Friedrich-Alexander-Universität Erlangen-Nürnberg
Lehrstuhl für Rechnergestützten Schaltungsentwurf
Prof. Dr.-Ing. Wolfram H. Glauert

Studienarbeit

Entwicklung von Design-Validierungsregeln f¨ur einen Halbleiterprozess

Bearbeiter: Gunter K¨onigsmann
Betreuer : Prof. Dr.-Ing. Wolfram H. Glauert
Dipl.-Ing. J¨urgen Krumm

Beginn : 12.11.2001
Abgabe : 12.04.2002

Ich versichere, daß ich die Arbeit ohne fremde Hilfe und ohne Benutzung anderer als der angegebenen Quellen angefertigt habe und daß die Arbeit in gleicher oder ¨ahnlicher Form noch keiner anderen Pr¨ufungsbeh¨orde vorgelegen hat und von dieser als Teil einer Pr¨ufungsleistung angenommen wurde. Alle Ausf¨uhrungen, die w¨ortlich oder sinngem¨aß ¨ubernommen wurden, sind als solche gekennzeichnet.

(Gunter K¨onigsmann)

Zusammenfassung

Ziel dieser Arbeit war es, das Layoutentwicklungssystem Cadence auf eine neue organische Technologie anzupassen, so dass ein Design Rule Check und ein Layout Versus Schematic-Test an importierten Layouts vorgenommen werden kann. Neben der eigentlichen Arbeit und der Dokumentation derselben wurde versucht, zu beschreiben, wie Cadence intern funktioniert, und die Teile, die hier verwendet wurden, an beliebige neue Technologien anzupassen sind. Hauptproblem war, ein ¨ubliches Programm an nicht standardgem¨aße Transistoren anzupassen.

Abstract

The intention of this paper was to adapt the integrated layout development system Cadence to a new organic process technology to implement a Design Rule Check and a Layout versus Schematic Test.

Save to document this the second aim was showing how to adjust Cadence to any new technology as far as the scope of this work reaches. The main difficulty was to adjust a standard-program to non- standard-Transistors

Inhaltsverzeichnis

Kapitel 1

Einleitung

Die etablierten Silizium-Halbleitertechnologien erlauben die Produktion großer St¨uckzahlen hochqualitativer und schneller Bauteile. Die Produktionskosten sind dabei niedrig genug, dass auch ausgesprochene Lifestyleprodukte (Tamagochi, Gameboy-Kamera etc.) in diesen Technologien hergestellt werden.

Der Produktion von Silizium-Halbleitertechnologien findet unter extremen Reinraumbedingungen statt. Auch ben¨otigt die Herstellung selbst einfacher Chips oft weit ¨uber tausend Produktionsschritte. Die ben¨otigten Maschinen und R¨aumlichkeiten k¨onnen nur ¨uber große St¨uckzahlen finanziert werden.

Auch die Herstellung von Kleinserien ist kostenintensiv, da die zahlreichen Lithographieschritte Masken mit extrem niedrigen Strukturgr¨oßen erfordern. Die hohe Zahl an Produktionsschritten erh¨oht die Zeitdauer von Produktionsbeginn bis zum ersten fertigen Chip auf einige Wochen bis Monate [32]. Gegenstand dieser Arbeit ist eine Technologie, die Transistoren aus Polymerwerkstoffen einsetzt. Diese lassen sich theoretisch mittels eines umgebauten Tintenstrahldruckers fertigen. Die hohen Schaltgeschwindigkeiten und niedrigen Leckstr¨ome der CMOS- Technologie lassen sich hier nicht erreichen. Investitionskosten und Vorlaufzeiten beschr¨anken sich daf¨ur auf ein Minimum.

Wer schon einmal versucht hat, von Hand integrierte Schaltungen selbst geringer Komplexit¨at zu entwerfen oder Fehler darin zu suchen, weiß, wie viel Arbeit ein professionelles Schaltungsentwurfssystem dem Entwickler abnimmt.

Schon einfache Schaltungen enthalten in der Regel weit ¨uber tausend Transistoren. Meist beschr¨anken sich große Teile eines Schaltungsentwurfs auf wenige (z.B. 8) Transistortypen. Die eindeutige Zuweisung von Transistoren in Schaltbild und Schaltung f¨ur den Vergleich derselben ist auch durch die Gr¨oße und weitgehende Ähnlichkeit der Baugruppen zueinander stark erschwert.

Auch formale Entwurfsregeln, deren Nichteinhalten die Zuverl¨assigkeit von Schaltungen stark reduzieren kann, sind in einer Schaltung von z.B. 5mm Durchmesser und 0,35μm Detailgr¨oße schwer zu verifizieren. Selbst wenn man den Schaltungsentwurf (im Folgendem kurz: Layout) auf mehrere Meter Durchmesser vergr¨ößert, ist das Finden eines einzelnen Fehlers allein aufgrund des Umfangs dieser Aufgabe nahezu unm¨oglich.

F¨ur beide Aufgaben m¨ussen alle im Schaltungsentwurf enthaltenen Bauteile detektiert und vermessen werden. Eng verwandt damit ist dadurch die Aufgabe, die Einfl¨usse der Verdrahtung auf die elektrischen Parameter der Schaltung als parasit¨are (ungewollte) Bauteile aufzufassen, um diese f¨ur eine genaue Simulation der konkreten Realisation eines Schaltkreises verwenden zu k¨onnen.

Ziel dieser Arbeit ist, Cadence, eine der popul¨are Entwicklungsumgebung f¨ur integrierte Schaltungen an eine

Polymertechnologie anzupassen, die in vielen Hinsichten un¨ublich (und somit f¨ur das Programm schwer zu handhabend) ist. Da der Mangel an passender Literatur jede neue Aufgabe entschieden verlangsamt, wurde außerdem versucht, den allgemeinen Weg daf¨ur zu dokumentieren, soweit dies im Bereich der konkret vorliegenden Technologie m¨oglich war.

Die Implementierung wurde flexibel gehalten, um m¨oglichst viele im vorliegenden Fertigungsprozess verwandten Technologien abdecken zu k¨onnen.

Kapitel 2

Entwurf einer integrierten Schaltung

Der Entwurf einer integrierten Schaltung erfolgt in mehreren Schritten. Die konkrete Realisation der einzelnen Schritte variiert von Hersteller zu Hersteller, allen gemein ist jedoch folgende Vorgehensweise [27]:

- Textuelle Beschreibung der Aufgabe, die von der Schaltung zu erf¨ullen ist.
- Aufteilung der Schaltung in einzelne Baugruppen.
- Bei digitalen Baugruppen: Algorithmische Beschreibung und Minimierung der Schaltfunktion.
- Entwurf des technologieunabh¨angigen Prinzipschaltplans (Bild 2.1 a).
- Entwurf des technologieabh¨angigen Schaltplans (Bild 2.1 b). In diesem Schaltplan ist jedes in der fertigen Schaltung enthaltene Bauteil mit allen f¨ur dessen Dimensionierung n¨otigen Parametern eingezeichnet. Der Schaltplan enth¨alt Informationen, welcher Anschluss der Bauteile mit welchem verbunden ist. Informationen ¨uber die konkrete Platzierung der Bauteile und den Verlauf der Verdrahtung enth¨alt er nicht.
- Zeichnen des Layouts (Bild 2.1 c). Das Layout ist eine Zeichnung des fertigen Chips. Es enth¨alt alle Informationen ¨uber die konkrete Realisation der fertigen Schaltung. Die f¨ur die Produktion des Chip n¨otigen Masken werden maschinell aus dem Layout ermittelt.
- Simulation der fertigen Schaltung einschliesslich der Verdrahtung. Bei der Verdrahtung der einzelnen Bauteile untereinander entstehen zwangsl¨aufig ungewollte (parastit¨are) Bauteile.

Layout und Schaltplan werden mit getrennten Programmen entworfen. Die einzelnen Programme k¨onnen sich aber in der Regel untereinander verst¨andigen, um Fehler beim Zeichnen zu erkennen.

Beide in der vorliegenden Arbeit bearbeiteten Aufgaben stehen im Zusammenhang mit dem letzten Schritt des Schaltungsentwurfs:

1. Automatisches Auffinden nahezu aller denkbaren Fehler, die beim Schritt vom Schaltplan zum Layout entstehen k¨onnen.
2. Eintragen der durch die Verdrahtung entstandenen parasit¨aren Bauelemente in den Schaltplan f¨ur eine nachfolgende Schaltungssimulation.

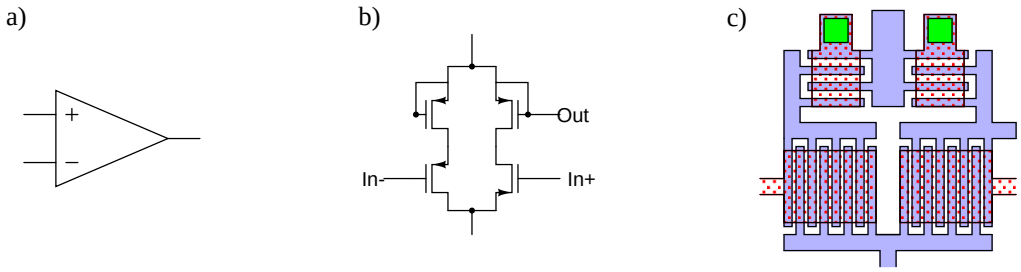


Abbildung 2.1: Schritte des Schaltungsentwurfs: a) Prinzipschaltplan, b) Technologieabh¨angiger Schaltplan, c) Layout

Kapitel 3

Das Arbeitsmaterial

Dieses Kapitel beschreibt die wichtigsten Daten der behandelten Technologie, sowie die verwendeten Programme und Dateien.

3.1 Technologie

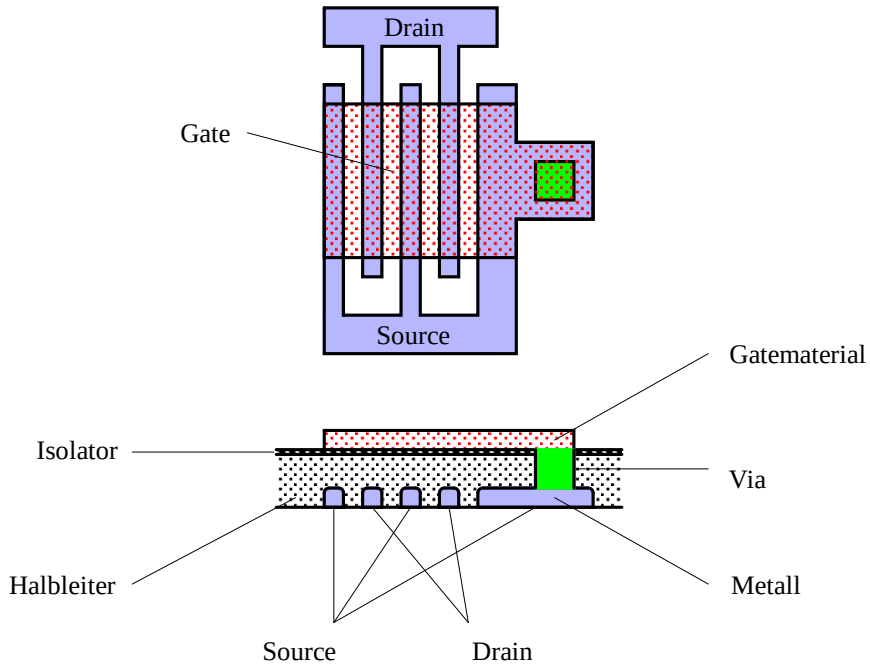


Abbildung 3.1: Transistor mit Durchkontaktierung zwischen Source und Gate in Aufsicht und Schnitt

Die Hauptherausforderung bei der Anpassung des Systems stellte die verwendete Technologie, die sich in einigen Punkten stark von der ¨ublichen Siliziumtechnologie unterscheidet [26, 28, 29, 30]:

Silizium-MOS-Transistoren sind sehr einfach durch UND-ODER-NICHT- Verkn¨upfungen der einzelnen Materialschichten eindeutig zu erkennen, und die ¨ublichen Programme stellen praktische, genau auf diese Tran-

sistoren und ihre Parameter zugeschnittene Befehle zur Verf¨ugung. Diese Befehle sind sehr m¨achtig, m¨ussen aber in dieser Arbeit mit viel Aufwand umgangen werden, da sie nicht die n¨otige Flexibilit¨at aufweisen.

In der vorliegenden Technologie entsteht ein Transistor (Bilder 3.2 und 3.1) auf andere Weise. Ein in der vorliegenden Polymertechnologie gefertigter Chip besteht aus f¨unf Materialschichten (englisch Layer):

- Die unterste Materialschicht, auf die die Schaltung aufgetragen wird, ist nichtleitend und wird als „Substrat“ bezeichnet.
- Auf dieses Substrat werden Leiterbahnen aus einem elektrisch leitf¨ahigen Material aufgetragen. Diese Materialschicht wird in der vorliegenden Arbeit als METAL-Layer bezeichnet.
- Der Chip wird nun ganzfl¨achig mit einer Polymerschicht mit nichtlinearen elektrischen Eigenschaften ¨uberzogen. Die METAL- Schicht ist mit dieser elektrisch verbunden.
- Auf die Oberfl¨ache dieser Halbleiterschicht wird ganzfl¨achig ein Isolator aufgetragen.
- Eine zweite Ebene von Leitungen bildet die oberste Schicht des Schaltkreises. Sie wird in Anlehnung an die CMOS-Technologie im Folgenden als GATE-Layer bezeichnet. Über Durchkontaktierungen kann sie mit der METAL-Schicht verbunden werden. Wenn an Objekte in dieser Leitungsebene eine negative Spannung angelegt wird, beginnt die Polymerschicht darunter elektrisch zu leiten.

Wenn nun zwei parallele METAL-Leitungen von einer GATE-Leitung ¨uberkreuzt werden, fließt zwischen den METAL-Leitungen ein Strom, der stark von der Spannung an der GATE-Leitung abh¨angt. Dies ist das Verhalten eines Feldeffekttransistors (Bild 3.2).

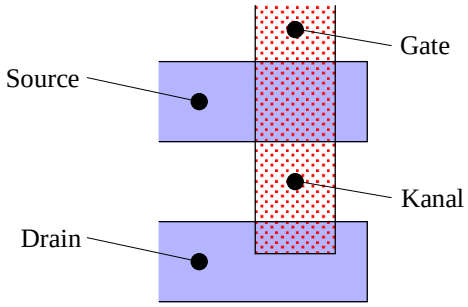


Abbildung 3.2: Einfacher Transistor mit einem Kanal

Die Detektion und das Vermessen von Bauteilen dieser Art aus einem fertig gezeichnetem Schaltungsentwurf (englisch Layout) heraus ist aus folgenden Gr¨unden schwer zu implementieren:

- Transistoren sind nicht durch eine eindeutige Kombination von Materialschichten definiert, die außerhalb von Transistoren nicht existieren kann.
- Polymertransistoren werden durch Strukturen gebildet, die in keiner der etablierten Technologien Bauteile formen. Funktionen, die f¨ur deren Detektion und Vermessung geeignet sind, werden von Cadence nur teilweise unterst¨utzt.
- Es ist ein der Regel nicht vermeidbar, dass die Struktur, die einen Transistor ausmacht, durch die Verdrahtung der Schaltung entsteht. Diese parasit¨aren Transistoren sind funktionsf¨ahige Bauteile, k¨onnen aber durch entsprechende Entwurfsregeln so klein gehalten werden, dass sie keine nennenswerten Str¨ome ziehen. Beim Vergleich von Layout und Schaltplan d¨urfen nicht als Bauteile verglichen werden.

