layer 2)“: Anzahl der Kurzschlüsse zwischen Segmenten, für den ersten und den zweiten Layer
- Spalte „Kurzschl. an Vias“: Anzahl der durch Kontaktlöcher verursachten Kurzschlüsse

- Spalte „überfl. Teilnetze“: Anzahl der überflüssigen Teilnetze
- Spalte „offene Seg. (A/E/AE)“: Anzahl der Segmente, die am Anfang, am Ende und am Anfang und Ende offen sind
- Spalte „DRC (x/y)“: Anzahl der x-Segmente und der y-Segmente, die die Design Rules verletzen (durch Zellen verlaufen)
- Spalte „äquiv. Vias“: Anzahl der äquivalenten Kontaktlöcher (mehrfach vorhandene Kontaktlöcher mit gleichen Koordinaten)
- Spalte „überfl. Vias“: Anzahl der überflüssigen Kontaktlöcher

Chip	Segmentkurzschl. (Layer 1/Layer 2)	Kurzschl. an Vias	überfl. Teilnetze	offene Seg. (A/E/AE)	DRC (x/y)	äquiv. Vias	überfl. Vias
T10	0/0	0	0	0/0/0	0/0	0	0
T20	0/0	0	0	0/0/0	0/0	0	0
T50	0/0	0	0	0/0/0	0/0	3	0
T100	0/0	0	0	0/2/0	0/0	3	1
T200	19/0	1	1	0/1/1	0/1	4	0
T200*	0/0	0	0	0/0/0	0/0	4	0
P45	0/0	0	0	0/0/0	2/0	0	0
P276	0/0	0	0	0/1/0	0/0	3	1

Tabelle 6.5: Ergebnisse des Programms EXTRACT (nachoptimierte LAYOUTS)

Betrachtet man die Tabelle, stellt man zunächst fest, daß die Chips T10 und T20 frei von Layoutfehlern sind. Bei größeren Chips (T50, T100, T200, P276) fällt zunächst auf, daß diese nach der Nachoptimierung einige äquivalente Kontaktlöcher enthalten, was sich durch Betrachten der nachoptimierten Layoutdatei auch leicht verifizieren läßt. Dieser Fehler ist harmlos, deutet jedoch auf einen möglichen Fehler in der Layout-Ausgaberoutine des Nachoptimierers hin. Auch gibt es bei den Chips T100 und P276 je ein überflüssiges Kontaktloch (also ein Kontaktloch, das keine Segmente miteinander verbindet). Dieser Fehler ist ebenfalls harmlos, könnte jedoch auch ein durch ein fehlendes oder zu kurzes Segment ausgelöster Folgefehler sein.
In den Layouts der Chips T100 und P276 treten Segmente auf, deren Enden offen sind. Bei der Betrachtung der Layouts mit dem Programm DISPLAY wurde festgestellt, daß es sich um Segmente handelt, die zu lang, aber nicht überflüssig, sind.
Bei dem Chip T200 treten die gleichen Fehler auf, wie bei der nicht nachoptimierten Version des Layouts (s. Tabelle 6.2), was nicht weiter verwunderlich ist, da der Nachoptimierer solche Fehler nicht erkennen kann. Der Nachoptimierer geht von einer fehlerfreien Eingabe aus. Bei dem Versuch, fehlerhafte Layouts nachzuoptimieren, verhält sich der Nachoptimierer nach dem Grundsatz „garbage in – garbage out“, was wohl auch legitim ist. Der Chip T200*, dessen Layoutfehler manuell korrigiert worden sind, wird vom Nachoptimierer fehlerfrei verarbeitet, es treten lediglich wieder vier äquivalente Kontaktlöcher auf.
Beim Chip P45 treten jedoch Fehler auf, die bisher bei keinem anderen Layout aufgetreten sind: Es gibt zwei x-Segmente, die den durch eine Zelle belegten Bereich schneiden, also die Design Rules von ESPRO verletzen. Die Betrachtung des Layouts ergab hierbei, daß zwei Anschlüsse einer Zelle vom Nachoptimierer an die Seiten der Zelle gelegt worden sind (s. Abbildung 6.3). Dieses Verhalten des Nachoptimierers ist wohl kein Fehler, sondern beabsichtigt, da dadurch Platz gespart wird. Zellanschlüsse dürfen in ESPRO jedoch nur von oben oder von unten mit vertikalen Segmenten angeschlossen werden, obwohl eine Technologie denkbar wäre, die auch die hier auftretende Variante zulassen würde.

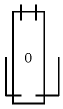

Abbildung 6.3: In ESPRO verbotene, aber technologisch durchaus denkbare Art des Zellanschlusses

6.2.3 Ergebnisse des Netzlistenvergleichs

Nach der Extraktion und Fehlersuche in den nachoptimierten Beispiellayouts erfolgte auch hier ein Netzlistenvergleich, um sicher sein zu können, daß auch nach der Nachoptimierung die Layouts genau die Originalnetzlisten implementieren. Dazu wurden mit dem Programm COMPARE die extrahierten Netzlisten der nachoptimierten Layouts mit den Originalnetzlisten verglichen. Die Ergebnisse dieser Vergleiche sind in der Tabelle 6.6 dargestellt. Die Tabelle ist dabei genau so aufgebaut wie die Tabelle 6.3.

Chip	Zeit (sec)	Übereinstimmungen	Fehlende Nachbarn	überzählige Nachbarn	unterbrochene Netze
T10	0.01	77 (100%)	0	0	0
T20	0.01	144 (100%)	0	0	0
T50	0.03	346 (100%)	0	0	0
T100	0.04	684 (100%)	0	0	0
T200	0.09	1358 (99.9%)	1 (0.1%)	0	1 (1.0%)
T200*	0.07	1359 (100%)	0	0	0
P45	0.03	215 (100%)	0	0	0
P276	0.11	836 (100%)	0	0	0

Tabelle 6.6: Ergebnisse des Programms COMPARE mit nachoptimierten Layouts

Die Laufzeit ist wieder in allen Fällen sehr kurz und durch die Anzahl der Netze und der Zellen bestimmt. Es treten hier nur beim Chip T200 zwei Fehler auf. Dies sind die gleichen Fehler wie bei der nicht nachoptimierten Version dieses Chips (s. Tabelle 6.3), da der Nachoptimierer hier eine Eingabe erhalten hat, die bereits fehlerhaft war. Der Chip T200*, bei dem vor der Nachoptimierung die Layoutfehler manuell korrigiert worden sind, wird fehlerfrei verarbeitet.

6.3 Mit NETZMIN nachoptimierte Layouts

In diesem Abschnitt werden die mit Hilfe des Nachoptimierers NETZMIN aus [Kas95] nachoptimierten Layouts bearbeitet. Während der Nachoptimierer NACHOPT versucht, die Anzahl der Kontaktlöcher und die für die Verdrahtung benötigte Fläche zu reduzieren, versucht NETZMIN, ein ausgewähltes Netz eines Layouts zu verkürzen. Deshalb wurde bei den vom DKROUTER erzeugten Layouts jeweils das längste Netz gesucht und mit NETZMIN verkürzt. Danach erfolgte die Extraktion und Fehlersuche sowie der Netzlistenvergleich mit Hilfe der Programme EXTRACT und COMPARE.

6.3.1 Extraktion und Fehlersuche

Bei der Evaluierung des Programms EXTRACT mit den in der Netzlänge nachoptimierten Layouts wurde zunächst wieder der Zeit- und Speicherbedarf untersucht. Die Ergebnisse dieser Untersuchung sind in der Tabelle 6.7 dargestellt. Die Tabelle enthält die benötigte Rechenzeit und die Anzahl der Netze, Segmente, Kontaktlöcher und der normalisierten Segmente. Auch in diese Evaluierung konnten die Chips T500, T1000 und P690 aufgrund interner Grenzen im Programm NETZMIN nicht einbezogen werden. Beim Chip T200* handelt es sich wieder um die manuell korrigierte Version des Chips T200 (s. 6.2.1).

Chip	Zeit (sec)	Netze	Segmente	Vias	norm. Segmente
T10	0.44	17	204	132	204
T20	0.94	32	441	301	441
T50	2.74	53	1180	841	1180
T100	6.37	75	2385	1697	2385
T200	17.47	103	5037	3678	5038
T200*	17.42	103	5037	3679	5037
P45	1.48	67	637	431	637
P276	10.16	299	2901	2090	2901

Tabelle 6.7: Zeit- und Speicherbedarf des Programms EXTRACT mit nachoptimierten Layouts (NETZMIN)

Es ergibt sich hier ein ähnliches Bild wie in 6.2.1. Auch hier ist die Rechenzeit hauptsächlich von der Anzahl der Segmente abhängig, und auch die Anzahl der normalisierten Segmente entspricht der Anzahl der Originalsegmente. Der Nachoptimierer NETZMIN erzeugt also – wie der Nachoptimierer NACHOPT – ausschließlich Segmente, die je zwei Knoten miteinander verbinden, also normalisierten Segmenten entsprechen. Man kann auch hier feststellen, daß die von EXTRACT erzeugten normalisierten Segmente den von NETZMIN generierten Segmenten entsprechen. Für die normalisierten Segmente wird von EXTRACT also genau so viel Speicher benötigt wie für die Originalsegmente.