- Um gr¨oßere Str¨ome erreichen zu k¨onnen, werden in der vorliegenden Technologie oft einige identische Transistoren parallel geschaltet. Um die Anzahl der in einer Schaltung enthaltenen Bauteile zu reduzieren, ist es n¨otig, diese zu einem Transistor zusammenzufassen.

Anschlusspads stellen die Verbindung der Schaltung zur Außenwelt her. Sie werden als großfl¨achige Durchkontaktierungen ausgef¨uhrt.

Neben den mit der Aufgabenstellung gegebenen Bauteilen wurden experimentell Kondensatoren eingef¨uhrt: METAL-Objekte haben eine gewisse Kapazit¨at zu dar¨uberliegenden GATE-Objekten. Daten ¨uber Wert und Reproduzierbarkeit dieser Kapazit¨at konnten im Ramen dieser Arbeit nicht ermittelt werden.

3.2 Programme

Cadence ist ein Entwicklungssystem f¨ur integrierte Halbleiterbausteine, das aus einer großen Anzahl von Programmen besteht. Nur ein Bruchteil davon wurde im Zuge dieser Arbeit verwendet.

Allen Programmen gemein ist, dass sie ¨uber die Programmiersprache SKILL (Anhang A, Seite 64) konfiguriert werden. Zu beachten ist, dass viele Programme eigene Dialekte dieser Programmiersprache verwenden, die nur bestimmte Konstrukte und Schreibweisen unterst¨utzen. Empfohlen ist nur die im Anhang A beschriebenen Schreibweise der Befehle, die alle Programme verstehen. Eventuelle Einschr¨ankungen werden im Implementierungsteil der Arbeit, wo n¨otig, erl¨autert.

Um dem Leser einen Einblick in Cadence zu erm¨oglichen, werden die verwendeten Programme im Folgenden kurz dargestellt.

Alle f¨ur diese Arbeit verwendeten Programme besitzen die Versionsnummer 4.4.3.

3.2.1 CIW

Das CIW (Command Interaction Window, Bild 3.3) ist das Herzst¨uck von Cadence, von dem aus die einzelnen Anwendungen aufgerufen werden.

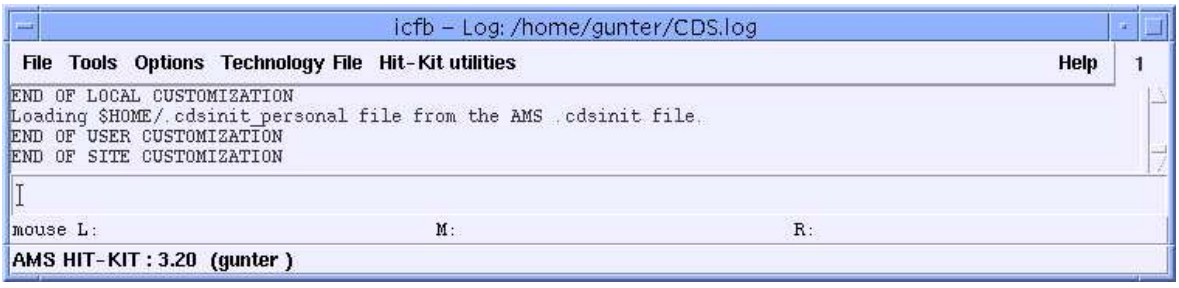


Abbildung 3.3: CIW

In der Titelleiste des Programmfensters stehen der Name des Programms, sowie die Datei, in die alle Statusmeldungen geschrieben werden. Die zweite Zeile enth¨alt ein Men¨u, ¨uber das es m¨oglich ist, weitere Programme zu starten. Auf diese folgt der Bereich, in den alle Programme, die zu Cadence geh¨oren ihre Status- und Fehlermeldungen ausgeben. In die letzte Zeile dieses Bereichs kann der Benutzer Befehle in der im gesamten

Programmsystem eingesetzten Programmiersprache SKILL (S. 64, Anhang A) schreiben, die Cadence sofort ausf¨uhrt. Die Ausgaben der von Hand eingegebenen Befehle werden im CIW mitprotokolliert.

Die Fußzeile des CIW zeigt an, welche Funktionen an welche Maustaste gebunden sind, falls von der aktiven Anwendung definiert.

3.2.2 Library Manager

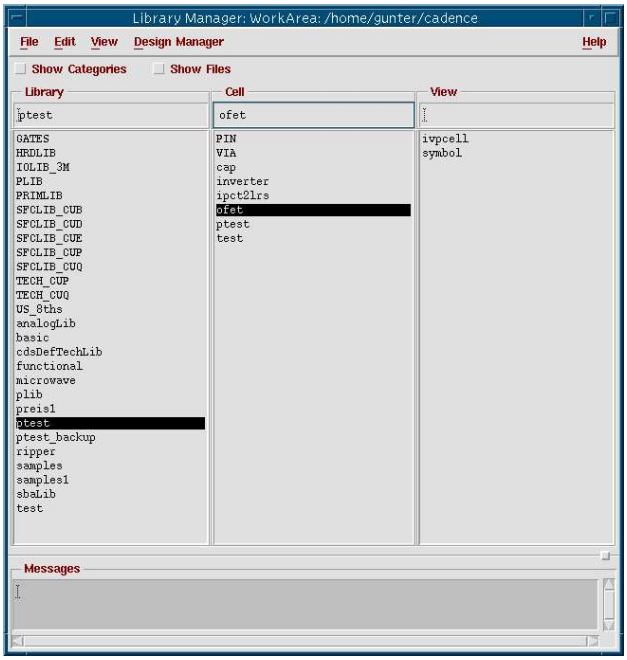


Abbildung 3.4: Library Manager

Die verwendeten Technologien und Bauteile werden in Bibliotheken zusammengefasst, die der Library Manager (Bibliotheksverwalter, Bild 3.4) verwaltet. In der linken Spalte kann die Bibliothek ausgew¨ahlt werden, an der man arbeiten will. Die mittlere Spalte zeigt das zu bearbeitende Bauteil/Layout (Bild eines Bauelements) an. Oft existieren Bauteile in verschiedenen Ansichten (z.B. Schaltplan `(schematic)`, Symbol `(symbol)`, Layout `(layout)`, oder Layout, aus dem die einzelnen Bauteile extrahiert wurden `(extracted)`). Diese werden in der rechten Spalte angew¨ahlt.

Dr¨ucken der rechten Maustaste ¨offnet ein Fenster, ¨uber das die Elemente der jeweiligen Spalte verwaltet werden k¨onnen.

3.2.3 CDF

Fast jedes Programm ben¨otigt eigene Bauteilmodelle, die auf die jeweilige Applikation zugeschnitten sind. Damit sich die einzelnen Programme untereinander verst¨andigen k¨onnen, werden die Parameter (Kanall¨ange, Kanalweite, Kapazit¨at etc.) der Bauteile zentral im‚Component Description Format [18]‘ abgespeichert. Der CDF- Editor (Bild 3.5), mit dem die einzelnen Parameter, die ein Bauteil besitzen soll, festgelegt werden, ruft der Men¨upunkt *tools/cdf* im CIW (Command Interpreter Window, Bild 3.3) auf.

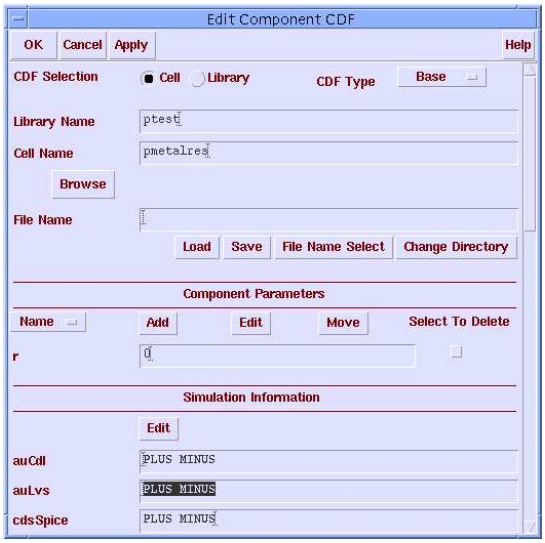


Abbildung 3.5: CDF-Editor

Neben der globalen CDF-Beschreibung (CDF Type=*Base*) lassen sich zwei weitere Parameters¨atze anw¨ahlen [17]:

- *User*: Änderungen an diesem Parametersatz wirken sich nur auf den Benutzer aus, der diese vorgenommen hat. Eine *User*- Beschreibung ersetzt die kompletten globalen Einstellungen.
- *Effective*: Dieser Parametersatz ist benutzerspezifisch, ersetzt aber, anders als der *User*-Satz, nur die Parameter, die in dieser CDF- Beschreibung ver¨andert wurden.

3.2.4 Diva

Die im Zuge dieser Arbeit implementierten Aufgaben (DRC, LVS und Layoutextraktion, siehe jeweils Unten) werden von einem Programmpaket erledigt, dessen Name Diva ist. Die verwendeten Programmteile werden in diesem Kapitel im Einzelnem angesprochen.

3.2.5 Oberfläche des Composer

Der Composer (Verfasser, Bild 3.6) erm¨oglicht das Zeichnen von Schaltungssymbolen. Zwei Varianten dieses Programmes k¨onnen aufgerufen werden: Eine f¨ur symbolische Schaltpl¨ane (Composer–Schematic) und eine f¨ur Bauteilsymbole (Composer–Symbol).

3.2.6 Oberfläche des Virtuoso

Virtuoso [2] ist ein umfangreiches Zeichenprogramm, mit dem Layouts bearbeitet werden. Alle Funktionen sind ¨uber Men¨us erreichbar. Der‚*tools*‘-Men¨upunkt bindet weitere Programme und deren Befehle in die Men¨uzeile ein.

Die im Zuge dieser Arbeit implementierten Aufgaben k¨onnen alle vom Men¨upunkt *verify* aus gestartet werden.

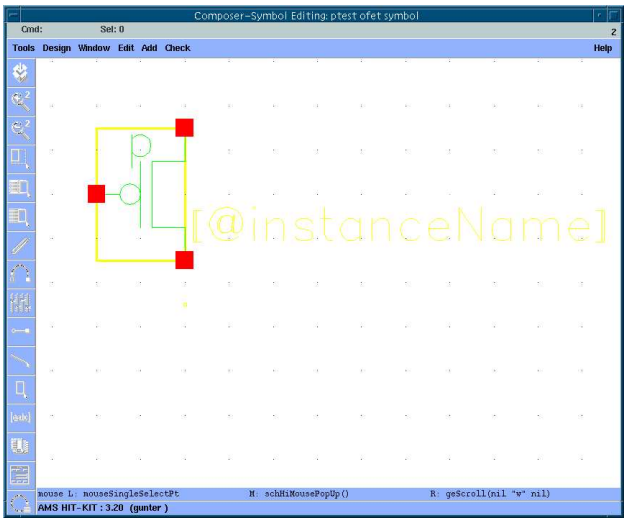


Abbildung 3.6: Composer

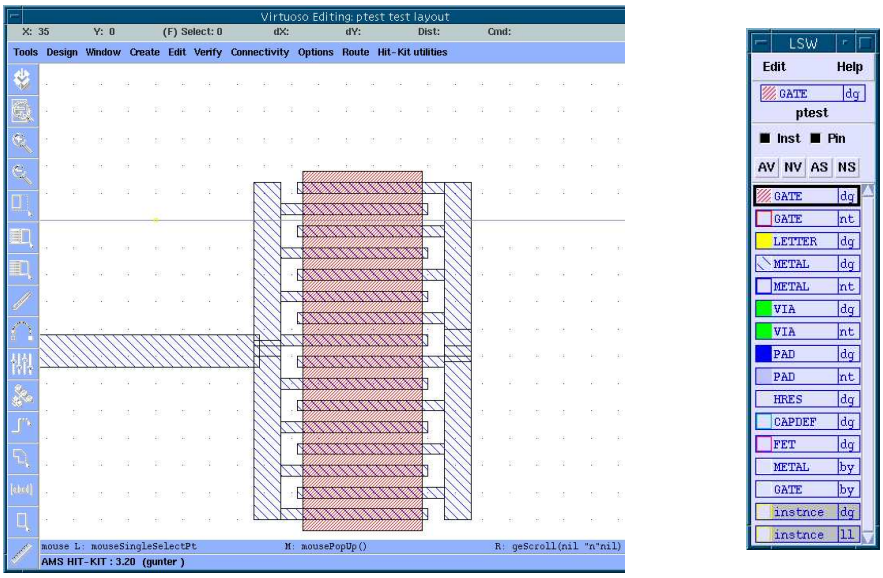


Abbildung 3.7: Virtuoso

3.2.7 LSW

Das Layer Selection Window (LSW, Bild 3.7 rechts) wird von Virtuoso gestartet, wenn es noch nicht aktiv ist. In ihm wird die aktive Zeichenebene (Layer) ausgew¨ahlt, sowie die Sichtbarkeit der einzelnen Layer definiert.

3.2.8 Layoutextraktion

Die Layoutextraktion (Bild 3.8) erm¨oglicht es, alle Bauteile, die in einem Layout enthalten sind, zu extrahieren, um die Schaltung simulieren und mit dem Schaltplan vergleichen zu k¨onnen.

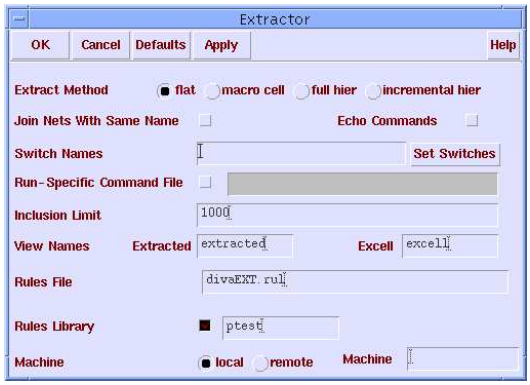


Abbildung 3.8: Layoutextraktor

3.2.9 Design Rule Check

Der Design Rule Check (Entwurfsregeltest, Bild 3.9) überprüft, ob ein Layout den formalen Entwurfsregeln entspricht, die für eine bestimmte Technologie definiert sind. Entwurfsregeln sind zum Beispiel die Mindestabstände zwischen Leitungen oder die Mindestlinienbreiten.

Abbildung 3.9: Design Rule Check-Programm

3.2.10 LVS

Das LVS (Layout Versus Schematic, Layout gegen Schaltplan)-Programm (Bild 3.10) vergleicht Schaltpläne mit fertig gezeichneten Layouts.

Große Teile dieses Programms sind streng von Cadence getrennt, und können auch per Kommandozeile gestartet werden. Die Statusmeldungen dieser Programme werden nicht in das CIW ausgegeben. Sie lassen sich über den ‚*Monitor*‘ - Knopf des LVS-Fensters erreichen.