6.3.2 Ergebnisse der Fehlersuche im Layout

Nach der Untersuchung des Zeit-und Speicherbedarfs von EXTRACT wurde das Ergebnis der von EXTRACT durchgeführten Fehlersuche analysiert. Dazu sind alle Fehler, die in den Layouts aufgetreten sind, in der Tabelle 6.8 zusammengefaßt. Fehlerklassen, die bei keinem der Layouts aufgetreten sind, wurden nicht in die Tabelle aufgenommen. Damit enthält die Tabelle folgende Fehlerarten:

- Spalte „Segmentkurzschl. (Layer 1/Layer 2)“: Anzahl der Kurzschlüsse zwischen Segmenten, für den ersten und den zweiten Layer
- Spalte „Kurzschl. an Vias“: Anzahl der durch Kontaktlöcher verursachten Kurzschlüsse
- Spalte „überfl. Teilnetze“: Anzahl der überflüssigen Teilnetze
- Spalte „offene Seg. (A/E/AE)“: Anzahl der Segmente, die am Anfang, am Ende und am Anfang und Ende offen sind
- Spalte „DRC (x/y)“: Anzahl der x-Segmente und der y-Segmente, die die Design Rules verletzen (durch Zellen verlaufen)

Bei der Betrachtung der Tabelle 6.8 fällt auf, daß die Layouts T10 und T20 unechte Fehler enthalten: Ein bzw. zwei Segmente, die am Anfang offen und überflüssig sind. Andere Fehler treten bei diesen Chips nicht auf. Diese Fehler können für sich allein, aber auch als Folgefehler eines unterbrochenen Netzes auftreten. Um dieses zu untersuchen, müssen die Extraktionsdateien der beiden Chips näher betrachtet werden. Beim Chip T10 gehören die beiden offenen Segmente zum Netz 12. Es wurde deshalb die extrahierte Netzliste dieses Netzes betrachtet. Dabei wurde festgestellt, daß

Chip	Segmentkurzschl. (Layer 1/Layer 2)	Kurzschl. an Vias	überfl. Teilnetze	offene Seg. (A/E/AE)	DRC (x/y)
T10	0/0	0	0	2/0/0	0/0
T20	0/0	0	0	1/0/0	0/0
T50	0/0	0	0	0/0/0	0/0
T100	0/0	0	0	0/0/0	0/0
T200	21/0	1	1	0/1/1	0/1
T200*	0/0	0	0	0/0/0	0/0
P45	0/0	0	0	0/0/0	0/0
P276	0/0	0	0	0/0/0	0/0

Tabelle 6.8: Ergebnisse des Programms EXTRACT (mit NETZMIN nachoptimierte LAYOUTS)

das Netz aus zwei Teilnetzen besteht, wobei der Pad P8 am zweiten Teilnetz hängt, während die anderen Nachbarn dieses Netzes zum ersten Teilnetz gehören. Das Netz 12 ist also unterbrochen. Um den Grund für diese Netzunterbrechung feststellen zu können, wurde das Layout mit dem Programm DISPLAY betrachtet. Dabei wurde festgestellt, daß das Segment, das den Pad P8 mit dem übrigen Netz verbinden soll, vorhanden ist, jedoch das Kontaktloch, das dieses Segment mit einem anderen Segment dieses Netzes verbinden soll, fehlt. Deshalb werden diese beiden Segmente als offen und überflüssig gemeldet. Es handelt sich also um Folgefehler, die durch ein fehlendes Kontaktloch ausgelöst worden sind (Abbildung 6.4).

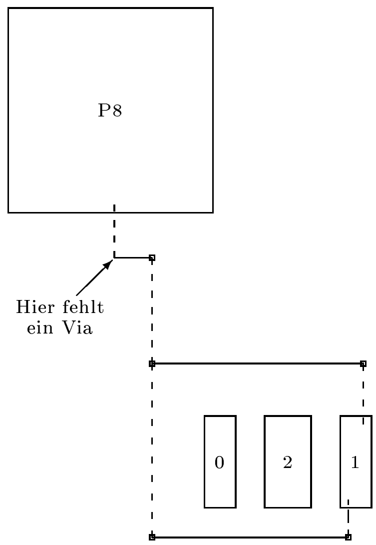

Abbildung 6.4: Ausschnitt aus T10, Netz 12: Die Verbindung zu P8 ist aufgrund eines fehlenden Kontaktloches unterbrochen

Beim Layout des Chips T20 tritt ebenfalls ein offenes und überflüssiges Segment auf. Das Segment gehört zu Netz 0. Betrachtet man jedoch die extrahierte Netzliste dieses Netzes, stellt man fest, daß diese nur aus einem Teilnetz besteht. Es kann sich also nicht um ein unterbrochenes Netz handeln. Möglich ist jedoch, daß eine Verbindung zu einem anderen Netznachbarn fehlt, was aber erst beim Netzlistenvergleich festgestellt werden kann.

6.3.3 Ergebnisse des Netzlistenvergleichs

Auch hier erfolgte nach der Extraktion und Fehlersuche in den Layouts ein Netzlistenvergleich, um sicher sein zu können, daß auch nach der Optimierung der Netzlängen die Layouts genau die Originalnetzlisten implementieren. Dazu wurden mit Hilfe des Programms COMPARE die extrahierten Netzlisten mit den Originalnetzlisten verglichen. Die Ergebnisse dieser Vergleiche sind in der Tabelle 6.9 dargestellt. Diese Tabelle ist genau so aufgebaut wie die Tabelle 6.3.

Chip	Zeit (sec)	Übereinstimmungen	Fehlende Nachbarn	überzählige Nachbarn	unterbrochene Netze
T10	0.01	77 (100%)	0	0	1 (5.9%)
T20	0.01	144 (100%)	0	0	0
T50	0.03	346 (100%)	0	0	0
T100	0.05	684 (100%)	0	0	0
T200	0.11	1358 (99.9%)	1 (0.1%)	0	1 (1.0%)
T200*	0.11	1359 (100%)	0	0	0
P45	0.03	215 (100%)	0	0	0
P276	0.12	836 (100%)	0	0	0

Tabelle 6.9: Ergebnisse des Programms COMPARE mit nachoptimierten Layouts (NETZMIN)

Man sieht, daß bei dem Chip T10 ein Netz als möglicherweise unterbrochen gemeldet wird. Es handelt sich dabei um das Netz 12, das schon bei der Analyse der Layoutfehler behandelt wurde. Interessant ist, daß sich beim Chip T20 keine Abweichungen zeigen. Damit steht fest, daß die bei der Fehlersuche aufgetretenen unechten Fehler keine Folgefehler sind, sondern tatsächlich für sich allein auftreten. Betrachtet man das Netz 0 dieses Chips mit dem Programm DISPLAY, stellt man fest, daß es ein Segment gibt, daß nur mit einem Ende mit diesem Netz verbunden und an nichts weiterem angeschlossen ist.
Bei dem Chip T200 (fehlerhaftes Layout als Eingabe für NETZMIN) treten die gleichen Fehler auf, wie nach der Nachoptimierung mit dem Programm NACHOPT. Auch NETZMIN kann bei einem fehlerhaften Eingabelayout keine fehlerfreie Ausgabe liefern und verhält sich hier nach dem Grundsatz "garbage in – garbage out". Mit dem manuell korrigierten Beispielchip T200* liefert der Nachoptimierer ein korrektes Ergebnis.

6.4 Weitere Tests des Verifikationstools

Um die Fehlersuchroutinen des Programms EXTRACT während dessen Entwicklungsphase testen zu können, wurden einige der verwendeten Beispiellayouts bewußt mit Fehlern versehen. Die Ergebnisse dieser Tests sollen hier kurz dargestellt werden.
Zunächst wurde bei dem vom DKROUTER erzeugten Beispiellayout T100 ein Kontaktloch entfernt. Außerdem wurde ein Segment vom ersten in den zweiten Layer und ein Segment vom zweiten in den ersten Layer verlegt. Das so manipulierte Layout wurde nun mit EXTRACT und COMPARE untersucht. Die Ergebnisse konnten in der grafischen Darstellung mit dem Programm DISPLAY nachvollzogen werden. Im einzelnen handelte es sich dabei um folgende Fehler:

- 9 überflüssige Kontaktlöcher (verursacht durch die Verlegung der beiden Segmente)
- 3 Segmentkurzschlüsse im ersten Layer (verursacht durch die Verlegung eines Segments aus dem zweiten in den ersten Layer)
- 2 am Anfang offene Segmente (verursacht durch Entfernen eines Kontaktloches)
- 1 unterbrochenes Netz (verursacht durch Entfernen eines Kontaktloches)

Durch das bewußte Einbringen von Fehlern in Layoutdateien konnten die Programme EXTRACT und COMPARE gut ausgetestet werden. Es wurden während der Entwicklung der Software noch weitere solche Tests durchgeführt, die hier jedoch nicht im einzelnen beschrieben werden sollen, da diese genau so ablaufen, wie der hier exemplarisch beschriebene Test.

6.5 Untersuchungen zur Komplexität

Nach der Evaluierung der Programme EXTRACT und COMPARE mit verschiedenen Beispiellayouts wurde aufgrund der Messungen der CPU-Zeit die Komplexität der Programme ansatzweise abgeschätzt.
Die vom Programm EXTRACT benötigte CPU-Zeit hängt von der Anzahl der Segmente bzw. der normalisierten Segmente ab. Betrachtet man die Werte in den Tabellen 6.1, 6.4 und 6.7, so kann man eine fast lineare Abhängigkeit feststellen. Die bei der Extraktion und Fehlersuche in ESPRO-Layouts auftretenden Probleme sind also in P enthalten. Dies ist vor allem darauf zurückzuführen, daß in einer ESPRO-Layoutdatei alle Segmente mit der Nummer des Netzes versehen sind, zu dem sie gehören. Dadurch, daß diese eigentlich redundante Information hier gut ausgenutzt wurde, konnte z. B. das sonst als NP-schwer geltende Problem der Extraktion auf ein in P liegendes Problem reduziert werden.
Diese Überlegungen treffen auch für den Netzlistenvergleich zu. Durch Ausnutzung der redundanten Informationen ist bekannt, welche extrahierte Netzliste mit welcher Originalnetzliste verglichen werden muß. Damit tritt das NP-vollständige Isomorphieproblem hier nicht auf, der Netzlistenvergleich reduziert sich fast auf einen einfachen Dateivergleich. Dieser kann leicht in linearer Zeit durchgeführt werden, was die Werte in den Tabellen 6.3, 6.6 und 6.9 bestätigen. Die CPU-Zeit ist dabei von der Anzahl der Netznachbarn abhängig.

6.6 Ergebnisse der Evaluierung

Die Evaluierung hat gezeigt, daß ein Verifikationstool für ESPRO notwendig ist, da es nicht möglich oder extrem zeitaufwendig ist, Layouts von Chips mit mehr als zwanzig Zellen (s. Abbildungen 6.5, 6.6, 6.7, 6.8 und 2.1) durch Betrachten des Layouts manuell zu verifizieren. Mit Hilfe des Verifikationstools ist dies hingegen innerhalb weniger Sekunden bis Minuten möglich. Eine Verifikation der Layouts ist auch nötig, da es unmöglich ist, Programme von der Größe und Komplexität des Routers DKROUTER oder der Nachoptimierer NACHOPT und NETZMIN auf andere Art (z. B. formal) zu verifizieren. Auch bei der Fehlersuche in Routern oder Nachoptimierern kann das Verifikationstool gut eingesetzt werden, um Hinweise zu erhalten, wo ein Router oder Nachoptimierer fehlerhaft sein könnte.
Das Programm DKROUTER arbeitet zuverlässig, solange die Chips nicht zu groß und komplex (T500 und T1000) werden. Bei großen, komplexen Chips wie T200, T500 und T1000 treten vereinzelt Fehler auf, die zur Nichtfunktion der Schaltung führen würden.
Das Programm NACHOPT arbeitet ebenfalls zuverlässig, erzeugt jedoch mitunter kleine, aber harmlose Layoutfehler. Außerdem scheint das Programm eine Optimierungsregel zu verwenden, die zu in ESPRO unzulässigen Konstrukten führt (Zellanschlüsse von der Seite, Abb. 6.3). Die internen Grenzen des Programms dürfen nicht überschritten werden (maximal 300 Zellen), da sonst zwar keine Fehlermeldung erzeugt, aber ein völlig fehlerhaftes Layout ausgegeben wird (es erschien nicht sinnvoll, dieses in die Evaluierung einzubeziehen). Größere Layouts (T500, T1000, P690) konnten deshalb nicht in die Evaluierung einbezogen werden, so daß sich nicht feststellen läßt, wie sich der Nachoptimierer bei großen, komplexen Chips verhält.
Das Programm NETZMIN erzeugt mitunter schwere Layoutfehler. Dies trat gerade bei dem kleinsten Layout auf. Es scheint, als ob eine Optimierungsregel falsch implementiert ist, so daß beim Erzeugen bestimmter Strukturen Kontaktlöcher nicht gesetzt werden. Die internen Grenzen (maximal 300 Zellen) dürfen auch bei diesem Programm nicht überschritten werden, da dann keine Fehlermeldung erzeugt, aber ein völlig fehlerhaftes Layout ausgegeben wird. Die größeren Beispiel-

layouts (T500, T1000, P690) konnten deshalb auch bei diesem Programm nicht in die Evaluierung einbezogen werden.