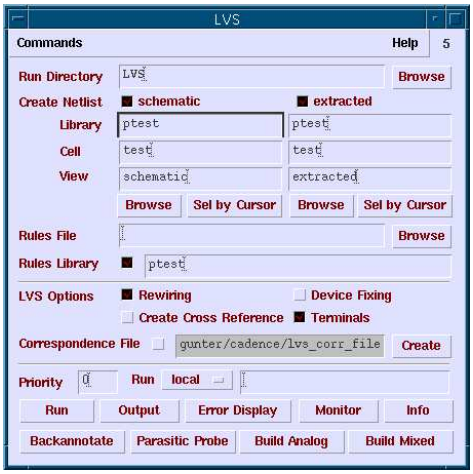


Abbildung 3.10: LVS

3.3 Verwendete Dateien

Die verwendeten Programme werden von einer Vielzahl von Dateien aus konfiguriert. Die meisten von diesen bestehen aus ASCII-Texten, die mit einem beliebigem Texteditor (z.B. EMACS) editiert werden k¨onnen.

3.3.1 techfile.txt

Die Datei techfile.txt enth¨alt die globale Technologiebeschreibung. Neben der Definition der Layer (Materialschichten und Bauteilmasken, S. 3.4) geh¨oren dazu von diversen Programmen verwendete Bauteilmodelle und Parameter, auf die von allen Konfigurationsdateien aus zugegriffen werden kann.

Nahezu alle Definitionen in dieser Datei k¨onnen vom Anwender komfortabel ¨uber Men¨us im CIW (S. 7) editiert werden. Informationen, die der Anwender ver¨andern k¨onnen soll, werden daher sinnvollerweise hier abgelegt.

3.3.2 display.drf

Um die gezeichneten Layouts darstellen zu k¨onnen, muss jedem Layer ein Farbpaket zugewiesen werden. Farbpakete bestehen jeweils aus Farbe und F¨ullmuster von Objekten, sowie Linienfarbe und -Muster f¨ur deren Umrandung. Sie werden in der Datei display.drf definiert.

Die Dateiendung .drf steht f¨ur‚Display Resource File‘ (Anzeigeressourcendatei)

3.3.3 divaEXT.rul, divaDRC.rul

Die Dateien divaEXT.rul und divaDRC.rul unterscheiden sich in ihrem Aufbau wenig voneinander. Sie konfigurieren den Design Rule Check (DRC) und die Bauteilextraktion (EXT) des Programmpakets Diva. Die Endung .rul steht f¨ur Rules, zu deutsch Regeln.

3.3.4 `divaLVS.rul`

Die „Layout Versus Schematic Test“-Konfigurationsdatei von Diva besteht gr¨oßtenteils aus kurzen Routinen, die Bauteile entscheiden, ob zwei Bauteile als gleich anzusehen sind. Weitere Routinen entscheiden, ob zwei Bauteile, die parallel oder in Reihe geschaltet sind, zusammengefasst werden k¨onnen.

3.3.5 `streamtab`

Die Datei `streamtab` legt fest, welcher Datenstrom beim Import von GDS II-Dateien [7] welcher Materialschicht in Cadence entspricht.

Die meisten Cadence-Installationen lesen die ben¨otigten Werte aus der Datei `techfile.txt` aus, und ben¨otigen diese Datei daher nicht.

3.3.6 `pcells.txt`

Die Datei `pcells.txt` wird gebraucht, um frei verformbare Bauteile f¨ur die Bauteilextraktion generieren zu k¨onnen. Sie wird nur zum Erzeugen der Bauteilebibliothek verwendet, und kann einen beliebigen Namen besitzen.

3.3.7 `ptest`

Keine eigendliche Datei ist die Bibliothek `ptest`, (Polymertest) in der der Programmcode dieser Arbeit zusammengefasst wurde. Um den Namen der Bibliothek zu ¨andern, ist in den Dateien `divaDRC.rul, divaEXT.rul` und `pcells.txt` das Wort ‚`ptest`‘ gegen den gew¨unschten Bibliotheksnamen auszuwechseln. Die Bibliothek wird erzeugt, indem im LSW (Bild 3.7) der Men¨upunkt *file/new Library* angeklickt wird. Im darauf erscheinenden Dialogfenster ist *Compile new Techfile* anzugeben, des Weiteren der Name der Bibliothek und der des Techfiles `techfile.txt`.

3.4 Layer

Bei den meisten Layoutprogrammen wird das Layout in so genannte Layer (Bild 3.11, Tabelle 3.1) zusammengefasst. Jeder Layer entspricht im Idealfall entweder einer physikalischen Schicht eines Bauteils (ein Layer pro Metallebene, Polysilizumlayer, n- und n^+-Layer, ...) oder wird f¨ur technische Gr¨unde verwendet (z.B. Layer, der die Umrisse der beabsichtigten bipolaren Transistoren markiert, um diese von parasit¨aren Transistoren unterscheiden zu k¨onnen.)

Alle in dieser Arbeit verwendeten Beispiele enthalten mit einer Ausnahme (VIA) ausschließlich die auch in schwarz-weiß unterscheidbaren Layer METAL und GATE.

H¨aufig bei der Schaltungsentwicklung verwendet werden Autorouter. Dies sind Programme, die auf einem Chip platzierten Elemente automatisch verdrahten. Mit der Einf¨uhrung dieser Programmklasse wurde es n¨otig, Leitungen, die vom Benutzer fest eingezeichnet sind, von solchen, die vom Autorouter verlegt werden d¨urfen, zu unterscheiden. Dies geschieht, indem den Objekten in den einzelnen Layern Verwendungszwecke (Purposes, Tabelle 3.2) zugewiesen werden.

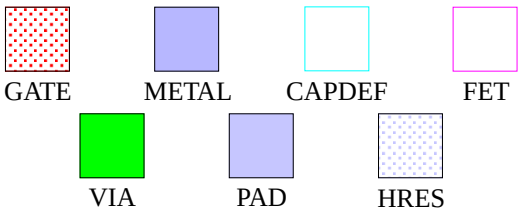


Abbildung 3.11: Die im folgenden verwendeten Layer mit Farben

Tabelle 3.1: Die f¨ur den Anwender sichtbaren Layer

GATE	Gate-Material; da leitend auch als Leitung verwendbar.
METAL	Metallleitungen; bilden unter Anderem auch die Source- und Drain-Regionen der Transistoren.
CAPDEF	Mit CAPDEF markierte Stellen, an denen sich METAL und GATE ¨uberlappen, werden vom Bauteilextraktor als Kondensatoren erkannt und extrahiert. Diese Funktion ist n¨otig, um gewollte von parasit¨aren Kondensatoren unterscheiden zu k¨onnen.
FET	Dieser Layer wurde eingef¨uhrt, um FETs manuell markieren und die Autodetektion ¨uberbr¨ucken zu k¨onnen. Er umfasst das kleinste Rechteck, das alle aktiven Bereiche eines Transistors enth¨alt. Konsequente Verwendung dieses Layers Spart Zeit bei Extraktion und Design Rule Checks.
LETTER	Es ist bei Testschaltungen ¨ublich, die Namen der Bauteile, Anschl¨usse etc. direkt auf die Schaltung zu drucken. Dies erm¨oglicht das Auffinden von Anschlusspads unter dem Mikroskop. Buchstaben m¨ussen weder genau reproduzierbar noch mit definierten elektrischen Eigenschaften ausgestattet sein. Sogar ihre Abst¨ande zu Leitungen k¨onnen kleiner sein, als die zwischen zwei Leitungen voneinander, da sie weder elektrisch verbunden sind, noch große Fl¨achen haben. Um Buchstaben mit eigenen Design Rules versehen zu k¨onnen, und sie graphisch besser sichtbar zu machen, wurde f¨ur sie ein eigener Layer eingef¨uhrt.
VIA	Elektrische Verbindungen zwischen METAL und GATE.
PAD	Anschlusspads. Pads in der vorliegenden Technologie haben die Besonderheit, dass sie neben ihrer Hauptaufgabe auch als Vias fungieren.
HRES	Bei Technologien, bei denen das Halbleitermaterial normalerweise die gesamte Chipfl¨ache bedeckt, wird dieses selbst nicht als eigener Layer definiert. Um Aussparungen im Halbleitermaterial realisieren zu k¨onnen, ist es zweckm¨aßiger, diesen Layer einzuf¨uhren, der eventuelle L¨ocher markiert.

Tabelle 3.2: F¨ur den Anwender sichtbare Purposes

net	Extrahierte elektrische Verbindungen (Leitungen, Pads und Vias).
drawing	Alle ¨ubrigen Objekte, sowie das gesamte Layout.
warning	Warnungen, die DRC, LVS Backannotate und Extraktion generieren.
error	von Cadence generierte Fehlermarkierungen

3.4.1 Cadence- interne Darstellung von Layern

Layer werden innerhalb von Cadence in zwei getrennten Strukturen gespeichert: Einer Liste, die das Aussehen des Layers enth¨alt, und einer zweiten, die besagt, welcher Teil welches Layers mit welchem elektrisch verbunden ist [6, `geomStamp`].

Inhalt von Layern

Das einfachste Verfahren, Layer im Speicher eines Rechners zu modellieren, ist das Modellieren derselben als Bitmap. (Bild 3.12, links.)

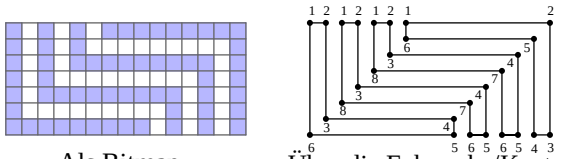


Abbildung 3.12: Ablegung von Layern

Der Layer wird hier als Gitter aus quadratischen Bl¨ocken behandelt, von denen jeder einzelne gef¨ullt, oder leer sein kann. Leider steigt der Speicherverbrauch mit dem Quadrat der Aufl¨osung, und diese ist nach dem kleinsten im Layout vorkommendem Detail zu bemessen. Die strenge Ausrichtung nach einem Gitter schr¨ankt außerdem alle Maße auf genaue Vielfache der Gitterkonstante ein.

Flexibler ist es, alle Objekte ¨uber ihren Typ und ihre Koordinaten zu beschreiben (Bild 3.12, rechts).

Cadence beschr¨ankt sich der Einfachheit halber auf Polygone. Jedes Objekt wird programmintern ¨uber eine Liste seiner Eckpunkte repr¨asentiert. Der Speicheraufwand ist hier nur von der Anzahl der Details abh¨angig. Der Verarbeitungsaufwand pro Detail ist jedoch betr¨achtlich.

Rechtecke sind keine eigenen Objekte, sondern rechteckige Polygone mit vier Kanten. Linien sind nicht geschlossene Polygone. Punkte sind Listen mit nur einem Eintrag f¨ur die Koordinate [6].

Realisierung von gebogenen Strukturen

Kreise und Kurven im Layout werden ¨uber Polygone approximiert (Bild 3.13). Bei der Verwendung von gebogenen Strukturen wie Kreisen ist zu beachten, dass dies nur innerhalb von gewissen Grenzen genau sein kann,

und manchmal zu unvorhergesehenen Resultaten f¨uhrt. So k¨onnen die Abst¨ande von Kreisen zu Polygonen je nach Approximationsgenauigkeit schwanken, und dadurch Entwurfsregeln verletzen.

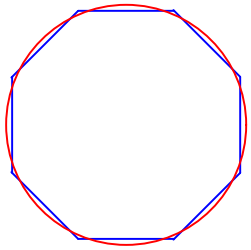


Abbildung 3.13: Approximierung eines Kreises durch ein Achteck

Hohle Objekte

Hohle Objekte (Doughnutstrukturen) werden ¨uber ‚C‘-f¨ormige Objekte realisiert, deren Öffnung die Weite 0 hat (Bild 3.14). Die Stelle, an der sich die beiden Kanten ber¨uhren, wird von L¨angenmessfunktionen [6] vernachl¨assigt.

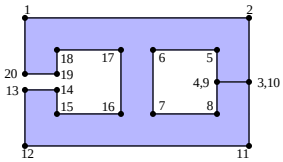


Abbildung 3.14: Realisierung von Doughnutstrukturen

Da die Öffnung der Weite 0 an eine beliebige Stelle des Polygons gelegt werden kann, w¨are das Ergebnis der Messfunktion ansonsten undefiniert. Die Hohlr¨aume selber werden mitgemessen. Die Messung der L¨ange eines Linienzuges liefert also nicht nur den Umfang eines Objektes zur¨uck, sondern den Umfang des Objektes plus den der Hohlr¨aume [6].

Information ¨uber elektrische Verbindungen

Der Inhalt von Layern wird in einer Liste von Objekten gespeichert, die wiederum aus Listen ihrer Eckpunkte bestehen. F¨ur jeden Layer im extrahiertem Layout existiert eine zweite Liste, die f¨ur jedes Objekt einen Eintrag enth¨alt, die Netzknotennummer. Zwei elektrisch miteinander verbundene Objekte erkennt Cadence daran, dass diese dieselbe Netzknotennummer besitzen [6, `geomStamp`]. Funktionen, die auf den Inhalt von Layern zugreifen, liefern einen neuen Layer mit dem neuen Inhalt zur¨uck. Funktionen, die die Netzknotennummern ver¨andern, definieren unter Umst¨anden auch die Knotennummern der Layer, die an die Funktion als Argument ¨ubergeben wurden, neu [6, 5].

Kapitel 4

Prinzip

4.1 Design Rule Checks

Viele h¨aufig entstehende Fehler k¨onnen durch relativ einfaches Vermessen der Geometrie von Objekten detektiert werden. Zu d¨unne Leitungen haben einen zu hohen Widerstand, zu kleine Abst¨ande bedeuten oft h¨ohere Leckstr¨ome. Auch die Zuverl¨assigkeit eines Bauteils und die Ausbeute k¨onnen durch das Verletzen von sicheren Mindestabst¨anden beeintr¨achtigt werden.

Der Test auf Entwurfsregeln (Design Rules) garantiert weder das Funktionieren der Schaltung, noch bedeuten Verletzungen dieser Sicherheitsregeln immer das Gegenteil. Bei der Fehlervermeidung leisten Design Rule Checks jedoch gute Dienste.

4.1.1 Prinzip

Die Idee, auf der Design Rule Checks in den meisten Programmen basieren, ist, einen Layer pro detektierbaren Fehlertyp zu definieren, in den die fehlerhaften Stellen eingezeichnet werden [4, 6, 23].

Cadence stellt hierzu mehrere Klassen von Werkzeugen zur Verf¨ugung. Diese liefern Layer zur¨uck, die alle Objekte oder Regionen von den Eingangslayern enthalten, die bestimmte Bedingungen erf¨ullen [6]:

- Logische Operationen liefern Layer, in denen alle Stellen markiert sind, an denen sich die Eingangslayer ¨uberlappen, nicht ¨uberlappen, oder mindestens ein Layer besetzt ist.

- Überlappungsfunktionen suchen ganze Objekte in einem Layer, die Objekte im zweiten Layer auf eine bestimmte Weise ber¨uhren. M¨oglichkeiten f¨ur Ber¨uhrungsmodi sind Ber¨uhrungen von innen und außen, sowie ganze, teilweise und beidseitige Überlappungen.

- L¨angenmessungsfunktionen suchen Kanten, die sich in einem bestimmtem L¨angenbereich bewegen.

- Cadenceenth¨alt zwei ziemlich m¨achtige Funktionen, die alle Kanten aller Objekte eines Layers um den selben Betrag nach außen (Expansion) oder nach innen (Kontraktion) bewegen. Die Hauptverwendung

dieser Funktionen im Zuge dieser Arbeit ist die L¨uckendetektion: Das Expandieren f¨ullt alle L¨ucken und Hohlr¨aume bis zum Doppeltem Expansionsbetrag. Doppelt, da die W¨ande zu beiden Seiten der L¨ucke verschoben werden. Ein nachfolgendes Kontraktieren f¨uhrt alle Objekte auf die Normalumrisse zur¨uck.

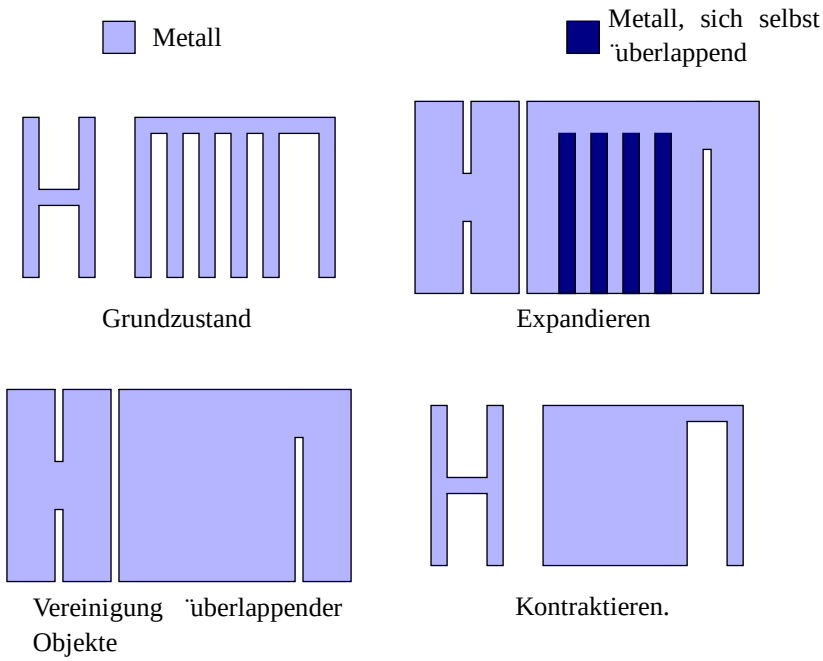


Abbildung 4.1: L¨uckendetektion durch Expandieren

- Die Objektsuche nach formalen Parametern sucht Rechtecke, nicht achsenparallele Objekte, Polygone,...
- Kantensuche (Selten verwendet) sucht parallele, antiparallele, etc. Kanten.
- Der `drc`-Befehl entspricht am ehesten den ¨ublichen formalen Design Rules: Dieser Befehl sucht Objekte, die zu kleine L¨ucken enthalten, zu nahe beieinander sind, sich nicht weit genug ¨uberlappen, usw.

  Zu Besonderheiten dieser Funktion sei auf [4] verwiesen.

4.1.2 Implementierte Design Rule Checks

Die Fehlermeldungen, die Verst¨oße gegen Entwurfsregeln generieren, wurden mit einem aus zwei Zeichen bestehenden Fehlercode versehen (Vgl. Anhang B) F¨ur jeden Fehlertyp wurde ein anderes erstes Zeichen des Fehlercodes vergeben.

HRES

Der HRES-Layer markiert L¨ocher im Halbleiter. In dieser Arbeit wird davon ausgegangen, dass Metallobjekte (Leitungen, FETs und Metallbuchstaben) und Objekte im GATE-Layer einen Mindestabstand zu L¨ochern halten

m¨ussen.

Vias und Pads sowie FETs sind nur innerhalb von GATE- Gebieten zugelassen und brauchen daher keine gesonderten Regeln in Bezug auf den HRES- Layer.

LETTER

Es ist ¨ublich, Erl¨auterungen direkt auf die Schaltung zu Drucken. Beispiele sind Pad-Namen f¨ur das Kontaktieren von Hand und die Namen einzelner Baugruppen.

Buchstaben, die direkt auf die Schaltung gedruckt werden, bestehen —je nach Technologie —entweder aus GATE-oder METAL-Material, und m¨ussen zu beiden, da leitf¨ahig, einen Mindestabstand halten, der vom DRC-Programm ¨uberpr¨uft wird.

Ein Mindestabstand zu HRES-Gebieten wurde ebenso implementiert. Buchstaben sind nicht elektrisch verbunden, beeinflussen also relativ nahegelegene Bauteile nur unwesentlich. Um eine eigene Entwurfsregel daf¨ur schreiben zu k¨onnen, und um Buchstaben optisch hervorheben zu k¨onnen, wurde ein eigener Layer f¨ur Buchstaben definiert.

CAPDEF

Dieser Layer markiert, wo Bereiche, in denen METAL und GATE sich ¨uberlappen, als Kondensatoren detektiert werden sollen. Nicht erlaubt ist ein Kondensator, der die Grenze eines CAPDEF-Bereiches kreuzt (Fehlermeldung C2), sowie Kondensatoren, die FET-Regionen ber¨uhren oder ¨uberlappen.

CAPDEF-Regionen ganz ohne Kondensatoren verletzen eine weitere Design Rule (C1).

VIA/PAD

Vias und Pads haben bis auf die Zahlenwerte die gleichen Eigenschaften: Beide m¨ussen um einen gewissen Betrag von METAL und GATE ¨uberlappt werden und haben einen Minimaldurchmesser, um fehlerfreie Kontaktierung (Pads) und fertigungstoleranzinsensitive Widerst¨ande (Vias) einzuhalten.

Des Weiteren haben sie einen Mindestabstand zu den aktiven Gebieten eines Transistors. Kondensatoren machen nur Sinn, wenn GATE- und METAL-Platte nicht elektrisch leitend miteinander verbunden sind. Durch Pads oder Vias kurzgeschlossene Kondensatoren werden daher auch als Fehler erkannt.

FET

Der Layer FET markiert das kleinste Rechteck, das alle aktiven Regionen eines Transistors enth¨alt. Es darf weder L¨ucken im Gatematerial, noch elektrisch unverbundene Metallst¨ucke umfassen. Alle Transistoren m¨ussen rechteckig sein. Da diese Art von Transistoren einfach und automatisiert zu zeichnen ist, und bei Platzmangel eher die Platzierung als die Form der Bauteile variiert wird, ist diese Einschr¨ankung ohne praktischen Belang. Sie ist n¨otig, um ein automatisches Ausmessen des Transistors beim LVS realisieren zu k¨onnen. Fehler, die bei der Transistorautodetektion gemacht werden, resultieren erfahrungsgem¨aß fast immer in nichtrechteckigen Transistoren im extrahierten Layout.

F¨ur gefaltete FETs (Bild 5.4 , S. 48) mit nicht schlangenf¨ormigen Kan¨alen ist eine minimale METAL/FET-Überlappung definiert, mit der man angeben kann, um welchen Betrag ein Source/Drain-Finger mindestens aus dem FET ragen muss, falls er dies tut. F¨ur schlangenf¨ormige Kan¨ale ist die kleinste FET/METALÜberlappung ¨uber die minimale Kanall¨ange implizit gegeben. Ein Test auf gleichm¨aßige Kanall¨ange innerhalb eines Transistors wird von Diva nicht angeboten.

GATE

Leitungen aus Gate-Material und Gatefl¨achen haben eine Mindestbreite und einen Minimalabstand zu weiteren GATE-Leitungen.

Eine minimale Einkerbungsweite (Minimalabstand zwischen zwei Teilen desselben GATE-Objektes) wurde ebenfalls definiert.

Transistorgates sind in der vorliegenden Technologie immer breiter, als die d¨unnste zugelassene GATE-Leitung. Eine aufw¨andige Fallunterscheidung zwischen Gates und GATE-Leitungen entf¨allt daher.

Transistoren und 2-Ebenen-Verdrahtung verlangen Kreuzungen zwischen METAL und GATE. Ein Minimalabstand von Objekten in den beiden Layern l¨asst sich trotzdem sinnvoll definieren: Minimalabst¨ande in Cadence greifen nur bei Außenkanten von Objekten, die das andere Objekt weder kreuzen, noch sich innerhalb von diesen befinden [4]. Kanten, die Leitungskreuzungen bilden, werden von diesem Test ignoriert.

Die Minimalabst¨ande von Verdrahtungsleitungen zueinander werden, um Leckstr¨ome zu minimieren, m¨oglichst groß gew¨ahlt. Innerhalb von Transistoren sind die Abst¨ande der Leitungen zueinander durch die kleinste erlaubte Kanall¨ange gegeben, die m¨oglichst gering gehalten wird. Die Einf¨uhrung eines strengen Maßes f¨ur die Verdrahtung und eines toleranten f¨ur Transistoren ist daher w¨unschenswert. Drei Wege, um dies zu erreichen, wurden untersucht:

1. Ein eigener Layer, der die Bereiche markiert, in denen die kleineren METAL-Maße erlaubt sind. Neben der daraus resultierenden Un¨ubersichtlichkeit des Layouts erh¨ohte der zus¨atzliche Layer den Zeichenaufwand zu sehr.

2. Umdefinieren der Verwendung des FET-Layers, so dass dieser das auszumessende Transistorgebiet und die daran anschließenden Source- und Drainfingerteile enth¨alt. Die Ermittlung der Kan¨ale w¨are bei den verwendeten Transistoren weiterhin ¨uber logische Operationen m¨oglich. Ein fehlerhaft eingezeichneter FET-Marker ist allerdings nicht mehr zu detektieren, da nun alle denkbaren GATE-METAL-Kombinationen auch n¨otig sind. Nahe beieinanderliegende Transistoren werden beim Expandieren bei der automatischen Erkennung von Transistoren (Kapitel 5.4.3) manchmal miteinander verbunden. Auch dies w¨are bei dieser Art von FET-Marker unm¨oglich zu detektieren.

3. Transistoren erzeugen, wenn sie nach dem strengen Regelsatz beurteilt werden, notwendigerweise Fehler. Die detektierten Fehler ¨uberlappen den Transistor jedoch in der Regel um einen gewissen Betrag.

   Alle Regionen, die nach dem strengeren Regelsatz Fehler enthalten, und die aktive Gebiete ¨uberlappen, kreuzen sicher die Region (inneres des Transistors), f¨ur die sie nicht gelten. Alle diese Gebiete zu verwerfen ist jedoch nicht sinnvoll, da damit auch echte Fehler verworfen werden.

   Formale Regeln, die echte Fehler in jedem Fall von solchen, die von Transistoren generiert werden, unterscheiden k¨onnen, existieren nicht. Implementiert wurde ein heuristischer Ansatz: Fehler, die keine aktiven Gebiete sind sicher echte Fehler. Solche, die diese um einen Mindestbetrag (siehe Tabelle 5.1) ¨uberlappen, werden ebenso als echte Fehler angesehen.

Dieser Kompromiss bedeutet f¨ur den Benutzer den geringsten Aufwand, und hat eine relativ hohe Treffsicherheit. Er wurde deshalb f¨ur die Implementierung gew¨ahlt.

F¨ur die Minimalbreiten von FETs wird nach demselben Schema verfahren.

Das Techfile unterst¨utzt f¨ur jeden Layer nur eine Linienbreite und einen Mindestabstand. Um dies zu umgehen wurde der zweite Parameter als Technologieparameter deklariert.

Um schlangenf¨ormige FETs bei Bedarf verbieten zu k¨onnen, wurde eine ¨uber den Schalter ‚allowserpentshapedfet‘ im DRC-Men¨u anw¨ahlbare Regel eingef¨uhrt, die nichtrechteckige aktive Regionen als Fehler mit dem Fehlercode MZ markiert. Alle im Zuge dieser Arbeit verwendeten Schalter sind in Anhang A.2 dokumentiert.

METAL

Leitungen außerhalb von Transistoren m¨ussen einen relativ großen Abstand zueinander haben, um Leckstr¨ome zu minimieren. Des Weiteren sollten sie breit sein, was den Leitungswiderstand gering h¨alt. Innerhalb von Transistoren sind oft mehrere Source- und Drain-Finger parallel geschaltet, und d¨urfen, um den Platzbedarf zu reduzieren, d¨unner sein als Einzelleitungen.

Eine kammf¨ormige Source/Drain-Struktur kann auch noch innerhalb eines kleinen Bereichs um die aktiven Regionen herum existieren. Minimale Linienbreiten und -Abst¨ande d¨urfen dort die Minimalwerte f¨ur Linien innerhalb von FETs annehmen.

Aktive Regionen

Kan¨ale, die k¨urzer als der Minimalwert sind, werden als Fehler markiert (Fehlercode MY). Eine Überpr¨ufung auf eine erlaubte Maximall¨ange ist in vorgesehen, kann aber mangels eines passenden L¨angen¨uberpr¨ufungsbefehls in der verwendeten Version von Cadence nicht implementiert werden. Kan¨ale mit einer L¨ange, die einen frei w¨ahlbaren Betrag (Tabelle 5.1) ¨ubersteigt, werden nicht mehr als solche erkannt. Der Leitwert des Polymerwerkstoffes, der die Kan¨ale bildet, ist niedrig genug, um diese N¨aherung zu rechtfertigen. Da Kan¨ale an Leitungskreuzungen entstehen, und bei der Verdrahtung nicht vermeidbar sind, werden FETs mit einem zu kleinem Umfang als parasit¨ar markiert. Der Layoutzeichner kann Transistoren mit Hilfe des FET- Hilfslayers manuell als gewollt markieren.

Sonstige Tests

Mit dem Verwendungszweck ‚`net`‘ abgelegte Objekte kommen in einem nichtextrahiertem Layout nicht vor, und werden als Fehler markiert.

Nicht achsenparallele GATE-, METAL-, PAD-, VIA- und FET-Kanten sind in der verwendeten Technologie bisher nicht vorgesehen, und gelten daher ebenso als Fehler.

Durch die Weise, auf die Cadence Layer ablegt (Bild 3.12), ist es m¨oglich, dass Hohlr¨aume in Objekten deren ¨außeren Rand ¨uberlappen. Solche nicht eindeutig definierten Formen werden ebenfalls als Fehler erkannt.

In den folgenden Tabellen (4.1–4.4) werden die getesteten Entwurfsregeln nach deren Art geordnet aufgelistet. Die Eintr¨age entsprechen den Nummern der Fehlermeldungen und sind zum leichteren Auffinden der Entsprechungen im Programmcode gedacht. Der Vorteil dieser Darstellungsart ist, dass implizit durch ande-

re Entwurfsregeln gegebene Regeln dargestellt werden k¨onnen. Deren Nummern wurden hier in Klammern gesetzt.

Bei den Überlappungsregeln ¨uberlappen die linken Layer die Unteren.

Tabellen, in denen die DRCs nach Layern aufgeschl¨usselt sind, sind in Anhang B zu finden.