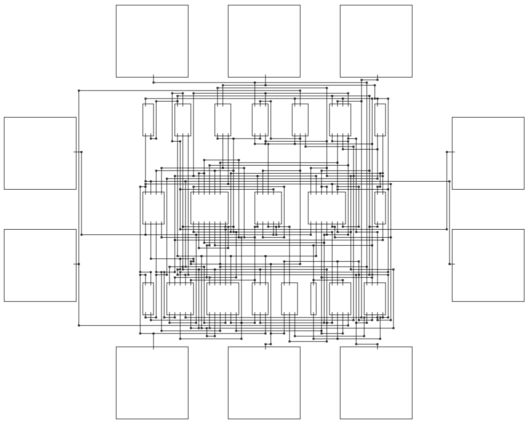

Abbildung 6.5: Der Beispielchip T20

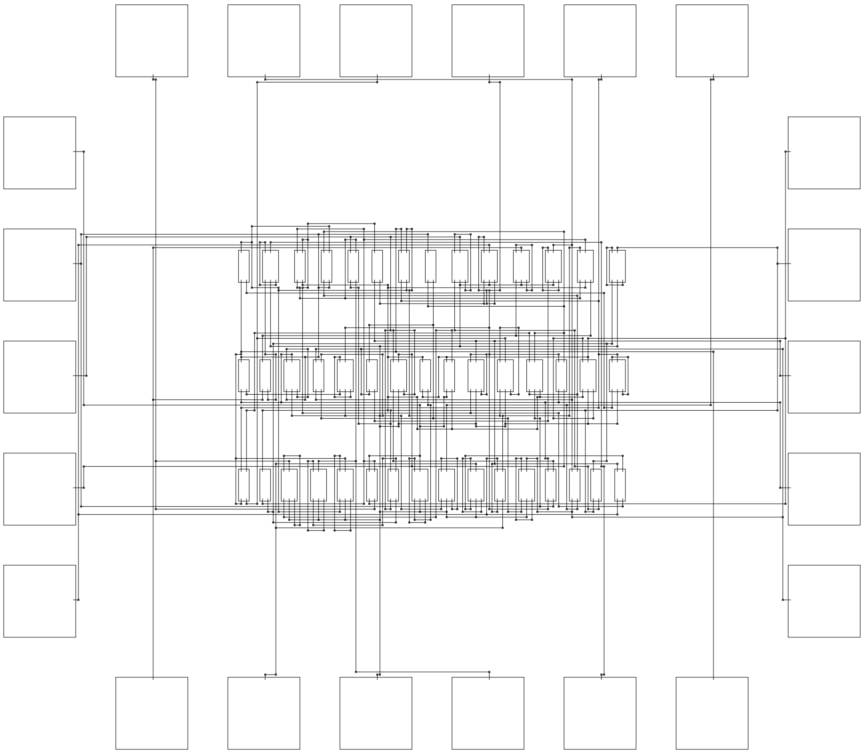

Abbildung 6.6: Der Beispielchip P45

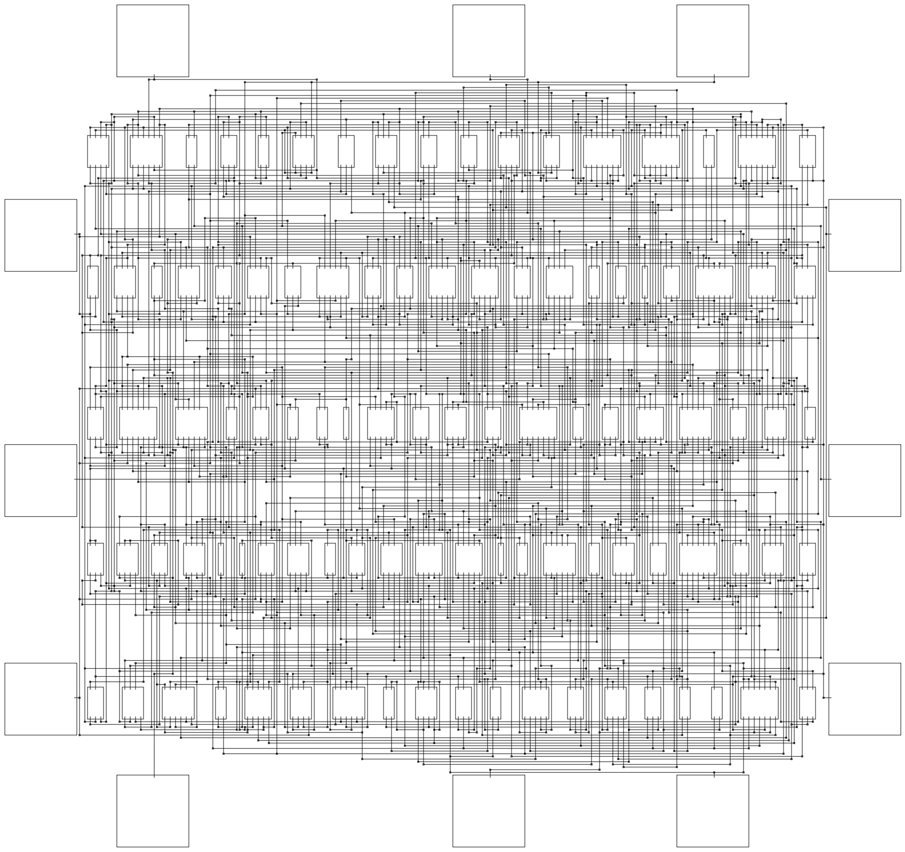

Abbildung 6.7: Der Beispielchip T100

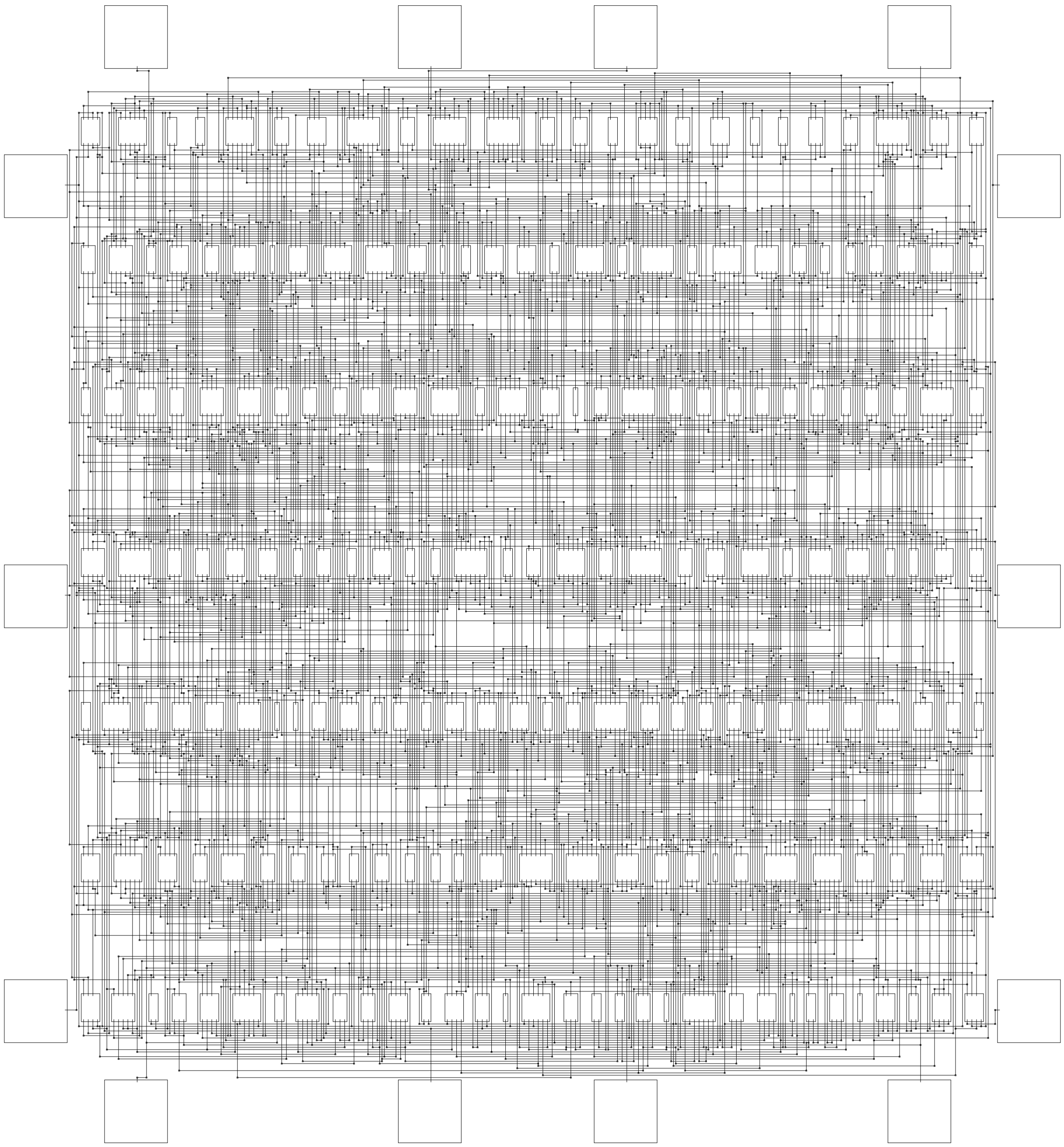

Abbildung 6.8: Der Beispielchip T200

Kapitel 7

Zusammenfassung und Ausblick

In dieser Arbeit wurde ein Tool entwickelt und vorgestellt, das die Verifikation von Standardzell-Entwürfen ermöglicht, die mit dem System ESPRO erstellt worden sind. Außerdem wurde das Verifikationstool an einigen Beispielschaltungen erprobt und es wurden die Ergebnisse dieser Evaluierung vorgestellt. Das Hauptaugenmerk war dabei jedoch auf die Entwicklung des Tools, von der Analyse möglicher Fehlerarten in Standardzell-Entwürfen bis hin zur Konzeption und Implementierung der Algorithmen, gerichtet.
Nach der Analyse der einzelnen Fehlerarten und deren Auswirkungen wurde für jede Fehlerart ein Algorithmus entwickelt, um solche Fehler finden zu können. Diese Algorithmen wurden anschließend implementiert, so daß nun ein Verifikationstool für ESPRO zur Verfügung steht.
Durch die offene Architektur des ESPRO-Systems war es leicht möglich, das Verifikationstool in das Gesamtsystem einzubinden. Dazu war nur die Kenntnis der ESPRO-Dateiformate von Basisdateien und Layoutdateien erforderlich, nicht jedoch der internen Arbeitsweisen der anderen ESPRO-Tools. Für das Ergebnis des Extraktors wurde ein eigenes Format (Extraktionsdateien) entwickelt, das dem der Basisdateien sehr ähnlich ist. Das System weist also eine außerordentlich hohe Flexibilität und Erweiterbarkeit auf, was hier gut ausgenutzt werden konnte. Das Verifikationstool kann für sämtliche Layouts eingesetzt werden, die mit dem ESPRO-System erstellt wurden.
Bei der Implementierung und auch bei der Evaluierung hat es sich als günstig erwiesen, das Verifikationstool in zwei Programme aufzuteilen. Das Programm EXTRACT, das die Extraktion und die Fehlersuche durchführt, sowie das Programm COMPARE für den anschließenden Netzlistenvergleich. Es ist sinnvoll, die Extraktion und die Fehlersuche in einem Programm unterzubringen, da diese auf denselben Daten, nämlich den Layoutdaten, arbeiten. Auch ließen sich Extraktion und Fehlersuche nicht immer vollständig voneinander trennen, da einige Fehler bereits bei der Extraktion entdeckt werden. Der Netzlistenvergleich jedoch benötigt die Layoutdaten nicht, sondern arbeitet mit dem Ergebnis der Extraktion. So ließ sich der Netzlistenvergleich gut von der Extraktion und Fehlersuche trennen und in einem eigenen Programm unterbringen. Das Programm COMPARE dient damit praktisch der weiteren Auswertung der vom Programm EXTRACT gelieferten Ergebnisse.
Die Evaluierung des Tools hat gezeigt, daß ein Verifikationstool notwendig war. Die Ergebnisse haben gezeigt, daß die bisher für das ESPRO-System zur Verfügung stehenden Router und Nachoptimierer noch Fehler enthalten. Das Verifikationstool kann nun gut als Debug-Hilfsmittel eingesetzt werden, um diese Fehler zu erkennen und zu beseitigen. Dies bleibt jedoch den Entwicklern dieser Tools überlassen. Das Interesse seitens der Entwickler der Router und Nachoptimierer an diesem Verifikationstool ist deshalb entsprechend hoch. Nur durch die automatische Überprüfung der Layouts können die Entwickler der Router und Nachoptimierer überhaupt auf Fehler in ihren Programmen hingewiesen werden, da die manuelle Überprüfung schon bei kleineren Layouts nahezu unmöglich ist. Fehler in einem Programm können jedoch nur dann gefunden werden, wenn bekannt ist, wie sich diese auswirken. Eines der Haupteinsatzgebiete des Verifikationstools wird deshalb die Unterstützung bei der Entwicklung und beim Debuggen von Verdrahtungs- und

Nachoptimierungstools sein.
Ein weiteres Haupteinsatzgebiet ist die Kontrolle der Ergebnisse bei der Verdrahtung und Nachoptimierung, um mehr Sicherheit beim Vergleich der Entwurfswerte von Layouts zu erzielen. Entwurfswerte von Standardzell-Layouts sind nur dann wirklich vergleichbar, wenn diese von fehlerfreien Layouts stammen. Dies kann nur durch eine automatisierte Layoutkontrolle, wie sie dieses Verifikationstool ermöglicht, erreicht werden.
Mit dem Verifikationstool wurde also eine Lücke in ESPRO geschlossen. Dies führt zu einer weiteren Abrundung des Gesamtsystems. Die Ergebnisse, die das System liefert, sind jetzt besser überprüfbar. Es kann zwar keine hundertprozentige Sicherheit gewährleistet werden, da auch die Implementierung des Verifikationstools Fehler enthalten kann. Die Situation ist jedoch sehr viel besser als vorher, wo überhaupt keine Überprüfung der Ergebnisse vorgenommen wurde. Bisher konnten alle Fehler, die vom Verifikationstool bei der Evaluierung gemeldet wurden, in der grafischen Darstellung der Layouts nachvollzogen werden, es gab also keine Falschmeldungen (*false positives*). Die Fehlersuchalgorithmen konnten außerdem getestet werden, indem von Hand Layouts erstellt wurden, die bewußt eingebrachte Fehler enthielten.
Insgesamt läßt sich also ein positives Fazit bezüglich des entwickelten Tools und dessen Evaluierung ziehen. Auch große Layouts, deren manuelle Überprüfung als sehr zeitaufwendig bis unmöglich anzusehen ist, können innerhalb weniger Minuten verifiziert werden. Die bisherigen Ergebnisse haben also die an das Verifikationstool gestellten Erwartungen vollständig erfüllt.