Tabelle 4.1: Layer, die sich nicht ¨uberlappen d¨urfen

	HRES	CAPDEF	FET	VIA	PAD	GATE	METAL
CAPDEF							
FET	(NE)	N3					
VIA	(NE,NF)	N5	N6				
PAD	(NE,NF)	N8	N9	NA			
GATE	NB						
METAL	NC						
LETTER	ND		(NE)	(NE,NF)	(NE,NF)	NE	NF

Tabelle 4.2: Layerkombinationen, f¨ur die Minimalabst¨ande implementiert sind

	HRES	CAPDEF	FET	VIA	PAD	GATE	METAL	LETTER
HRES								
CAPDEF								
FET	(D6)	DH	D2					
VIA	(D6,D8)	DI	DJ	D3				
PAD	(D6,D8)	DK	DL	D4	D5			
GATE	D6					D7		
METAL	D8					D9,DA	DB,DC	
LETTER	DD		(DF)	(DF,DG)	(DF,DG)	DF	DG	

Tabelle 4.3: Layer, die sich ganzfl¨achig ¨uberlappen m¨ussen

	HRES	CAPDEF	FET	VIA	PAD	GATE	METAL
CAPDEF							
FET							
VIA							
PAD							
GATE			A1	A2	A3		
METAL				A4	A5		
LETTER							

Tabelle 4.4: Layerkombinationen, f¨ur die minimaleÜberlappungen implementiert sind

	HRES	CAPDEF	FET	VIA	PAD	GATE	METAL	LETTER
HRES								
CAPDEF								
FET							(X)[1]	
VIA								
PAD								
GATE			O2	O3	O4			
METAL			O5	O6	O7			
LETTER								

[1] Implizit gegeben ¨uber die minimale Kanalweite.

Tabelle 4.5: Sich nur auf einen Layer beziehende DRCs

	HRES	CAPDEF	FET	VIA	PAD	GATE	METAL	LETTER
Uneindeutige Formen	M1	M2	M3	M4	M5	M6	M7	M8
Nicht achsenparallele Kanten		M9	MA	MB	MC	MD	ME	
Minimale Breite	MF			MG	MH	MI	MJ,MK	
Kleinste Einkerbung	ML			MN	MO	MP	MQ	
Rechteckige Form			MR					

4.2 Extraktion

Um ein Layout mit dem Schaltplan vergleichen zu k¨onnen, ist es n¨otig, alle im Layout vorhandenen Bauteile zu detektieren und zu vermessen [23, 19, 20]. Neben den absichtlich generierten Elementen entstehen bei der Verdrahtung parasit¨are Bauteile (Bild 4.3), die von der konkreten Realisierung der Schaltung abh¨angen, und in genaue Schaltungssimulationen einbezogen werden m¨ussen. Die Aufgabe der Layoutextraktion ist es nun, alle im Layout enthaltenen Bauteile zu detektieren, zu vermessen, und in obige Gruppen einzuordnen.

4.2.1 Leitungsextraktion

Die Konfiguration der Leitungsdetektion wurde von den Entwicklern von Cadence sehr einfach gehalten: Der Programmierer definiert VIAs, die den METAL- und den GATE-Layer verbinden, und deklariert diese Layer dadurch als elektrisch leitf¨ahig.

In einem zweitem Schritt werden die Leitungen zusammen mit allen anderen Elementen, die nicht vermessen zu werden brauchen (Buchstaben, Vias und Pads) in das extrahierte Layout kopiert. Leitungen bekommen automatisch den Verwendungszweck ‚`net`‘ (Tabelle 3.2), alle anderen Objekte werden als ‚`drawing`‘ deklariert.

Durch einen Bug in den Anzeigeroutinen von Virtuoso 4.4.3 w¨are es n¨utzlich, die Leitungen innerhalb von Feldeffekttransistoren ausblenden zu k¨onnen: Leitungen werden generell ¨uber Bauteile gezeichnet. Transistoren, die aus vielen Source- und Drainfingern bestehen, ergeben deshalb, wenn zu klein dargestellt, ausgef¨ullte Fl¨achen der Farbe der Metallumrandung, anstatt als Bauteil markiert zu sein.

Einfaches Herausschneiden aller METAL-Leitungen innerhalb von Transistoren behebt zwar dieses Problem, daf¨ur wurden aber die dadurch getrennten Leitungen von Virtuoso als unterbrochene Leitungen behandelt. Das LVS funktioniert weiterhin: Die Information ¨uber die Zugeh¨origkeit von Netzknoten wird vom Inhalt der Layer getrennt abgespeichert [6, `geomStamp`].

Um Transistoren dennoch deutlich sichtbar zu machen, wurde die Weite aller Bauteilumrandungen auf 2 Pixel erh¨oht.

Leitungen, die aus Polygonen mit einer zu hohen Anzahl von Eckpunkten bestehen, werden in kleinere Leitungen aufgeteilt. Die Information, welche Leitung mit welcher verbunden ist, bleibt dabei f¨ur LVS und Simulation erhalten. Auch hier wird das Layout weiterhin korrekt abgearbeitet, selbst wenn zwei dieser Polygone innerhalb eines Transistors aufeinanderstoßen. Die einzige Ausnahme bildet auch hier der Layout-Editor Virtuoso, der getrennte Leitungsteile als getrennte Polygone betrachtet.

Nachdem in der Literatur nichts ¨uber die maximale Anzahl von Eckpunkten einer Leitung in Erfahrung zu bringen war, wurde experimentell ein maximaler Wert von 1024 Eckpunkten ermittelt. Dank der hohen Anzahl der Source/Drain-Finger innerhalb eines Transistors wird dieser Maximalwert hier im Gegensatz zur CMOS-Technologie h¨aufig erreicht. Die Begrenzung auf 1024 Eckpunkte scheint auf verschiedene Teile des Extraktors zuzutreffen (Kapitel 4.2.4, Seite 27).

4.2.2 Bauteilextraktion

In der Datei `divaEXT.rul` werden analog zum DRC Hilfslayer, die dieses mal die Umrisse aller zu detektierenden Bauteile eines Typs enthalten, generiert. Der DRC-Befehl ist hier nicht mehr verwendbar, da der Extraktor ihn nicht unterst¨utzt. Zus¨atzlich kommt hier die Einschr¨ankung, dass keine Befehle, die mit einzel-

nen Objekten arbeiten, existieren, hier richtig zu tragen. So ist es zum Beispiel nicht m¨oglich, jeden einzelnen Transistor getrennt zu expandieren, was Überlappungen der expandierten Transistoren vermeiden w¨urde.

Als n¨achstes definiert `divaEXT.rul` Anzahl, Namen und Layer der Anschl¨usse (Pins), die das Bauteil haben soll. Cadence geht nun die Objekte in den Erkennungslayern einzeln durch und versucht, Kontakte mit diesen zu verbinden. Ist dies nicht eindeutig m¨oglich, wird eine Warnmeldung mit der linken oberen Bauteilkoordinate in das CIW (Bild 3.3) ausgegeben.

Da Source- und Drainanschluss der Transistoren den selben Layer (METAL) teilen, wird automatisch davon ausgegangen, dass beide Anschl¨usse gegeneinander austauschbar sind.

Der Extraktor unterst¨utzt eine Vielzahl von Messbefehlen, die die Parameter (Kanall¨ange, etc. Siehe auch Kapitel 3.2.3) der detektierten Bauteile ermitteln. Alle Messbefehle messen relativ zum Erkennungslayer des Bauelements. Da jedes Bauteil einen eigenen Erkennungslayer besitzt, wird auf diese Weise die Festlegung des Bauteil, auf das sich die Messung bezieht, ¨uberfl¨ussig. Nachteil dieser Vorgehensweise ist die Notwendigkeit einer Vielzahl von Messbefehlen.

4.2.3 Kondensatoren

Bei Kondensatoren in Halbleitertechnologie ist es aufgrund der extremen Geometrien notwendig, die Kapazit¨at in eine Fl¨achen- und eine Kantenkapazit¨at aufzuteilen (Bild 4.2). Die Fl¨achenkapazit¨at entspricht der ‚klassischen‘ Kapazit¨at eines Plattenkondensators, die Kantenkapazit¨at wird durch E-Felder an den Kanten der Kondensatorplatten hervorgerufen.

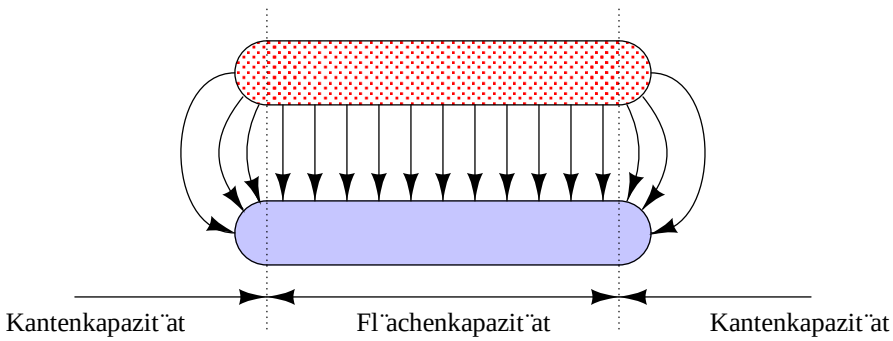


Abbildung 4.2: Kanten- und Fl¨achenkapazit¨at

Bei herk¨ommlichen Siliziumtechnologien ist die Kapazit¨at einer Polysiliziumleitung in allen Layerkonfigurationen n¨aherungsweise gleich. Alle Kapazit¨atsanteile außer der Kapazit¨at zwischen dieser Schicht und dem Halbleitermaterial sind durch die großen Abst¨ande zu diesen beiden Schichten gering genug, um vernachl¨assigt werden zu k¨onnen.

Im Gegensatz dazu d¨urfte sich die Fl¨achenkapazit¨at von fertigen Kondensatoren stark von der von METAL- und GATE-Leitungen allein unterscheiden: Außerhalb von GATE/METAL- Überlappungsgebieten fehlt die zweite Kondensatorplatte. Da jeder Layer im Techfile nur einen einzigen Eintrag f¨ur Fl¨achenkapazit¨aten besitzt, wurde die Fl¨achenkapazit¨at f¨ur Kondensatorgebiete formal dem Layer CAPDEF zugewiesen.

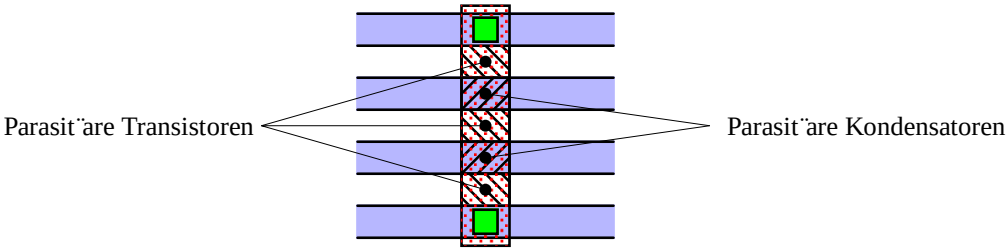


Abbildung 4.3: Parasit¨are Bauteile

4.2.4 Extraktion parasitärer Bauteile

Es ist nicht zu vermeiden, dass durch die Verdrahtung parasit¨are Transistoren und Kondensatoren entstehen (Bild 4.3). Die Extraktion dieser Bauteile erh¨oht die Genauigkeit von Schaltungssimulationen.

Parasit¨are Kondensatoren

Parasit¨are Kondensatoren entstehen ¨uberall dort, wo METAL- und GATE-Leitungen sich kreuzen. Sie werden als eigenes Bauteil (`pcapacitor`) erkannt. Parasit¨are Kondensatoren werden beim „Layout gegen Schaltbild“-Vergleich ignoriert.

Innerhalb von Feldeffekttransistoren bildet jeder Source-oder Drainfinger eine Kapazit¨at. Um die Anzahl der zu extrahierenden Bauteile drastisch zu reduzieren und gleichzeitig das extrahierte Schaltbild ¨ubersichtlicher zu machen, wird die GATE-METAL-Kapazit¨at von Transistoren als Ganzes ermittelt und in die CDF-Beschreibung des Transistorenmodells (Bild 3.5) als Parameter `Cgm` eingetragen, anstatt sie f¨ur jeden einzelnen Finger als eigenes Bauteil zu extrahieren. Die Aufteilung in Source– Gate- und Gate–Drain-Kapazit¨at geht dabei verloren: Die Funktionen, die verwendet werden, um Bauteile zu vermessen, k¨onnen zwei Anschl¨usse, die sich im selben Layer befinden, nicht voneinander unterscheiden. Bei den hier verwendeten hohen Mengen von Kan¨alen pro Transistor ist eine Ungenauigkeit der Kapazit¨atsverteilung um einen halben Finger pro Source- bzw. Drainanschluss vertretbar. Bei Transistoren mit einem Kanal oder einer ungeraden Anzahl von Kan¨alen ist die Kapazit¨at auf beide Anschl¨usse gleich verteilt.

Parasit¨are Transistoren

Parasit¨are Transistoren (Bild 4.3) entstehen ¨uberall, wo zwei METAL-Leitungen eine GATE-Leitung ¨uberkreuzen.

Die Str¨ome, die durch Transistoren mit weit auseinanderliegenden METAL-Leitungen fließen, sind sehr gering. Transistoren, deren Kan¨ale l¨anger als `gunDetectMaxLength` (Tabelle 5.1) sind, werden daher ignoriert. Der Parameter `gunDetectMaxLength` wird im Techfile definiert, kann aber ¨uber das CIW (Bild 3.3) editiert werden.

Alle Transistoren, die nicht vom Benutzer markiert, bzw. automatisch erkannt wurden und obige Bedingung erf¨ullen, werden als eigenes Bauteil (`pofet`) erkannt. Das Modell `pofet` unterscheidet sich vom Transistormodell `ofet` f¨ur erw¨unschte Transistoren in zwei Punkten: Die Parameter `Nr` f¨ur die Anzahl der Kan¨ale und die Gate-METAL-Kapazit¨at der Source/Drainfinger innerhalb des Transistors (`Cgm`) sind nicht definiert: Die f¨ur Transistoren mit mehreren Kan¨alen notwendige Interdigitalstruktur ist als parasit¨ares Bauteil nicht denkbar. Sollten dennoch zwei parasit¨are Transistoren dieselben Source- und Drainknoten teilen, werden sie korrekt als

getrennte, parallelgeschaltete Transistoren extrahiert.

Leitungswiderst¨ande und -Kapazit¨aten

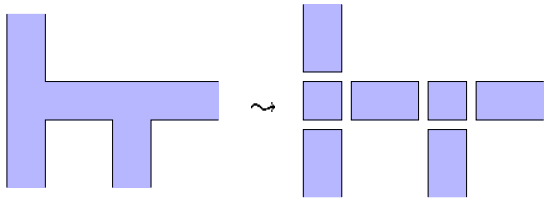


Abbildung 4.4: Auftrennung einer Leitung in einzelne Segmente

Cadence erlaubt das Extrahieren von Leitungswiderst¨anden und Substratkapazit¨aten [5]. Dazu teilt das System alle Leitungen in einzelne Segmente auf (Bild 4.4). Jedem Segment wird nun ein eigener Innenwiderstand und eine Substratkapazit¨at zugewiesen. F¨ur Ecken und Leitungseinengungen sindÜbergangswiderst¨ande vorgesehen. Jedem Segment wird eine eigene Netzknotennummer (Kapitel 3.4.1) zugewiesen.

Substratkapazit¨aten sind nur auf Layer anwendbar, f¨ur die ein Schichtwiderstand definiert ist.