Im Rahmen zukünftiger Arbeiten können auch noch einige weitere Aktivitäten durchgeführt werden, die den Rahmen dieser Diplomarbeit bei weitem gesprengt hätten. Hier sollen nur einige Beispiele genannt werden, die Liste solcher Aktivitäten läßt sich wahrscheinlich beliebig fortsetzen.

- Evaluierung anhand weiterer Beispiele: Die im Rahmen dieser Arbeit durchgeführte Evaluierung konnte aus Zeitgründen nur anhand einiger weniger Beispiellayouts erfolgen. Hier wäre es wünschenswert, die Evaluierung mit weiteren Beispielen fortzusetzen. So kann der Prozess der Layouterstellung beeinflußt werden, indem beim Placement und beim Routing die Parameter der Plazierungs- und Verdrahtungstools verändert werden. Außerdem wurde nur die Kombination GDPLACER mit DKROUTER (beide aus [Kla94]) verwendet, es stehen jedoch noch weitere Plazierungs- und Verdrahtungsprogramme zur Verfügung. Da GDPLACER und DKROUTER aufeinander abgestimmt sind, sollte z. B. einmal ein Plazierungstool verwendet werden, das nicht speziell auf DKROUTER abgestimmt ist (z. B. TWPlacer aus [Kah95]). So ergibt sich dann eine Vielzahl weiterer Layouts, mit denen die Evaluierung des Verifikationstools weiter fortgesetzt werden kann.

- "Ursachenforschung": Wenn bei der Fehlersuche sehr viele gleichartige Fehler auftreten, so kann es sein, daß alle diese Fehler eine einzige Ursache haben, also alle Folgefehler eines einzigen anderen Fehlers sind. So wurden beispielsweise alle Kurzschlüsse, die im Layout des Chips T200 auftreten, durch ein einziges falsch gelegtes Segment verursacht, was jedoch erst bei der manuellen Nachbetrachtung der Fehler festgestellt werden kann. Solche Verkettungen von Fehlern müßten einer weiteren Untersuchung unterzogen werden. Dann sollte versucht werden, diese Untersuchung zu algorithmisieren, so daß bei der Fehlersuche automatisch Hinweise auf die Ursachen von Fehlern (z. B. ein Segment, das mehrere Kurzschlüsse verursacht) gegeben werden können. So kann die Notwendigkeit der manuellen Nachbetrachtung von Layoutfehlern reduziert werden.

- Komplexitätsbetrachtungen: Die Komplexität der im Rahmen dieser Arbeit entwickelten Algorithmen wurde nur anhand der Evaluierungsergebnisse (Rechenzeiten für die verschiedenen Beispielchips) abgeschätzt. Diese Werte beziehen sich jedoch auf das gesamte Tool und nicht auf die einzelnen Algorithmen und sind außerdem ungenau, da empirisch ermittelt. Es wäre eventuell wünschenswert, die Komplexitäten aller einzelnen Teile des Verifikationstools zu untersuchen und eventuell auch noch zu verbessern. Dazu müßten genauere Komplexitätsanalysen der einzelnen Algorithmen durchgeführt werden.

- Debugging der Router und Nachoptimierer: Aufgrund der Ergebnisse der Evaluierung ergeben sich Hinweise, an welchen Stellen der Router und die Nachoptimierer fehlerhaft sein könnten. Nun sollten diese Hinweise verwendet werden, die Programmfehler zu finden und zu beseitigen, also das Debugging aufgrund der durch die Verifikation gewonnen Hinweise durchzuführen. Dies war jedoch nicht das Ziel dieser Arbeit. Es hätte eine den zeitlichen Rahmen weitgehend sprengende Einarbeitung in die internen Funktionsweisen der Programme erfordert, da sonst nur wage Vermutungen geäußert werden können. Außerdem sollte das Debugging der Programme am besten von deren Entwicklern selbst durchgeführt werden, da diese ihre Programme kennen und keine weitere Einarbeitung benötigen. Für andere Personen dürfte die Einarbeitung in den fremden Sourcecode zwar möglich, aber zu zeitaufwendig sein.

Der Einsatz von Placement- und Routing-Heuristiken ist außerdem nicht ausschließlich auf die Plazierung und Verdrahtung bei integrierten Schaltungen beschränkt. Vielmehr gibt es eine Vielzahl von anderen Anwendungsgebieten. So werden Plazierungsheuristiken auch in der Lagerhaltung eingesetzt, und Routing findet z. B. auch in der Optimierung von Wegen und Fahrplänen Anwendung. Auch bei solchen Anwendungen von Placement und Routing ist eine Verifikation der Ergebnisse wünschenswert, da auch hier oft verschiedene Verfahren nebeneinander benutzt werden und die Qualität der Ergebnisse verglichen wird. Auch hier sind nur Ergebnisse vergleichbar, die korrekt sind. Die Verifikation bei Routingproblemen kann in ähnlicher Weise durchgeführt werden wie in dieser Arbeit geschehen: durch eine Umkehrung des Routing und Vergleich mit den ursprünglichen Vorgaben. So ist also auch der Einsatz eines Verifikationstools, wie es in dieser Arbeit vorgestellt wurde, nicht allein auf den Entwurf integrierter Schaltungen beschränkt, sondern das Tool kann auch an andere Problemstellungen angepaßt werden, bei denen Placement und vor allem Routing zur Anwendung kommt.

Anhang A

Beschreibung der Programme

In diesem Kapitel wird die im Rahmen der Diplomarbeit entwickelte Software beschrieben. Dabei steht die Bedienung der Programme EXTRACT und COMPARE im Vordergrund. Es werden jedoch auch Hinweise zum Sourcecode, insbesondere zur Portierung auf andere Systeme, gegeben. Außerdem wird die Anwendung der Software in einer Beispielsitzung erläutert.

A.1 EXTRACT

Die Algorithmen zur Extraktion und zur Fehlersuche im Layout sind in dem Programm EXTRACT implementiert, welches hier beschrieben wird. Zur Zeit steht dieses Programm für die Betriebssysteme RISC OS, UNIX (SunOS) und MS-DOS zur Verfügung und liegt der Arbeit auf Diskette bei.

A.1.1 Bedienung des Programms

Nach dem Start des Programms durch Eingabe von `Extract` erscheinen auf dem Bildschirm zunächst folgende Meldungen:

```
EXTRACT - Extraktor für ESPRO-Layoutdateien
Version vom (Datum)

Name der Layoutdatei:
```

Es ist nun der Name einer ESPRO-Layoutdatei einzugeben. Die Erweiterungen der Dateinamen (z. B. .LAY, .EXT, .ERR, .LST, .CHP) sind stets mit anzugeben (bei Layoutdateien normalerweise .LAY). Nun fragt das Programm nach der *Extraktionsdatei*. Dies ist die erste Ausgabedatei des Programms.

```
Name der Extraktionsdatei (1. Ausgabedatei):
```

In diese Datei werden die extrahierten Netzlisten hineingeschrieben. Extraktionsdateien sollten die Namenserweiterung .EXT bekommen. Danach muß der Name der *Fehlerdatei* eingegeben werden:

```
Name der Fehlerdatei (2. Ausgabedatei):
```

In dieser Datei werden alle bei der Fehlersuche erkannten Fehler protokolliert. Der Name der Datei sollte mit der Erweiterung .ERR versehen werden. Nun muß die Warnstufe festgelegt werden:

```
Warnstufe:
  0 - Fehler: nur Anzahl; Warnungen: nur Anzahl
  1 - Fehler: vollständig; Warnungen: nur Anzahl
  2 - Fehler: vollständig; Warnungen: vollständig
  3 - Konfigurationsdatei verwenden
```

Die Warnstufe bestimmt, ob bei Fehlermeldungen (schweren Fehlern) und Warnungen (unechten Fehlern) nur die Anzahl der Fehler in der jeweiligen Fehlerklasse ausgebeben werden soll, oder ob diese vollständig ausgegeben werden sollen. Bei Warnstufe 0 wird bei allen Arten von Fehlern nur jeweils deren Anzahl ausgegeben. Bei Warnstufe 1 werden alle schweren Fehler vollständig ausgegeben, jedoch nur die Anzahl der unechten Fehler. Bei Warnstufe 2 werden alle Fehler vollständig ausgegeben. Bei Warnstufe 3 muß im aktuellen Verzeichnis eine Datei mit dem Namen EXTRACT.CFG vorhanden sein. In der Konfigurationsdatei kann für jede Fehlerart einzeln eingestellt werden, ob diese vollständig ausgegeben werden soll oder nicht.
Als letztes muß festgelegt werden, ob das normalisierte Layout ausgegeben werden soll. Dies ist normalerweise nicht nötig, kann jedoch z. B. für die Nachoptimierer aus [Kas95] nützlich sein.

```
Soll das normalisierte Layout ausgegeben werden (J/N)?
```

Wird diese Frage mit `J` beantwortet, wird nach dem Namen der normalisierten Layoutdatei gefragt:

```
Name der normalisierten Layoutdatei (3. Ausgabedatei):
```

Hier ist dann der Name einzugeben. Damit sind alle Parameter festgelegt. Sollte die Layoutdatei nicht vorhanden sein oder sich eine der Ausgabedateien nicht anlegen lassen, bricht das Programm mit einer entsprechenden Fehlermeldung ab. Andernfalls beginnt das Programm mit dem Einlesen der Layoutdatei. Da die ESPRO-Layoutdateien keine Angaben ber die Anzahl der Zellen, Pads, Segmente und Kontaktlöcher enthalten, muß diese Datei in zwei Durchgängen eingelesen werden. Im ersten Durchgang wird nur die Anzahl aller Elemente ermittelt und entsprechend viel Arbeitsspeicher reserviert. Es erscheinen dann folgende Meldungen:

```
Analysieren der Layoutdatei...
  Pads: (Anzahl)
  Zellen: (Anzahl)
  Segmente: (Anzahl)
  Kontaktlöcher: (Anzahl)
  Netze: (Anzahl)
```

Sollte der Arbeitsspeicher nicht ausreichen, um die Layoutdatei vollständig einzulesen, wird hier mit einer entsprechenden Fehlermeldung abgebrochen. Andernfalls wird die Datei eingelesen, und die eingelesenen Daten werden sortiert.

```
Einlesen der Layoutdatei...
Sortieren der Segmente, Vias, Zellen und Pads ...
```

Wurde beim Sortieren erkannt, daß äquivalente Kontaktlöcher vorhanden sind, so werden diese jetzt gesucht und aus dem Speicher entfernt. Es erscheint dann die Meldung:

```
Suchen von äquivalenten Vias...
  (Anzahl) äquivalente Vias gefunden.
```

Nun erfolgt das Normalisieren der Segmente, und danach das Extrahieren der Netzlisten.

`Normalisieren der Segmente...`
  (Anzahl) `normalisierte Segmente erzeugt`

`Extrahieren der Netzlisten`

Sollten während des Extrahierens Kurzschlüsse an Pins oder Pads festgestellt worden sein, so erscheint die Meldung „*(Anzahl)* `Kurzschlüsse an Pins oder Pads gefunden`". Nach dem Extrahieren erfolgt die Suche von Fehlern im Layout. Dies wird durch entsprechende Meldungen angezeigt:

`Suchen von überflüssigen Kontaktlöchern...`

`Suchen von Kurzschlüssen an Kontaktlöchern...`

`Suchen von Segmentkurzschlüssen...`

`Suchen von offenen Segmentenden...`

`Design Rule Check...`

`Suchen von Kreisen...`

Werden hierbei Fehler festgestellt, so wird jeweils die Anzahl der Fehler auf dem Bildschirm ausgegeben. Zuletzt wird der Zeitbedarf des Programms angezeigt:

`Zeitbedarf:` t `Sekunden`

Sollte auch die Ausgabe des normalisierten Layouts gewünscht worden sein, erfolgt dies jetzt. Da dies nicht zum eigentlichen Algorithmus gehört, wird der Zeitbedarf für diesen Programmteil auch nicht mit ermittelt.

```
Umsortieren der normalisierten Segmente...
Schreiben der normalisierten Layoutdatei...
```

Damit ist das Programm beendet.

A.1.2 Fehlermeldungen und Warnungen

In diesem Abschnitt werden alle Fehlermeldungen und Warnungen erklärt, die in einer Fehlerdatei auftreten können. Jede Fehlermeldung und jede Warnung hat eine Nummer. Fehlermeldungen bezeichnen schwere Fehler (s. Kap. 3.1), Warnungen unechte Fehler (s. Kap. 3.2).

Fehler 1-8 und Warnungen 21-27

Die folgenden Fehlermeldungen mit den Nummern 1 bis 8 (4 ist nicht belegt) und Warnungen mit den Nummern 21 bis 27 werden in allen Warnstufen ausgegeben. Sie enthalten jeweils die Anzahl der Fehler.

`Fehler 1: Insgesamt` i `Segmente der Länge Null gefunden.`
In der Layoutdatei sind Segmente enthalten, deren Anfangs- und Endkoordinate gleich sind, die also die Länge Null haben. Dieser Fehler wird beim Normalisieren der Segmente festgestellt.

`Fehler 2: Insgesamt` i `Kurzschlüsse an Kontaktlöchern gefunden.`
Es gibt Kontaktlöcher, die Segmente miteinander verbinden, die zu verschiedenen Netzen gehören.

`Fehler 3: Insgesamt` i `Kurzschlüsse an Zellen oder Pads gefunden.`
Es gibt Pins oder Pads, die mit mehreren verschiedenen Netzen verbunden sind. Dies führt zu Kurzschlüssen zwischen den Netzen.

`Fehler 5: Insgesamt` i `Kurzschlüsse in Layer 1.`
Es gibt im ersten Layer Segmente verschiedener Netze, die durch Kreuzung oder Überlappung miteinander verbunden sind. Dies führt zu Kurzschlüssen zwischen den entsprechenden Netzen.

`Fehler 6: Insgesamt` i `Kurzschlüsse in Layer 2.`
Es gibt im zweiten Layer Segmente verschiedener Netze, die durch Kreuzung oder Überlappung miteinander verbunden sind. Dies führt zu Kurzschlüssen zwischen den entsprechenden Netzen.

`Fehler 7:` i `x-Segmente verlaufen durch Zellen.`
Es gibt x-Segmente, die durch Zellen belegte Bereiche schneiden. Dies ist in ESPRO verboten (routing over the cell).

`Fehler 8:` i `y-Segmente verlaufen durch Zellen.`
Es gibt y-Segmente, die durch Zellen belegte Bereiche schneiden. Dies ist in ESPRO verboten (routing over the cell).

`Warnung 21: Insgesamt` i `überflüssige Vias gefunden.`
Im Layout treten Kontaktlöcher auf, die mit nur einem oder mit keinem Segment verbunden und damit überflüssig sind.

`Warnung 22: Insgesamt` i `äquivalente Vias gefunden.`
Im Layout treten Kontaktlöcher mit denselben Koordinaten mehrfach auf.

`Warnung 23:` i `Segmente am Anfang offen.`
Es gibt Segmente, deren Anfangspunkt mit keinem anderen Segment, Kontaktloch, Pin oder Pad verbunden ist.

`Warnung 24:` i `Segmente am Ende offen.`
Es gibt Segmente, deren Endpunkt mit keinem anderen Segment, Kontaktloch, Pin oder Pad verbunden ist.

`Warnung 25:` i `Segmente am Anfang und Ende offen.`
Es gibt Segmente, bei denen sowohl deren Anfangs- als auch deren Endpunkt mit keinem anderen Segment, Kontaktloch, Pin oder Pad verbunden ist.

`Warnung 26: Insgesamt` i `überflüssige Teilnetze gefunden.`
Es treten Segmente auf, die zwar untereinander verbunden sind, bei denen jedoch kein Segment mit einem Pin oder Pad verbunden ist.

`Warnung 27: Insgesamt` i `Netze enthalten Kreise.`
Es treten Netze auf, die Kreise enthalten.

Fehler 101-107

Die nun folgenden Fehlermeldungen mit den Nummern 101 bis 107 werden nur in den Warnstufen 1 oder 2 ausgegeben bzw. lassen sich bei Warnstufe 3 in der Konfigurationsdatei individuell ein- bzw. ausschalten. Die Fehler, deren Anzahl in den Fehlermeldungen 1 bis 7 ausgegeben wird, werden durch diese Fehlermeldungen vollständig ausgegeben.