Die Aufteilung der Leitungen in einzelne Segmente muss vor der Definition der Durchkontaktierungen erfolgen. Diese muss wieder vor der Extraktion der Bauelemente stehen. Nach der Definition der Schichtwiderst¨ande stellt jeder Source-Drainfinger einen eigenen Netzwerkknoten dar, der mit den anderen ¨uber Widerst¨ande verbunden ist. F¨ur die Transistorenextraktion ist es allerdings notwendig, dass alle Source- und alle Drainanschl¨usse denselben Netzwerkknoten darstellen. Ein versuchsweises Auftrennen aller Transistoren in kleine Transistoren mit nur einem Kanal scheiterte ebenfalls:

Die Definition von Schichtwiderst¨anden f¨ur Layer mit Objekten von einer gr¨oßeren Anzahl von Eckpunkten (ca. 1000, siehe Kapitel 4.2.1, Seite 24) f¨uhrt zu einem Cadence-internem Fehler (Schutzverletzung) und Programmabbruch. Da sich die vorliegende Technologie durch eine hohe Zahl von Eckpunkten des METAL-Layers auszeichnet, ist die verwendete Installation von Cadence 4.4.3 offenbar hierf¨ur ungeeignet.

4.2.5 Leckströme

Leckstr¨ome von der Verdrahtung durch das Substrat werden in allen etablierten Technologien normalerweise vernachl¨assigt. L¨angere Leitungen sind von allen Seiten her mit SiO_2 isoliert, das ein guter Isolator ist. In den ¨alteren Technologien (Bipolar, P- und NMOS) fließen zus¨atzlich in allen Netzknoten relativ hohe Str¨ome, so dass Leckstr¨ome durch das Substrat nicht relevant sind.

Dies ist in der vorliegenden Technologie zumindest f¨ur den METAL-Layer nicht gew¨ahrleistet (Bild 3.1). Obwohl nicht in Cadence vorgesehen, wurde deshalb nach einer Methode gesucht, diese Art von Leckstr¨omen zu simulieren.

Hierzu wird eine Kopie des METAL-Layers als VIA verwendet, um diesen ganzfl¨achig mit einem Substrat-Hilfslayer zu verbinden.

Um Substratwiderst¨ande zu definieren, ist es nicht notwendig, den METAL-Layer in einzelne Segmente zu zerlegen. Transistoren werden also korrekt extrahiert. Die Gr¨oße der zu verwendenden Polygone ¨ubersteigt jedoch auch hier die M¨oglichkeiten des Programms.

Auch der Versuch, Leckstr¨ome ¨uber ein eigenes Leitungsbauteil zu realisieren, scheiterte an der selben Fehlermeldung (`Segment Violation`). Dieses h¨atte erm¨oglicht, jede Leitung ¨uber einen Widerstand mit einem konstantem Potential zu verbinden. Ein eigenes Bauteilmodell f¨ur Leitungen hat zwei große Nachteile:

- Leckstr¨ome fließen in Wirklichkeit nicht zwischen den Leitungen und einem konstantem Potential, sondern zwischen den Leitungen selber, die unterschiedliche Potentiale besitzen Dies stellt die Genauigkeit des Modells in Frage.
- Diva bietet kein passendes Kriterium f¨ur die Gr¨oße des Substrat-METAL-Widerstands an: Dieser ist weder von der Fl¨ache der Leiterbahn abh¨angig (Strom fließt zwischen den Kanten von Leiterbahnen), noch von deren Umfang (der Widerstand ist proportional zum Abstand der Objekte zueinander).

4.3 LVS

Der Layout-Versus-Schematic-Check [16, 23] ben¨otigt nur ein Minimum an vom Entwickler geschriebenen Code. Nahezu alle technologiespezifischen Eigenschaften werden von Bauteilmodell und Extraktionsprogramm verwaltet. Die umfangreiche, allerdings l¨uckenhafte Dokumentation macht die Konfiguration des LVS allerdings trotzdem zeitaufw¨andig. Neben der offiziellen Cadence-Dokumentation wurde beim Konfigurieren des LVS Gebrauch von Dokumentation und Quellcode des EMACS-Texteditors [8, 9] gemacht, der in einer ¨ahnlichen Programmiersprache (Lisp) geschrieben wurde.

Zur Vorbereitung eines LVS ist es n¨otig, Layout und Schaltplan zu extrahieren. Beim Layout geschieht dies ¨uber die‚Extrakt‘-Funktion von Virtuoso. Der Schaltplan wird ¨uber die‚Check and Save‘-Funktion des Schaltplaneditors (Bild 3.6) extrahiert. Ein Speichern allein erf¨ullt diesen Zweck nicht.

Die gesamte LVS-Konfiguration findet in der Datei `divaLVS.rul` Platz. Der erste Teil besteht aus SKILL-Funktionen, die zwei Bauteile anhand ihrer CDF-Beschreibung (Kapitel 3.2.3) vergleichen. Der aufw¨andige Weg ¨uber SKILL-Funktionen muss gegangen werden, um die gr¨oßtm¨ogliche Flexibilit¨at zu gew¨ahrleisten [23]. So k¨onnen zum Beispiel beliebig geartete Toleranzen f¨ur die einzelnen Parameter implementiert werden. Die Vergleichsfunktionen k¨onnen entweder eine Fehlermeldung zur¨uckliefern oder den Wert**nil**‘, der anzeigt, dass die Bauteile als identisch gelten k¨onnen.

Des Weiteren ist es sinnvoll, Funktionen einzuf¨uhren, die z.B. parallel geschaltete Transistoren zu einem großen Transistor verbinden. Die zugeh¨origen Funktionen liefern den neuen Satz Bauteilparameter, oder‚`nil`‘, wenn die Vereinigung nicht m¨oglich war. Ein Beispiel f¨ur Bauteile, die eine Vereinigung notwendig machen, sind Transistoren mit mehreren Gates, die aufgrund zu breiter Source/Drainstege vom Extraktor getrennt wurden (Tabelle 5.1). Das Vereinfachen beider zu vergleichenden Schaltungen verringert außerdem den Aufwand beim eigentlichen Vergleich, und erm¨oglicht, elektrisch ¨aquivalente Schaltungen als identisch zu erkennen. Da Transistoren keine linearen Bauteile sind, und die meisten bei der Vereinigung von Transistoren ben¨otigten Parameter noch nicht gen¨ugend weit erforscht sind [30], beschr¨ankt sich die implementierte LVS-Implementation auf parallel geschaltete Transistoren mit gleichen Kanall¨angen.

Neben dieser konventionellen Vereinfachung der zu verlgeichenden Schaltungen ist es auch m¨oglich,‚pseudoparallele‘ Transistoren zu vereinen [14]. Diese Transistoren sind nicht wirklich parallel, das Vereinen derselben erzeugt aber im Ramen einer Simulation des Transistors als digitalen Schalter keine zus¨atzlichen Fehler. Ein Beispiel f¨ur pseudoparallele Transistoren ist in Bild 4.5 zu sehen.

Dieser Schritt ver¨andert allerdings gr¨ossere Teile der Schaltung, als die letzten. Da dies die Fehlersuche meist erschwert [16], ist er nicht zu empfehlen.

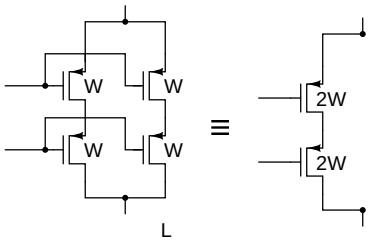


Abbildung 4.5: Vereinigung pseudoparalleler Transistoren

Parallel- und in Reihe geschaltete Kondensatoren werden immer vereint.

Cadence erkennt automatisch, dass Source- und Drainanschluss von Transistoren beliebig vertauschbar sind, da beide Anschl¨usse bei der Detektion denselben Layer teilen (Kapitel 4.2.2, Seite 25). Bei Kondensatoren muss dies von der LVS-Konfiguration aus festgelegt werden.

Der dritte Teil der LVS-Definitionen sagt nun, welcher der oben definierten Schritte in welcher Reihenfolge abgearbeitet wird.

Cadence versucht nun, alle parallel oder in Reihe verbundenen Bauteile zu vereinen, bis die Vereinigungsfunktion f¨ur jede m¨ogliche Kombination `nil` liefert. Danach wird mit den obigen Funktionen verglichen, welcher Transistor im Layout welcher im Schaltplan sein kann, und die Kombination gesucht, in der beide am besten ¨ubereinstimmen.

4.3.1 Fehlersuche

Prinzipiell kann ein LVS-Test nur herausfinden, ob Layout und Schaltplan identisch sind. Selbst bei kleinen Fehlern sind die Zuordnungen der einzelnen Schaltungsteile zueinander nicht mehr eindeutig ermittelbar. Um den Fehlerort besser eingrenzen zu k¨onnen, werden benannte Pads als Referenzpunkte behandelt, die sich in beiden Schaltungen an der selben Stelle befinden. Virtuoso erlaubt, weitere benannte Referenzpunkte in ein Layout einzuzeichnen.

Über den Men¨upunkt‚*Backannotate*‘ ist ein Programm erreichbar, das die verglichenen Bauteile und vermutlichen Fehler in den Schaltplan eintr¨agt. Auch bei der Programmierung dieses Programms wurde Wert auf Eigenst¨andigkeit gelegt: Es ist von der Kommandozeile aus startbar und gibt seine Fehlermeldungen nicht in das CIW aus. Sie k¨onnen daf¨ur ¨uber die‚*Monitor*‘-Funktion des LVS- Programms erreicht werden [22].

Kapitel 5

Implementierung

5.1 Bauteilmodelle

Die verwendeten Bauteilmodelle lassen sich in zwei Klassen aufteilen: Symbole (Symbol) f¨ur Schaltpl¨ane, und parametrisierte Zellen (P-Zellen, englisch PCells) f¨ur extrahierte Bauteile. Erstere sind umfangreich dokumentiert [2]. Die Dokumentation von P-Zellen hingegen beschr¨ankt sich auf wenige Abs¨atze [3].

5.1.1 Symbole

Jedes Bauteil, das in ein Schaltbild eingezeichnet werden soll, muss ein eigenes Erscheinungsbild haben. Dieses wird unter dem Namen des Bauteils und der Ansicht `symbol` im Library Manager (Bild 3.4) abgelegt.

Symbole werden im ‚Composer—Symbol‘ (Bild 3.6) gezeichnet. Um den Composer das erste Mal mit einem Symbol aufzurufen, w¨ahlt man zuerst die Bibliothek, in der das Symbol abgelegt werden soll, im Library Manager (Bild 3.4) aus. Unter dem Men¨upunkt *View/New/Cell View* ist nun ein Fenster erreichbar, in dem die Namen des Bauteils und der Ansicht `(symbol)` einzutragen sind. Auf die Frage, mit welchem Programm das Objekt ge¨offnet werden soll, ist mit‚Composer—Symbol‘ zu antworten.

Alle Objekte, die n¨otig sind, um ein Symbol zu zeichnen, k¨onnen im Composer unter dem Men¨upunkt *‚add‘* erreicht werden. Beim Einzeichnen von Pins ist zu beachten, dass der Bauteilextraktor die Pinnamen aus dem Symbol ¨ubernimmt.

5.1.2 P-Zellen

Bauelemente m¨ussen auch im extrahierten Layout ein Erscheinungsbild haben. Dies wird in P-Zellen abgelegt.

P-Zellen bestehen aus drei Teilen:

- Ein Bauteilsymbol, das eine feste Gr¨oße besitzt, und in der linken oberen Ecke des Bauteils angezeigt wird.
- Eine ‚Recognition Shape‘, einem Umriss von der Form des Bauteils, der im extrahierten Layout das Bauteil markiert.

- Eine Liste der Anschl¨usse des Bauteils.

  Die Parameter wie Kanall¨ange und Kapazit¨at werden im CDF-Editor (Bild 3.5) definiert.

Cadence bietet einen fertigen Befehl `ivCreatePCells` f¨ur die automatische Generierung von P-Zellen an. Er liest eine Liste der zu generierenden Zellen und die der zugrundeliegenden Symbole aus einer Datei. Die Gr¨oße des Symbols, das in der linken oberen Ecke des Bauteils abgebildet wird, muss vom Benutzer angegeben werden. Eine feste Einheit der Symbolgr¨oße war im Rahmen dieser Arbeit nicht ermittelbar. Die restlichen ben¨otigten Daten sind in der Symbolansicht enthalten. Das Format der Symbolliste ist folgendes:

```
ptest       ofet     symbol        auLvs                    50
ptest       cap      symbol        auLvs                    50
ptest       pofet    symbol        auLvs                    50
ptest       pcap     symbol        auLvs                    50

#Bibliothek  Bauteil  Quellansicht  zu generierende Zelle  Symbolgröße
```

In der vorliegenden Arbeit wurde sie unter dem Namen `pcells.txt` abgelegt.

Kommentare sind —anders als in allen anderen verwendeten Dateien— nicht erlaubt. Fehler im Dateiformat verursachen das Ende der Bearbeitung der Datei, aber keine Fehlermeldungen. Die Symbolliste wird eingelesen, indem in das CIW folgender Befehl eingegeben wird:

```
(ivCreatePCells "pcells.txt")
```

Die generierten P-Zellen werden im CIW mitprotokolliert. Es bleibt dem Benutzer ¨uberlassen, zu ¨uberpr¨ufen, ob die richtigen Zellen generiert wurden.

Der in der Dokumentation des `ivCreatePCells`-Befehls [3] empfohlene Name der Ansicht der P-Zellen (`ivpcells`) wird vom Extraktor akzeptiert. Im LVS-Programm ist jedoch der Ansichtname `auLvs` fest eingestellt, so dass auf diesen Namen ausgewichen werden musste.

5.1.3 Die verwendeten Bauteile

F¨ur Extraktion und LVS ist es n¨otig, folgende Bauteile anzubieten:

`ofet` Ein **o**rganischer **F**eld**e**ffekt**t**ransistor. Die Namen der Source-Drain- und Gate-Anschl¨usse werden von den Dateien `divaEXT.rul` und `divaLVS.rul` als ‚S‘, ‚D‘ und ‚G‘ angenommen. Die CDF-Beschreibung eines Transistors (Kapitel 3.2.3) enth¨alt folgende Parameter:

- `W`, `L`: Weite und L¨ange des Kanals
- `Nr`: Anzahl der Kan¨ale
- `Cgm`: Kapazit¨at der Source/Drainfinger gegen das Gate.

`pofet` Ein parasit¨arer `ofet` (Kapitel 4.2.4). Parasit¨are Transistoren haben nur einen Kanal, und ben¨otigen deshalb nur zwei Parameter, `W` und `L`.

cap Kondensatoren (**Cap**acitors). Kondensatoren sind Fl¨achen, an denen sich GATE- und METAL-Layer ¨uberlappen, und die als Kondensator markiert sind. Der einzige Parameter eines Kondensators ist dessen Kapazit¨at, C. Die Anschl¨usse besitzen die Namen‚CGATE‘ und ‚CMETAL‘.

pcap Parasit¨are Kondensatoren sindÜberlappungen von METAL- und GATE-Layer, die nicht als Kondensator markiert sind. Sie entstehen z.B. bei der Verdrahtung (Bild 4.3). Kondensatoren innerhalb von Feldeffekttransistoren werden ¨uber den Parameter *Cgm* verwaltet. pcap ist eine direkte Kopie des Modells cap.

5.2 display.drf

Die Datei display.drf muss in das Verzeichnis, in dem sich die Bibliothek ptest befindet, kopiert werden. Sie enth¨alt alle anzeigespezifischen Informationen ¨uber das Aussehen der Layer. F¨ullfarbe, F¨ullstil und Beschreibung der Umrandung von Objekten eines Layers werden in Farbpaketen zusammengefasst, die im Techfile (Kapitel 5.3.2) ihren Layern zugewiesen werden.