`Fehler 101: Segmente der Länge Null:`
Nach dieser Fehlermeldung werden alle Segmente ausgegeben, deren Anfangs- und Endkoordinaten gleich sind. Die Segmente werden – wie in den meisten anderen ESPRO-Dateien – in der Form `x=`x`:(`y_1`,`y_2`;L`l`,N`n`)` (y-Segment) bzw. `y=`y`:(`x_1`,`x_2`;L`l`,N`n`)` (x-Segment) ausgegeben. Dabei steht x, y, x_1, x_2, y_1 und y_2 für Koordinaten, l für den Layer (1 oder 2) und n für die Netznummer (Beispiel: `x=103:(10,22;L1,N35)`).

`Fehler 102: Koordinaten von Kurzschlüssen an Vias:`
Auf diese Fehlermeldung folgen alle Koordinaten von Kontaktlöchern, die Kurzschlüsse verursachen, zusammen mit den beteiligten Netzen. Die Ausgabe erfolgt dabei in der Form `(`x`,`y`;N`n`[`t_n`]+N`m`[`t_m`])`. Dabei stehen x und y für die Koordinaten des Kontaktloches, n und m für die Netze und t_n und t_m für deren Teilnetznummern.

`Fehler 103: Kurzschluß Netz` n `+ Netz` m `bei Z`z`(`p`).`
Der Pin p der Zelle z ist sowohl mit Netz n als auch mit Netz m verbunden. Dadurch sind diese beiden Netze kurzgeschlossen. Für jeden Fehler dieser Art wird eine solche Meldung ausgegeben.

`Fehler 104: Kurzschluß Netz` n `+ Netz` m `bei P`p`.`
Der Pad p ist sowohl mit Netz n als auch mit Netz m verbunden. Dadurch sind diese beiden Netze kurzgeschlossen. Die Meldung wird für jeden Fehler dieser Art ausgegeben.

`Fehler 105: Kurzschlüsse in Layer 1:`
Nach dieser Meldung werden alle Segmentkurzschlüsse ausgegeben, die im ersten Layer auftreten. Jeder Kurzschluß wird dabei in folgender Form ausgegeben:
`Netz` n`[`t_n`] + Netz` m`[`t_m`]; Segmente` *Segment1* `+` *Segment2*
Dabei sind n und m die Nummern der Netze, t_n und t_m die Teilnetznummern der beteiligten Segmente, und *Segment1* und *Segment2* die Segmente, die den Kurzschluß verursachen. Die Segmente werden in der gleichen Form ausgegeben wie bei Fehler 101. Beispiel:
`Netz 0[1] + 21[2]; Segmente y=108:(124,164;L1,N0[1]) + x=139:(100,153;L1,N21[2]).`

`Fehler 106: Kurzschlüsse in Layer 2:`
Nach dieser Meldung werden alle Segmentkurzschlüsse ausgegeben, die im ersten Layer auftreten. Die Ausgabe erfolgt dabei in der gleichen Form wie bei Fehler 105.

`Fehler 107: Segment` *Segment* `verläuft durch Zelle Z`z`.`
Das Segment *Segment* schneidet den durch die Zelle z belegten Bereich. Das Segment wird dabei in der gleichen Form ausgegeben wie bei Fehler 101. Sollte ein Segment mehrere Zellen schneiden, so wird diese Fehlermeldung nur für eine dieser Zellen ausgegeben.

Warnungen 201-207

Die Warnungen mit den Nummern 201 bis 207 werden nur in der Warnstufe 2 ausgegeben bzw. lassen sich bei Warnstufe 3 in der Konfigurationsdatei individuell ein- bzw. ausschalten.

`Warnung 201: Koordinaten überflüssiger Vias:`
Im Anschluß an diese Warnung werden die Koordinaten aller überflüssigen Kontaktlöcher ausgegeben. Die Ausgabe erfolgt in der Form `(`x`,`y`)`. Dabei stehen x und y für die x- bzw. y-Koordinate.

`Warnung 202: Koordinaten äquivalenter Vias:`
Im Anschluß an diese Warnung werden die Koordinaten aller äquivalenten Kontaktlöcher ausgegeben. Die Ausgabe erfolgt in der Form `(`x`,`y`)`. Dabei stehen x und y für die x- bzw. y-Koordinate.

`Warnung 203: Segment` *Segment* `offen am Anfang (überflüssig) / (zu lang).`
Für jedes Segment, dessen Anfangspunkt offen ist, wird diese Warnung ausgegeben. Das Segment wird dabei in der gleichen Form ausgegeben wie bei Fehler 101. Sollte das Segment zu lang sein, so wird `(zu lang)` ausgegeben. Ist das Segment jedoch überflüssig, wird `(überflüssig)` ausgegeben.

`Warnung 204: Segment` *Segment* `offen am Ende (überflüssig) / (zu lang).`
Für jedes Segment, dessen Endpunkt offen ist, wird diese Warnung ausgegeben. Alles andere entspricht der Warnung 203.

`Warnung 205: Segment` *Segment* `offen am Anfang und Ende (überflüssig) / (zu lang).`
Diese Warnung wird für jedes Segment erzeugt, bei dem sowohl der Anfangs- als auch der Endpunkt offen ist. Alles weitere entspricht der Warnung 203.

`Warnung 206: Überflüssige Teilnetze in Netz` n`:`
Diese Warnung wird für jedes Netz erzeugt, das überflüssige Teilnetze enthält. Es werden die Teilnetznummern aller überflüssigen Teilnetze des Netzes mit der Netznummer n ausgegeben.

`Warnung 207: Netz` n `enthält einen Kreis.`
Für jedes Netz, das mindestens einen Kreis enthält, wird diese Warnung erzeugt. Dabei ist n die Netznummer des betroffenen Netzes. Sollte ein Netz mehrere Kreise enthalten, wird jedoch nur eine Warnung erzeugt.

A.1.3 Die Konfigurationsdatei EXTRACT.CFG

Wurde die Warnstufe 3 gewählt, wird vom Programm die Konfigurationsdatei EXTRACT.CFG (unter RISC OS heißt diese CFG) eingelesen. Die Datei wird im aktuellen Arbeitsverzeichnis erwartet. Sollte diese nicht vorhanden oder fehlerhaft sein, wird mit einer entsprechenden Fehlermeldung abgebrochen.
Die Konfigurationsdatei ermöglicht es, jede der Fehlermeldungen 101-107 und Warnungen 201-207 individuell ein- oder auszuschalten. Die Konfigurationsdatei ist eine ASCII-Datei, in der jede Zeile folgendermaßen aufgebaut sein muß:
{*Schlüsselwort*}{`+`|`-`}
Jede Zeile enthält also ein Schlüsselwort, und unmittelbar hinter dem Schlüsselwort muß das Zeichen + oder - stehen. Falls eine Zeile mit einem Semikolon beginnt, wird diese überlesen (Kommentarzeile). Für jede der Fehlermeldungen und Warnungen gibt es ein Schlüsselwort. Diese Schlüsselwörter sind in der Tabelle A.1 zusammengefasst. Steht hinter einem Schlüsselwort das Zeichen +, so wird die entsprechende Fehlermeldung oder Warnung eingeschaltet. Steht hinter dem Schlüsselwort das Zeichen - oder ist das Schlüsselwort in der Konfigurationsdatei nicht vorhanden, ist die dazugehörige Meldung ausgeschaltet. Ein Beispiel für eine Konfigurationsdatei ist auf den beiliegenden Disketten enthalten.

A.1.4 Format von Extraktionsdateien

In der Extraktionsdatei steht das Ergebnis des Extraktionsvorgangs, also die etxrahierten Netzlisten. Extraktionsdateien sind im wesentlichen genau so aufgebaut wie die in [Kla94] beschriebenen Basisdateien. Sie enthalten

- die Anzahl der Pads,
- die Anzahl der Zellen und die Breite jeder einzelen Zelle,

Schlüsselwort	Fehler/Warnung
`NullSegment`	101
`Kurzschluss_Via`	102
`Kurzschluss_Pin`	103
`Kurzschluss_Pad`	104
`Kurzschluss_Segment_L1`	105
`Kurzschluss_Segment_L2`	106
`DesignRule`	107
`Via_Ueberfluessig`	201
`Via_Aequivalent`	202
`Segment_OffenAnfang`	203
`Segment_OffenEnde`	204
`Segment_OffenAnfangEnde`	205
`TeilnetzUeberfluessig`	206
`Kreis`	207

Tabelle A.1: Schlüsselwörter für die Konfigurationsdatei

- die Anzahl der Netze und
- die Netzliste jedes Netzes.

Gegenüber der Basisdatei enthält die Extraktionsdatei jedoch noch einige zusätzliche Informationen. Der Aufbau einer Extraktionsdatei sei deshalb hier kurz erläutert:

`>Extraktionsdaten_des_Schaltkreises:` *(Schaltkreisname)*

`>#Ports:` *(Anzahl der Pads)*
`>#Zellen:` *(Anzahl der Zellen)*
`>#Netze:` *(Anzahl der Netze)*

`>#Zellanschluesse:`
Für jede Zelle:
`>Zelle_Nr.` z`:` *(Anzahl)* `Anschluesse`

`>Netzlisten:`
Für jedes Netz:
`>Netz_Nr.` n`:` *(Anzahl)* `Nachbarn,` *(Anzahl)* `Teilnetze`
(*Netzliste*)

Die Netzliste enthält alle Nachbarn des Netzes in der Form `P`p`:`t für Pads (mit p für die Nummer des Pads und t für die Nummer des Teilnetzes, mit dem der Pad verbunden ist), oder `Z`z`(`p`):`t für Pins (mit z für die Nummer der Zelle, p für die Nummer des Pins der Zelle und t für die Nummer des Teilnetzes, mit dem der Pin verbunden ist).
Beispiel:

```
>Netz_Nr. 0: 8 Nachbarn, 1 Teilnetze
(P9:1, Z0(1):1, Z3(2):1,
Z4(1):1, Z6(2):1, Z8(1):1,
Z9(2):1, Z9(3):1)
```

A.1.5 Hinweise zum Sourcecode

Der Sourcecode des Programms EXTRACT ist auf der beiliegenden Diskette im Verzeichnis EXTRACT enthalten. Das Programm wurde znächst auf einem Acorn A5000 mit Acorn C/C++

Version 5.06 entwickelt und debugged. Danach wurde das Programm unter SunOS mit dem GNU-C-Compiler und unter MS-DOS ebenfalls mit dem GNU-C-Compiler übersetzt. Dabei wurde unter MS-DOS die EMX-Portierung des GCC verwendet. Diese Version erzeugt 32-Bit Protected Mode-Anwendungen, so daß – mit Ausnahme des Konvertierens vom Latin-1- in den IBM-Zeichensatz – keine Anpassungen des Sourcecodes an MS-DOS nötig sind. Beim Übersetzen unter MS-DOS muß das Makro MS_DOS definiert sein. Der Sourcecode ist in den Dateien EXTRACT.C, EXT_TOOL.C, EXT_MEM.C, EXT_CHEC.C und EXT_IO.C sowie EXTRACT.H, EXT_TOOL.H, EXT_MEM.H, EXT_CHEC.H und EXT_IO.H enthalten. Der Inhalt dieser Dateien ist in der Tabelle A.2 zusammengefasst.