5.2.1 Definition der Anzeigeeinheit

Im Gegensatz zum Bildschirm, auf dem Kanten meist weiß vor schwarzem Hintergrund markiert werden, verwenden Plotter weißes Papier, auf dem schwarze Kanten sinnvoll sind.

Einige Ausgabeger¨ate (z.B. Plotter) haben außerdem eine eingeschr¨ankte Farbtiefe. Es ist also sinnvoll, mehrere Farbschemata f¨ur verschiedene Anzeigen zu definieren. Im vorliegendem Fall wird der Name ‚display‘ (Anzeige) als Ger¨atename definiert:

```
drDefineDisplay(
 (display)
)
```

5.2.2 Füllmuster

F¨ullmuster werden als Bitmap definiert. Cadence erwartet F¨ullmuster als Listen, in denen Pixel der F¨ullfarbe durch Einsen, freie Pixel durch Nullen repr¨asentiert werden. Die F¨ullmusterdefinition wird durch folgenden Befehl eingeleitet:

```
drDefineStipple(
```

Als Beispiel sei hier das F¨ullmuster‚slash‘ (Schraffur) angegeben.

```
( display       slash
 (
  ( 0 0 1 0 0 0 0 0 )
  ( 0 1 0 0 0 0 0 0 )
```

```
  ( 1 0 0 0 0 0 0 0 )
  ( 0 0 0 0 0 0 0 1 )
  ( 0 0 0 0 0 0 1 0 )
  ( 0 0 0 0 0 1 0 0 )
  ( 0 0 0 0 1 0 0 0 )
  ( 0 0 0 1 0 0 0 0 )
 )
```

5.2.3 Linienmuster

Linienmuster werden ebenso als Bitmap definiert:

```
(drDefineLineStyle
;Anzeige  Mustername  Linienbreite  Füllmuster)
 display    solid          1         (1 1 1)
)
```

5.2.4 Farben

Farben werden ¨uber ihren Rot-, Gr¨un- und Blau-Anteil definiert, wobei 0 den dunkelsten Wert bedeutet. Zu beachten ist, dass hier auch auf Druckern additive Farbmischung verwendet wird (Rot und Gr¨un ergibt zum Beispiel Gelb.)

```
drDefineColor(
 ;(  DisplayName  Farbname  Rot  Grün  Blau  )
  (  display      white     255  255   255   )
  (  display      black     0    0     0     )
  (  display      red       255  0     0     )
  (  display      green     0    255   0     )
  (  display      blue      0    0     255   )
  (  display      purple    255  0     192   )
  (  display      cyan      0    192   192   )
  (  display      yellow    255  255   0     )
)
```

5.2.5 Layerdefinitionen

Je ein Satz von Linien- und F¨ullfarben und -Mustern wird zu einem Farbpaket zusammengefasst:

```
drDefinePacket(
 ;(  Anzeige  Paket     Füll-    Linien-  Füll    Ramen-  )
 ;                      muster   muster   Farbe   Farbe   )
  (  display  METAL     Metal    solid    blue    white   )
  (  display  METALnet  blank    solid    blue    blue    )
  (  display  METALbnd  blank    solid    white   white   )
        ⋮        ⋮        ⋮        ⋮        ⋮       ⋮
  (  display  CAPDEF    blank    solid    cyan    cyan    )
  (  display  FET       blank    solid    purple  purple  )
  (  display  device    blank    solid    yellow  yellow  )
)
```

Die Zuweisung der Farbpakete zu den Layern wird in Kapitel 5.3.2 (Seite 38) beschrieben.

5.3 techfile.txt

Die globale Technologiebeschreibung f¨ur Cadence steht in einem so genannten Techfile [7]. Neben der Konfiguration von Anwendungsprogrammen, wie Autoroutern oder Simulatoren, die außerhalb des Themenbereichs dieser Arbeit liegen, z¨ahlt hierzu die Definition der verwendeten Layer und die Festlegung globaler Parameter, wie die elektrischen Eigenschaften der einzelnen Layer (Schichtwiderstand, Kantenkapazit¨at,...) und die Entwurfsregeln.

Alle in den Textbeispielen verwendeten Zahlenwerte sind Beispielwerte.

5.3.1 Technologieparameter

Alle im Techfile festgelegten Parameter k¨onnen ¨uber die komfortablen Men¨us des CIW (Bild 3.3) editiert werden. Um dies m¨oglichst optimal nutzen zu k¨onnen, sind alle als Zahl gegebenen technologieabh¨angigen Parameter in diese Datei integriert.

Benannte Technologieparameter

Die Konfigurationsdateien f¨ur DRC (Kapitel 4.1) und Bauteilextraktion teilen denselben Code f¨ur die Detektion einiger wichtiger Bauteile. Einige Einstellungen (Tabelle 5.1) ben¨otigen technologieabh¨angige Zahlenwerte, die urspr¨unglich in Variablen abgelegt wurden. Um diese Zahlenwerte konfigurierbar zu machen, wurden sie in das Techfile integriert:

```
controls(
 techParams(
```

```
   ; Parameter              Wert
  (gunChannelMaxWidth       1   )
  (gunChannelMaxWidth       1   )
  (gunChannelMinWidth       2   )
  (gunMaxCrossingPerimeter  3.7 )
  (gunMaxViaPerimeter       9081)
              ⋮               ⋮
 )
)
```

Tabelle 5.1: Benannte Technologieparameter

gunChannelMaxLength	Maximal erlaubte Kanall¨ange von Transistoren.
gunChannelMinLength	Minimal erlaubte Kanall¨ange.
gunMaxCrossingPerimeter	Schwellwert, um erw¨unschte von parasit¨aren Transistoren unterscheiden zu k¨onnen.
gunMaxViaPerimeter	Schwellwert, um Vias von Pads zu trennen.
gunMetalSep	Minimaler Abstand von Metallteilen innerhalb von Transistoren.
gunMetalWidth	Minimale Source/Drainfingerbreite.
gunSDMaxWidth	Maximale Source/Drainfingerbreite. METAL-Objekte ¨uber dieser Breite trennen den Transistor.
gunFetSepWidth	Erlaubte Entfernung der Source/Drainfingern von den Kan¨alen eines Transistors.
gunDetectMaxLength	Maximale detektierbare Kanall¨ange.

Abstandsregeln

Die Abstandsregeln f¨ur den DRC k¨onnen erst nach der Definition aller Layer (n¨achstes Kapitel) festgelegt werden. Ihre Definition funktioniert folgendermaßen:

```
physicalRules(
 spacingRules(
   ; Regeltyp    Layer1   Layer2  Wert
  (minSpacing    "METAL"          1  )
  (minNotch      "METAL"          2.3)
  (minWidth      "METAL"          7  )
  (minEnclosure  "METAL"  "VIA"   9  )
  (minSpacing    "METAL"  "HRES"  21 )
        ⋮          ⋮        ⋮      ⋮
 )
)
```

Ebenso wie schon die Variablen werden auch die Abstandsregeln nur von den im Zuge dieser Arbeit geschriebenen Konfigurationsdateien verwendet. Cadence macht keinen selbstst¨andigen Gebrauch davon. Die verwendeten Regeltypen sind in Tabelle 5.2 aufgelistet.

Tabelle 5.2: Die verwendeten Abstandsregeltypen

minSpacing	Minimaler Abstand zwischen zwei Objekten.
minNotch	Kleinste erlaubte Einkerbung innerhalb eines Objekts.
minWidth	Minimale Weite z.B. von Leitungen.
minEnclosure	Definiert eine minimale Überlappung eines Layers über einen Anderen.

Elektrische Parameter

Auf die Abstandsregeln folgt die elektrische Charakterisierung der verwendeten leitfähigen Schichten. Dies sind im vorliegenden Fall die Flächenkapazität (`areaCap`) und Kantenkapazität (`edgeCapacitance`) eines Layers:

```
electricalRules(
 characterizationRules(
   ; Parameter      Layer1   Layer2  Wert
  (areaCap          "GATE"           1e-03 )
  (edgeCapacitance  "METAL"          5.0e-9)
          ⋮            ⋮               ⋮
 )
)
```

5.3.2 Layer

Layer und Purposes (Kapitel 3.4) werden in Cadence über Nummern repräsentiert. Einige Nummern sind von den Anwendungsprogrammen fest vergeben. Um dem Benutzer die Arbeit zu erleichtern, ist es sinnvoll, diesen Objekten Namen zuzuweisen, was folgendermaßen erledigt wird:

```
layerDefinitions(
 techPurposes(
  (Name     Nummer  Abkürzung)
  (warning   234      wng   )
  (tool1     235      tl1   )
  (tool0     236      tl0   )
  (label     237      lbl   )
  (error     239      err   )
      ⋮       ⋮        ⋮
 ) ;techPurposes
```

Die erste Spalte enthält die Namen der Purposes. Die darauf folgende Zahl ist die Nummer, der man diesen Namen zuweist. Die dritte Spalte enthält eine Abkürzung, auf die Anwendungen bei Platzmangel ausweichen können.

Obige Zeilen beschreiben den minimalen Satz an Verwendungszwecken, den Cadence verwendet. Eigene Purposes k¨onnen hinzugef¨ugt werden. Um Platz f¨ur Erweiterungen seitens des Systems freizuhalten, sind Nummern kleiner als 200 reserviert.

Die Layer werden analog definiert. Auch hier sind die Nummern ¨uber 200 f¨ur systeminterne Zwecke vorgesehen:

```
techLayers(
 (Name      Nummer  Abkürzung   )
 (CAPDEF    7       CAPDEF      )
 (FET       8       FET         )
 (HRES      10      HRES        )
 (METAL     11      METAL       )
 (VIA       13      VIA         )
 (GATE      14      GATE        )
 (LETTER    15      LETTER      )

  ;Vom System reservierte Layer:
 (Unrouted  200     Unroute     )
 (Row       201     Row         )
 (Group     202     Group       )
     ⋮       ⋮         ⋮
)
;techLayers
```

Die vollst¨andige Liste aller Cadence-internen Layer l¨asst sich in [7, Appendix A] nachlesen.

Eine wichtige weitere Eigenschaft von Layern ist die Anzeigepriorit¨at. Sie definiert, in welcher Reihenfolge die Layer gezeichnet werden:

```
techLayerPurposePriorities(
  (GATE      drawing )
  (GATE      net     )
  (LETTER    drawing )
  (VIA       drawing )
      ⋮         ⋮
  (Unrouted  drawing9)
 )
```

Die Layer werden in aufsteigender Reihenfolge ihrer Priorit¨at aufgelistet.

Nachdem die Layer definiert worden sind, ist es nun noch n¨otig, ihnen Farbpakete (siehe Kapitel 5.2.5, `display.drf`) zuzuweisen:

```
techDisplays(
;(Layer   Purpose  Packet     Attribute    )
 (HRES    drawing  HRES      t  t  t  t    t)
 (METAL   drawing  METAL     t  t  t  t    t)
 (METAL   net      METALnet  t  t  t  nil  t)
 (VIA     drawing  VIA       t  t  t  t    t)
 (VIA     net      VIAnet    t  t  t  nil  t)
 (GATE    drawing  GATE      t  t  t  t    t)
 (GATE    net      GATEnet   t  t  t  nil  t)
 (LETTER  drawing  LETTER    t  t  t  nil  t)
 (CAPDEF  drawing  CAPDEF    t  t  t  t    t)
 (FET     drawing  FET       t  t  t  t    t)
    ⋮       ⋮        ⋮      ⋮  ⋮  ⋮  ⋮    ⋮
)
```

Die vollst¨andige Liste ist auch hier wieder in [7] nachzulesen.

Die Attribute f¨ur den `techDisplays`-Befehl sind in Tabelle 5.3, [31] abgebildet. Sie k¨onnen die Werte **`t`** und **`nil`** annehmen.

Tabelle 5.3: Attribute von Layern

Vis	**`t`** bedeutet, der Layer ist beim Start von Virtuoso automatisch sichtbar. Die Sichtbarkeit kann in Virtuoso per Mausklick ge¨andert werden.
Sel	Legt fest, ob Objekte auf diesem Layer z.B. zum Kopieren angew¨ahlt werden k¨onnen
Con2ChgLy	Dieses Attribut legt fest, ob eine Änderung in diesem Layer die Ergebnisse von DRC und Bauteilextraktion beeinflusst.
DrgEnbl	besagt, dass ein Objekt in diesem Layer, w¨ahrend es verschoben wird, angezeigt wird. Bei einem **`nil`** werden nur die Umrisse des Objektes dargestellt.
Valid	Anw¨ahlbarkeit des Layers als Layer, mit dem gezeichnet wird.

5.3.3 GDS II

GDS II (Geometric Data Standard II) ist ein weit verbreitetes Format, ¨uber das verschiedene Programmpakete untereinander Layouts austauschen k¨onnen. Jedem Layer wird ein eigener Datenstrom (englisch Stream) zugewiesen. Alle Programme m¨ussen dabei dem selben Layer die selbe Datenstromnummer zuweisen. Dank dieser Datenstromorientiertheit wird GDS II oft auch kurz mit ‚Stream‘ bezeichnet. Die ben¨otigten Beziehungen werden folgendermaßen definiert:

```
layerRules(
 streamLayers(
 ;(         Layer          Strom#  Type  Zu importieren  )
  (("METAL"   "drawing")  1       0     t               )
  (("VIA"     "drawing")  2       0     t               )
  (("GATE"    "drawing")  3       0     t               )
  (("CAPDEF"  "drawing")  4       0     t               )
  (("FET"     "drawing")  42      0     t               )
 ) ;streamLayers
```

Die letzte Spalte legt fest, ob der besagte Stream ¨ubersetzt werden soll. Streams, deren Nummer nicht in dieser Sektion des Techfiles vorkommt, erzeugen Warnungen.

Um herauszufinden, welche Layernummern von einem anderem Programm verwendet werden, sei empfohlen, eine Stream-Datei ¨uber das Men¨u *File/import/stream* im CIW zu importieren. Das Stream-Konvertierungs-Programm PIPO schreibt Warnungen in seine Logdatei `PIPO.LOG`, die alle verwendeten Streamnummern enthalten. Die genauen Zuweisungen der Nummern zu Layern sind nur durch Probieren zu ermitteln.

In der speziellen Installation, an der diese Arbeit geschrieben wurde, wurden die Streameinstellungen durch Standardwerte ersetzt. Um diese zu ¨ubergehen, ist es n¨otig, eine Datei mit folgendem Inhalt zu schreiben:

```
METAL   drawing  1   0
VIA     drawing  2   0
GATE    drawing  3   0
CAPDEF  drawing  4   0
FET     drawing  42  0
```

Der Men¨upunkt *file/import/stream/User-Defined Data/Layer Map Table* des CIW liest diese Datei ein.

Die weiteren Einstellungen im verwendeten Techfile beziehen sich auf diverse Anwendungsprogramme, die beim Schreiben dieser Arbeit nicht verwendet wurden. Sie sind nur ansatzweise an die vorliegende Technologie angepasst, und wurden aus [7] zusammenkopiert.