Name	Inhalt
EXTRACT.C	Hauptprogramm; Extrahieren der Netzlisten; Suchen von kurzschlieeenden und überflüssigen Kontaktlöchern
EXT_TOOL.C	Hilfsfunktionen; Normalisieren der Segmente
EXT_CHEC.C	Fehlersuche: Segmentkurzschlüsse, offene Segmente, Design Rule Check, Kreise
EXT_IO.C	Dateiroutinen
EXT_MEM.C	Speicherverwaltung
EXTRACT.H	Datenstrukturen und Konstanten

Tabelle A.2: Sourcecode-Dateien des Programms EXTRACT

A.2 COMPARE

Der Algorithmus zum Netzlistenvergleich ist in dem Programm COMPARE implementiert, welches in diesem Abschnitt beschrieben wird. Auch dieses Programm steht – wie das Programm EXTRACT – zur Zeit für die Betriebssysteme RISC OS, UNIX (SunOS) und MS-DOS zur Verfügung und liegt der Arbeit auf Diskette bei.

A.2.1 Bedienung des Programms

Nach dem Start des Programms durch Eingabe von `Compare` erscheinen auf dem Bildschirm zunächst folgende Meldungen:

```
COMPARE - Vergleicher für
ESPRO-Basis- und Extraktionsdateien
Version vom: (Datum)

Erste Eingabedatei (Basis- oder Extraktionsdatei):
```

Soll eine Basisdatei mit einer Extraktionsdatei (also die Originalnetzlisten mit den extrahierten Netzlisten) verglichen werden, ist der Name einer Basisdatei einzugeben. Sollen hingegen zwei Extraktionsdateien verglichen werden (z. B. vor und nach einer Nachoptimierung), so muß der Name der ersten Extraktionsdatei eingegeben werden. Die Namenserweiterungen (hier also .LAY oder .EXT) sind stets mit anzugegeben. Danach fragt das Programm nach der zweiten Datei:

```
Zweite Eingabedatei (Extraktionsdatei):
```

Die zweite Eingabedatei muß immer eine Extraktionsdatei sein. Zuletzt fragt das Programm noch nach der Ergebnisdatei:

`Ergebnisdatei (Ausgabedatei):`

Dies ist eine Textdatei, in der alle vom Programm festgestellten Unterschiede protokolliert werden (ähnlich wie die Fehlerdatei bei EXTRACT). Damit sind alle Parameter festgelegt. Sollte sich eine der Dateien nicht öffnen lassen, wird mit einer Fehlermeldung abgebrochen. Andernfalls stellt das Programm zunächst fest, ob es sich bei den Eingabedateien um Basis- oder Extraktionsdateien handelt. Sollte die erste Datei keine Basis- oder Extraktionsdatei oder die zweite Datei keine Extraktionsdatei sein, wird mit einer Fehlermeldung abgebrochen. Andernfalls erscheint entweder:

`Vergleiche Basisdatei mit Extraktionsdatei`
oder
`Vergleiche zwei Extraktionsdateien`

Falls die Schaltkreisnamen in den beiden Dateien unterschiedlich sein sollten, erscheint die Meldung:

```
Die Schaltkreisnamen in den beiden Dateien
sind nicht gleich. Trotzdem fortfahren (J/N)?
```

Wird hier `J` eingegeben, wird das Programm fortgesetzt, andernfalls abgebrochen.
Danach werden die Dateien eingelesen:

`Einlesen der Dateien...`

Sollte dabei festgestellt werden, daß die Anzahl der Pads, Zellen oder Netze in den beiden Dateien unterschiedlich ist, so wird mit einer Fehlermeldung abgebrochen, da es sich in diesem Fall wahrscheinlich um unterschiedliche Schaltungen handelt. Es ist in diesem Fall nicht sinnvoll, den Vergleich fortzusetzen.
Nun werden zunächst die Breiten der Zellen (also die Anzahl der Anschlüsse jeder Zelle) verglichen. Werden Unterschiede festgestellt, so werden diese in die Ergebnisdatei geschrieben und die Anzahl der Unterschiede auf dem Bildschirm ausgegeben. Danach werden die Netzlisten verglichen. Auch hier werden Unterschiede in die Ergebnisdatei geschrieben. Zum Schluß werden noch einige statistische Daten ausgegeben:

`Vergleich der Zellbreiten...`

`Vergleich der Netzlisten...`

`Übereinstimmungen:` *(Anzahl)*
`Fehlende Nachbarn:` *(Anzahl)*
`Überzählige Nachbarn:` *(Anzahl)*

Sollten in der zweiten Datei Netze aufgetreten sein, die mehr als ein Teilnetz enthalten, wird deren Anzahl ausgegeben. Außerdem werden die Nummern dieser Netze in die Ergebnisdatei geschrieben.

`Netze mit mehr als einem Teilnetz:` *(Anzahl)*

Zum Schluß wird noch der Zeitbedarf ausgegeben:

`Zeitbedarf:` t `Sekunden`

Damit ist das Programm beendet.

A.2.2 Fehlermeldungen

In diesem Abschnitt werden alle Fehlermeldungen erklärt, die in der Ergebnisdatei auftreten können.

`Zellen mit unterschiedlicher Breite:`
Nach dieser Fehlermeldung werden die Nummen aller Zellen ausgegeben, deren Breite in der ersten Datei anders ist als in der zweiten Datei.

`Netz Nr.` n`: Fehlende Nachbarn:`
Im Anschluß an diese Fehlermeldung werden alle fehlenden Nachbarn des Netzes mit der Nummer n ausgegeben. Fehlende Nachbarn sind Nachbarn, die nur in der ersten Datei, jedoch nicht in der zweiten Datei vorhanden sind. Die Ausgabe erfolgt in der bekannten Form Pp bzw. Z$z(p)$.

`Netz Nr.` n`: Überzählige Nachbarn:`
Im Anschluß an diese Fehlermeldung werden alle überzähligen Nachbarn des Netzes mit der Nummer n ausgegeben. Überzählige Nachbarn sind Nachbarn, die nur in der zweiten Datei, jedoch nicht in der ersten Datei vorhanden sind. Die Ausgabe erfolgt in der gleichen Form wie bei fehlenden Nachbarn.

`Netze mit mehr als einem Teilnetz:`
Nach dieser Fehlermeldung werden die Nummern aller Netze ausgegeben, die mehr als ein Teilnetz haben. Bei diesen Netzen kann es sich um unterbrochene Netze handeln, oder es gibt überflüssige Teilnetze.

A.2.3 Hinweise zum Sourcecode

Der Sourcecode des Programms COMPARE ist auf der beiliegenden Diskette im Verzeichnis COMPARE enthalten. Es gelten hier sinngemäß die gleichen Hinweise wie zum Sourcecode des Programms EXTRACT. Das Programm COMPARE besteht aus den Dateien COMPARE.C, COMPARE.H, COMP_IO.C, COMP_IO.H, COMP_MEM.C und COMP_MEM.H. Der Inhalt dieser Dateien ist in der Tabelle A.3 zusammengefaßt.

Name	Inhalt
COMPARE.C	Hauptprogramm; Vergleich der Zellbreiten und der Netzlisten; Suchen von Netzen mit mehr als einem Teilnetz
COMP_IO.C	Dateiroutinen
COMP_MEM.C	Speicherverwaltung
COMPARE.H	Datenstrukturen und Konstanten

Tabelle A.3: Sourcecode-Dateien des Programms COMPARE

A.3 Beispielsitzung

In diesem Abschnitt wird die Anwendung der Software an einer Beispielsitzung erläutert. Es sei dabei angenommen, daß das Layout des auf der Diskette enthaltenen Beispielchips T200 bearbeitet

werden soll. Eingaben des Benutzers sind dabei in *kursiver* Schrift gehalten.
Zunächst wird das Programm EXTRACT gestartet. Als Layoutdatei wird T200.LAY, als Extraktionsdatei T200.EXT und als Fehlerdatei T200.ERR angegeben. Die normalisierten Segmente sollen nicht ausgegeben werden, und es wird die Warnstufe 2 verwendet:

```
EXTRACT - Extraktor für ESPRO-Layoutdateien
Version vom: 26. Mai 1997

Name der Layoutdatei: T200.LAY
Name der Extraktionsdatei (1. Ausgabedatei): T200.EXT
Name der Fehlerdatei (2. Ausgabedatei): T200.ERR


Warnstufe:
  0 - Fehler: nur Anzahl; Warnungen: nur Anzahl
  1 - Fehler: vollständig; Warnungen: nur Anzahl
  2 - Fehler: vollständig; Warnungen: vollständig
  3 - Konfigurationsdatei verwenden
? 2
Soll das normalisierte Layout ausgegeben werden (J/N)? N
Analysieren der Layoutdatei...
  Pads: 14
  Zellen: 200
  Segmente: 4155
  Kontaktlöcher: 3676
  Netze: 103
Einlesen der Layoutdatei...
Sortieren der Segmente, Vias, Zellen und Pads...
Normalisieren der Segmente...
  5028 normalisierte Segmente erzeugt
Extrahieren der Netzlisten
Suchen von überflüssigen Kontaktlöchern...
Suchen von Kurzschlüssen an Kontaktlöchern...
  1 Kurzschlüsse an Kontaktlöchern gefunden
Suchen von Segmentkurzschlüssen...
  -Layer 1...
  21 Segmentkurzschlüsse gefunden
  -Layer 2...
Suchen von offenen Segmentenden...
  1 Segment am Anfang und Ende offen
  1 Segment am Ende offen
Design Rule Check...
  1 y-Segmente verlaufen durch Zellen
Suchen von Kreisen...
Zeitbedarf: 37.25 Sekunden
Fertig!
```

Das Ergebnis des Programms EXTRACT findet sich nun in den Dateien T200.EXT und T200.ERR. Die von EXTRACT gemeldeten Fehler können nachvollzogen werden, indem man sich die entsprechenden Stellen im Layout z. B. mit dem Programm DISPLAY aus [Kla94] ansieht.

Nun erfolgt noch ein Netzlistenvergleich. Dazu wird das Programm COMPARE gestartet. Als Eingabedateien werden T200.CHP und T200.EXT angegeben, und als Ausgabedatei wird T200.LST angegeben:

```
COMPARE - Vergleicher für
ESPRO-Basis- und Extraktionsdateien
Version vom: 26. Mai 1997


Erste Eingabedatei (Basis- oder Extraktionsdatei): T200.CHP
Zweite Eingabedatei (Extraktionsdatei): T200.EXT
Ergebnisdatei (Ausgabedatei): T200.LST

Vergleiche Basisdatei mit Extraktionsdatei


Einlesen der Dateien...

Vergleich der Zellbreiten...

Vergleich der Netzlisten...

Übereinstimmungen: 1358 (99.9%)
Fehlende Nachbarn: 1 (0.1%)
Überzählige Nachbarn: 0 (0.0% bezogen auf die zweite Datei)
Netze mit mehr als einem Teilnetz: 1 (1.0%)
Zeitbedarf: 0.08 Sekunden
Fertig!
```

Das Ergebnis des Vergleichs steht nun in der Datei T200.LST. Die dort gemeldeten Fehler können überprüft werden, indem die betroffenen Netze mit dem Programm DISPLAY aus [Kla94] betrachtet werden.

Anhang B

Danksagung

Ich möchte mich bedanken bei

- Dr. R. Rauscher und Prof. Dr. K. Kaiser für die Betreuung dieser Arbeit
- Dipl.-Math. Dieter Klawan, der in seiner Diplomarbeit [Kla94] das ESPRO-System entwikkelt hat, wodurch die vorliegende Arbeit erst ermöglicht wurde

Anhang C

Inhalt der Disketten

C.1 Diskette 1: Source-Diskette

Diese Diskette enthält den Sourcecode der beiden Programme EXTRACT und COMPARE, und zwar sowohl in der Version für MS-DOS als auch für RISC OS und UNIX. Die Versionen für RISC OS und UNIX sind dabei als Archive auf der Diskette enthalten, um die bei diesen Betriebssystemen üblichen längeren Dateinamen zu erhalten. Der einzige Unterschied zwischen den Versionen für MS-DOS, RISC OS und UNIX ist der Zeichensatz: IBM-Zeichensatz in der MS-DOS-Version, Latin-1-Zeichensatz in den beiden anderen Versionen. Es gibt sonst keine weiteren Unterschiede im Sourcecode. Die MS-DOS-Version kann nur mit einem C-Compiler übersetzt werden, der 32-bit-Code (Protected Mode) erzeugt (z. B. GNU EMX oder Watcom C).

- Verzeichnis **MSDOS**:
  - Unterverzeichnis **EXTRACT**: Sourcecode des Programms EXTRACT:
EXTRACT.C .... Hauptprogramm
EXTRACT.H .... Header-Datei mit Datenstrukturdefinitionen
EXT_TOOL.C .... Hilfsfunktionen
EXT_TOOL.H .... Header-Datei zu EXT_TOOL.C
EXT_CHEC.C .... Fehlersuchfunktionen
EXT_CHEC.H .... Header-Datei zu EXT_CHEC.H
EXT_IO.C .... Dateifunktionen
EXT_IO.H .... Header-Datei zu EXT_IO.C
EXT_MEM.C .... Speicherverwaltung
EXT_MEM.H .... Header-Datei zu EXT_MEM.C
MK_EXT.BAT .... Batch-Datei zum Übersetzen des Programms
  - Unterverzeichnis **COMPARE**: Sourcecode des Programms COMPARE:
COMPARE.C .... Hauptprogramm
COMPARE.H .... Header-Datei mit Datenstrukturdefinitionen
COMP_IO.C .... Dateifunktionen
COMP_IO.H .... Header-Datei zu COMP_IO.C
COMP_MEM.C .... Speicherverwaltung
COMP_MEM.H .... Header-Datei zu COMP_MEM.C
MK_COMP.BAT .... Batch-Datei zum Übersetzen des Programms
- Verzeichnis **RISCOS**: Version für Acorn-Rechner ("Ur-Version"):
EXTRACT.ARC .... Archiv mit Sourcecode des Programms EXTRACT
COMPARE.ARC .... Archiv mit Sourcecode des Programms COMPARE

  Beide Archive wurden mit Hilfe des Programms !Spark erstellt und enthalten außerdem die ausführbaren Programme sowie Makefiles.

- Verzeichnis **UNIX**:
  EXTRACT.TAR .... Archiv mit Sourcecode des Programms EXTRACT
  COMPARE.TAR .... Archiv mit Sourcecode des Programms COMPARE

  Beide Archive wurden mit Hilfe des Programms TAR erstellt. Das Auspacken erfolgt ebenfalls mit diesem Programm.

C.2 Diskette 2: Programm-Diskette

Diese Diskette enthält die ausführbaren Programme für MS-DOS. Es handelt sich dabei um 32-Bit-Protected-Mode-Anwendungen. Damit diese Programme unter MS-DOS ausgeführt werden können, wird ein DPMI-Server benötigt. Auf der Diskette sind dazu die Programme RSX.EXE und EMX.EXE enthalten. Bevor die Programme EXTRACT.EXE und COMPARE.EXE verwendet werden können, müssen RSX.EXE und EMX.EXE installiert werden. Dazu werden diese einfach in das DOS-Systemverzeichnis kopiert, so daß diese von MS-DOS gefunden werden können. Weitere Installationsschritte sind nicht erforderlich. Auf dieser Diskette sind außerdem einige Beispieldateien enthalten.

- Verzeichnis **DPMI**:
  RSX.EXE .... DPMI-Server. Diese beiden Programme
  EMX.EXE .... am besten ins DOS-Verzeichnis kopieren.

- Verzeichnis **PROGRAMM**:
  EXTRACT.EXE .... Programm EXTRACT (Extraktion und Fehlersuche)
  COMPARE.EXE .... Programm COMPARE (Netzlistenvergleich)

- Verzeichnis **BEISPIEL**:

  Dieses Verzeichnis enthält insbesondere die im Kapitel 6 verwendeten Beispiele, in folgenden Unterverzeichnissen:
  BASIS .... Basisdateien (*.CHP)
  DKROUTER .... Mit DKROUTER erzeugte Layoutdateien (*.LAY)
  NACHOPT .... Mit NACHOPT nachoptimierte Layoutdateien (*.OPT)
  NETZMIN .... Mit NETZMIN nachoptimierte Layoutdateien (*.MIN)

Die beiden Disketten befinden sich auf der vorletzten Umschlagseite.

Anhang D

Erklärung

Ich versichere hiermit, daß ich die vorliegende Arbeit selbständig verfaßt und mich dabei keiner anderen als der angegebenen Quellen und Hilfsmittel bedient habe.

Gr. Grönau, den 26. Mai 1997

Olav Kahlbaum

Literaturverzeichnis

[Kla94] D. Klawan: ESPRO – Ein Programmpaket für Placement & Routing beim Standardzellentwurf, Diplomarbeit, Universität Hamburg, FB Mathematik, 1994

[Kas95] O. Kastek: Nachoptimierung von Standardzellentwürfen, Diplomarbeit, Universität Hamburg, FB Mathematik, 1995

[Rau94] R. Rauscher: Skript zur Vorlesung "CAD-Algorithmen für den VLSI-Entwurf", 1. Kapitel (Placement & Routing), Universität Hamburg, FB Informatik, 1994

[Kah95] O. Kahlbaum: Implementierung und Evaluierung von Standard-Heuristiken zum Placement und Routing, Studienarbeit, Universität Hamburg, FB Informatik, 1995

[RKB96] R. Rauscher, D. Klawan, H.-J. Bandelt: Results given by a New Evaluation System for Placement and Routing Heuristics, in: M.E. Kavanaugh (ed.), Proceedings of EUROMICRO '96, Los Alamitos: IEEE Computer Society Press, 1996, S. 259-266

[Zob94] Routing, Placement and Partitioning, G. W. Zobrist (ed.), Ablex Publishing Corp., Norwood, New Jersey, 1994

[Wol94] W. Wolf: Modern VLSI Design: A Systems Approach, Prentice Hall, New Jersey, 1994, Kap. 9.5

[Yos86] K.Yoshida: Layout Verification, in: Layout Design and Verification, T. Ohtsuki (ed.), Elsevier Science Publishers, North Holland, Amsterdam, 1986, S. 237ff.

[Sher95] N. Shervani: Algorithms for VLSI Physical Design Automation, Kluwer, Boston, 1995

[She89] A. Sherman: VLSI Placement and Routing: The PI Project, Springer Verlag, Berlin, 1989

[Hol86] T. Holfert: HEKI – Ein hierarchischer Schaltkreisextraktor für Gate-Arrays, Studienarbeit, Universität Hamburg, FB Informatik, 1986

[Hol88] T. Holfert: Verfahren zum Vergleich und zur Extraktion von Strukturbeschreibungen integrierter Schaltungen, Diplomarbeit, Universität Hamburg, FB Informatik, 1988

[Blö88] J. Blödel, R. W. Hartenstein, W. Nebel, M. Ryba: Automatische Extraktion von Registertransfer-Beschreibungen aus dem Layout integrierter Schaltungen, Zwischenbericht, Universität Kaiserslautern, FB Informatik, 1988

[Neb86] W. Nebel: REX – Automatic Extraction of RT-Level Descriptions from Integrated Circuit Layout Data, Dissertation, Universität Kaiserslautern, FB Informatik, 1986

[Bee95] F.Beeftink, A. J. van Genderen, N. P. van de Meijs: Accurate and Efficient Layout-to-Circuit Extraction for High-Speed MOS and Bipolar/BiCMOS Integrated Circuits, Proceedings ICCD: VLSI in Computers & Processors, October 1995, S. 360-365

[Rob95] A. B. Khang, G.Robins: On Optimal Interconnections for VLSI, Kluwer, Boston, 1995

[Sec85] C. Sechen, A. Sangiovanni-Vincentelli: The TimberWolf Placement and Routing Package, IEEE Journal of Solid State Ciruits 20, 2 (April 1985), S. 510ff.

[Sec86] C. Sechen, A. Sangiovanni-Vincentelli: TimberWolf 3.2: A New Standard Cell Placement and Global Routing Package, Proceedings of the 23rd ACM/IEEE Design Automation Conference, S. 432-439, 1986

[Dör88] W. Dörfler, W. Peschek: Einführung in die Mathematik für Informatiker, Hanser, München, 1988, Kap. 8

[Lee61] C. Lee: An Algorithm for Path Connection and its Applications, IRE Transactions Electronic Computers, EC-10 (1961), S. 346-365

[Ayr83] R. F. Ayres: VLSI/Silicon Compilation and the Art of Automatic Microchip Design, Prentice Hall, New Jersey, 1983

[Lag87] K. Lagemann: Rechnerstrukturen, Springer-Verlag, Berlin, 1987

[MeCo80] C. Mead, L. Conway: Introduction to VLSI Systems, Addison-Wesley, 1980

[Pos89] H.-U. Post: Entwurf und Technologie hochintegrierter Schaltungen, Teubner, Stuttgart, 1989