Um Rundungsfehler bei der Konvertierung von Eckpunktkoordinaten zu vermeiden, ist es sinnvoll, bei der Stream-Konvertierung den ‚Snap to x/y-grid‘-Knopf anzuw¨ahlen. Er aktiviert einen erfahrungsgem¨aß sehr treffsicheren Rundungsalgorithmus f¨ur Koordinaten.

5.4 DRC und Extraktion

Diva [6, 24] ist ein Programmpaket, das Design Rule Checks, Layoutextraktion und Layout-Versus-Schematic-Tests erm¨oglicht.

Die ersten beiden Aufgaben unterscheiden sich im Vorgehen so wenig, dass sie urspr¨unglich die selbe Konfigurationsdatei teilten [3]. Sie werden im Folgenden n¨aher erkl¨art.

5.4.1 Geometrische Operationen

Layoutextraktion und Design Rule Checks bestehen zum gr¨oßtem Teil aus mehr oder weniger elementaren geometrischen Operationen, die mit den Layern vorgenommen werden. Im erstem Fall versucht man, pro zu erkennender Bauelementart einen eigenen Layer zu erzeugen, in dem der Umriss aller Bauelemente dieser Art markiert ist, und weitere Layer, an denen die wichtigsten Bauelementparameter einfach zu messen sind.

Im zweitem Fall generiert man Layer, die die Umrisse der fehlerhaften Stellen des Layouts enthalten, und ¨ubergibt sie dem Layouteditor. Zus¨atzlich erm¨oglicht ein Befehl namens `drc`, Messungen an Hilfslayern vorzunehmen.

Logische Operationen

Die einfachsten Operationen, die Diva mit Layern vornehmen kann, sind folgende:

(`geomNot` Layer)
(`geomAnd` Layer1 Layer2)
(`geomAndNot` Layer1 Layer2)
(`geomOr` Layer1 Layer2)
(`geomXor` Layer1 Layer2)

Die Funktionen (Bild 5.1) der einzelnen Befehle sind folgende:

`geomAnd`	(Bild 5.1 a) sucht Bereiche, in denen beide Layer gef¨ullt sind,
`geomOr`	(Bild 5.1 b) vereint alle Objekte in beliebig vielen Layern in einem Layer. Der `geomOr`-Befehl besitzt eine Spezialform, `geomCat`.
`geomNot`	erzeugt einen Layer, der ¨uberall dort gef¨ullt ist, wo der Eingangslayer leer ist, und umgekehrt.
`geomXor`	(Bild 5.1 d) erzeugt einen Layer, der genau da gef¨ullt ist, wo genau einer der Eingangslayer ein Objekt enth¨alt,
`geomCat`	erzeugt analog `geomOr` einen Layer, der alle Objekte aus den Eingangslayern enth¨alt, vereinigt sich ¨uberlappende und schneidende Polygonen aber nicht. Konsequenter w¨are die Einf¨uhrung eines `raw`-Operators (Kapitel 5.4.1) gewesen.
`geomAndNot`	(Bild 5.1 c) entfernt alle Objekte aus dem zweiten Layer vom erstem Layer, und liefert das Ergebnis dieser Operation zur¨uck.

Diva verwendet interne Layer f¨ur seine Arbeit, um nicht mit dem Layouteditor zu kollidieren. Layernamen aus dem Layout lassen sich durch das Markieren derselben mit Anf¨uhrungsstrichen verwenden. Die Layer aus dem Layout sind nicht optimiert, sondern bestehen meist aus getrennten Polygonen, die sich ber¨uhren, oder sogar ¨uberschneiden. Der im Handbuch empfohlene Weg, die Layer aus dem Layout gleichzeitig zu optimieren und f¨ur das Extraktionsprogramm zu kopieren, ist folgender:

```
HRES =(geomOr "HRES" )
GATE =(geomOr "GATE" )
METAL=(geomOr "METAL")
```

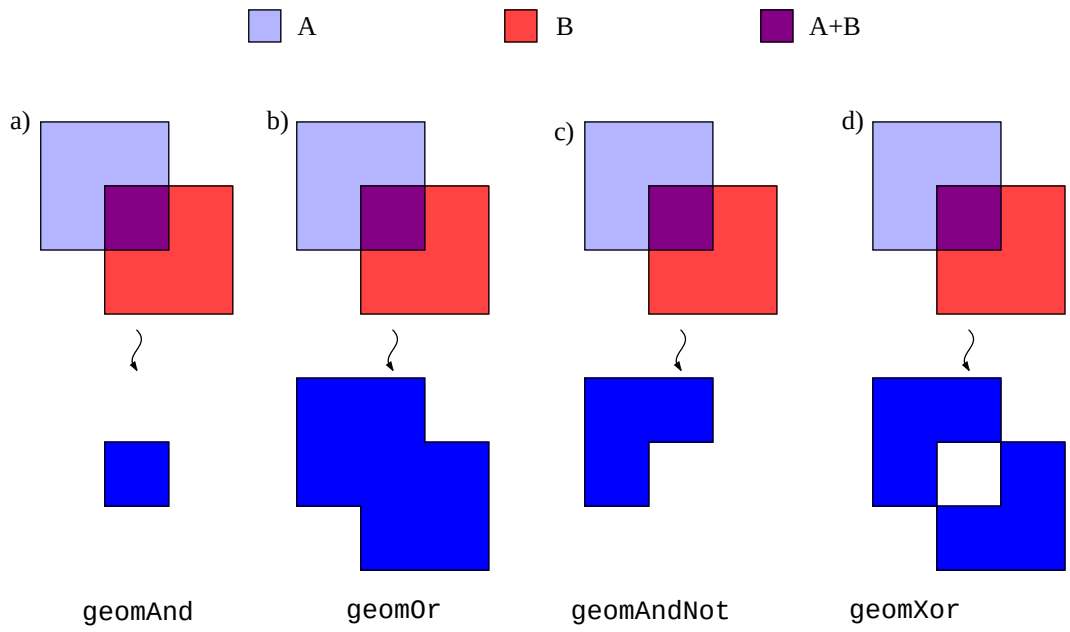


Abbildung 5.1: Logische Operationen

Ber¨uhrungsmodi von Layern

Die folgenden Funktionen sehen sich schneidende Kantenst¨ucke nicht als Ber¨uhrung an.

`(geomButting` Layer1 Layer2) findet alle Objekte in Layer 2, von denen mindestens eine Kante Objekte aus Layer 1 von außen ber¨uhrt.

`(geomButtOnly` Layer1 Layer2) sucht Objekte, die sich ber¨uhren, aber nicht schneiden.

`(geomCoincident` Layer1 Layer2) findet alle Objekte in Layer 1, die solche in Layer 2 von innen ber¨uhren.

`(geomCoinOnly` Layer1 Layer2) liefert alle Objekte aus Layer 1 zur¨uck, die Objekte aus Layer 2 von innen ber¨uhren, aber nicht ¨uberlappen.

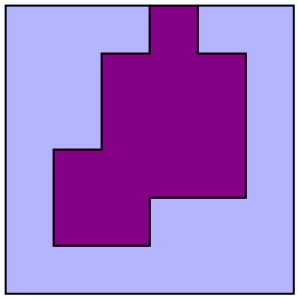
`(geomButtOrCoin` Layer1 Layer2) Sucht Objekte, die Objekte aus Layer 2 aus einer beliebigen Richtung ber¨uhren.

Sich ¨uberlappende Layer

(`geomOverlap` Layer1 Layer2) sucht Objekte aus Layer 1, die solche in Layer 2 schneiden.

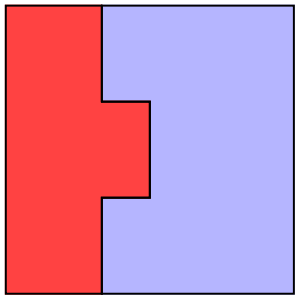
`(geomEnclose` Layer1 Layer2) sucht Objekte in Layer 1, die solche in Layer 2 ganz umschließen. Objekte d¨urfen den Rand des sie umschließenden Objekts ber¨uhren.

a)



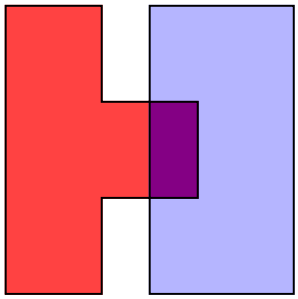
B ber¨uhrt A von Innen

b)



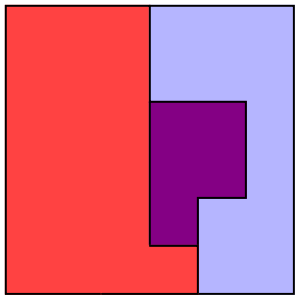
Sich von außen ber¨uhrende, jedoch nicht ¨uberschneidende Layer

c)



Sich ¨uberlappende, aber nicht ber¨uhrende Layer

d)



Sich ¨uberlappende und ber¨uhrende Layer

Abbildung 5.2: Ber¨uhrungsmodi

`(geomOutside` Layer1 Layer2) Die Bedingung von `geomOutside` sucht alle Objekte in Layer 1, die keine Objekte in Layer in Layer 2 ¨uberlappen.

`(geomAvoiding` Layer1 Layer2) sucht Objekte in Layer 1, die nichts in Layer 2 ber¨uhren oder ¨uberlappen

`(geomStraddle` Layer1 Layer2) sucht Objekte in Layer 1, die Objekte im zweiten Layer ¨uberkreuzen, dass heißt von mindestens zwei Seiten aus schneiden.

(`geomEnclose` Layer1 Layer2) sucht Objekte aus Layer 1, die solche in Layer 2 vollst¨andig in sich einschließen. Um zu verhindern, dass Objekte, die den Rand des sie einschließenden Objektes ber¨uhren, als eingeschlossen gez¨ahlt werden, kann man die nachfolgende Sonderform dieses Befehls verwenden:

(`geomEnclose` Layer1 Layer2 `exclusive`)

Der `exclusive`-Operator ist bei allen geometrischen Operationen m¨oglich, bei denen er sinnvoll definierbar ist. Er bestimmt, dass nur Objekte erkannt werden sollen, die keine anderen Bedingungen aus der jeweiligen Klasse der Bedingungen (hier Bild 5.2) erf¨ullt.

Die weiteren Operatoren f¨ur geometrische Gesetzm¨aßigkeiten sind folgende:

`contiguous`	gibt bei L¨angenmessungen an, dass nicht nur eine einzelne Kante, sondern immer der gesamte Kantenzug vermessen werden soll. Bei Polygonen mit L¨ochern wird der Umfang der Hohlr¨aume mitgez¨ahlt.
`fig`	bestimmt, dass Funktionen, die einzelne Kanten als Ergebnis haben, das ganze Objekt, zu dem die erkannte Kante geh¨ort, zur¨uckliefen sollen, statt dieser allein.
`diffNet`	bestimmt, dass nur Elemente, bei denen beide Partner nicht elektrisch leitend verbunden sind, gesucht werden.
`sameNet`	sucht nur Elemente, die elektrisch leitend verbunden sind.
`raw`	besagt, dass Objekte, die sich ber¨uhren, oder ¨uberschneiden, nicht automatisch zu Polygonen zusammengefasst werden sollen.
`keep`	sagt, dass nur Objekte gesucht werden sollen, die die gew¨unschte Bedingung n-mal erf¨ullen: `3<=keep<5` — sucht Objekte, die die mindestens 3-mal erf¨ullen, aber weniger als 4- mal. `keep=5` — solche, die dies genau 5-mal tun.
`ingore`	Das Gegenteil von `keep`: Ignore ignoriert Objekte, die die gew¨unschten Bedingungen n-mal erf¨ullen.

geomSize

Eines der wichtigsten (wenn auch langsamsten) Features von Diva ist die M¨oglichkeit Objekte zu Expandieren (Englisch: (to) resize). Hierbei werden alle Kanten eines Objekts um den selben Betrag verschoben. Der `geomSize`-Befehl hat zwei Argumente: Den zu behandelnden Layer und den Betrag, um den der Layer expandiert werden soll. Ein negativer Betrag f¨uhrt zu einer Kontraktion des Layers. Sich nach einer Expansion ber¨uhrende oder ¨uberlappende Objekte werden vereint. Bei der Expansion werden L¨ucken bis zum doppelten Betrag der Weite, um die expandiert wird, gef¨ullt.

Ein Beispiel f¨ur das Expandieren ist auf Seite 47 zu finden.

geomStretch

`geomStretch` expandiert wie `geomSize` einen Layer. Sich ber¨uhrende oder schneidende Objekte werden von diesem Befehl jedoch nicht vereint.

geomGetLength

Die L¨angenmessfunktion sucht Kanten, deren L¨angen sich in einem bestimmtem Bereich bewegen. Die Notation ist ¨ahnlich wie bei den `keep`- und `ignore`-Befehlen:

`(geomGetLength` Layer a`<[=]length<[=]b [contiguous] [fig])`

Weitere Befehle

(`geomBkgnd`)	erzeugt einen Layer, der aus einer geschlossenen Fl¨ache besteht, die mindestens so groß ist, wie das Layout (plus einen kleinen Sicherheitsabstand).
(`geomEmpty`)	erzeugt einen leeren Layer.
(`geomGetPurpose` ”Layer” ”Purpose”)	Die in Kapitel 5.4.1 beschriebenen Methode, Layer ¨uber den `geomOr`-Befehl nach Diva zu ¨ubernahmen, ¨ubernimmt ausschließlich Objekte mit dem Verwendungszweck „Drawing“. Objekte mit anderen Verwendungszwecken ¨ubernimmt der Befehl `geomGetPurpose`.
(`geomGetRectangle` Layer)	sucht Rechtecke,
(`geomGetPolygon` Layer)	sucht alle Objekte, die keine Rechtecke sind.
(`geomGetNon90` Layer)	findet alle Objekte, die Kanten haben, die nicht in X-oder Y-Richtung verlaufen.
(`geomGetNon45` Layer)	sucht alle Objekte, die Kanten, die nicht achsenparallel oder im 45°- Winkel zu den Achsen sind, besitzen.
(`geomGetHoled` Layer)	findet Objekte mit L¨ochern
(`geomGetNoHoles` Layer)	ist die dazu komplement¨are Funktion. Sie sucht Objekte ohne Hohlr¨aume.

5.4.2 Design Rule checks

Diva stellt einige Befehle zur Verf¨ugung, die nur f¨ur Entwurfsregel¨uberpr¨ufungen verwendet werden k¨onnen [6]. Eine Verwendung dieser Befehle in Konfigurationsdateien f¨ur andere Aufgaben f¨uhrt zu einer Fehlermeldung. Drei von diesen Befehlen wurden im Zuge dieser Arbeit verwendet:

`saveDerived`

Um die detektierten Regelverst¨oße anzeigen zu k¨onnen, stellt Diva den `saveDerived`-Befehl zur Verf¨ugung, der einen Layer mit einer Fehlermeldung versieht und an den Layouteditor weitergibt.

```
(saveDerived (geomAnd METAL HRES) "METAL overlaps HRES")
```

markiert zum Beispiel alle Stellen, an denen Metall ¨uber L¨ocher (HRES) gelegt wurde, als Fehler.

Der `saveDerived`-Befehl kann auch bei der Bauteilextraktion verwendet werden, um dabei erkannte Fehler zu markieren.

`drc`

Der drc-Befehl ist ein sehr m¨achtiger Befehl, der komplexe Messungen vornehmen kann. Falls der `drc`-Befehl als letztes Argument einen Fehlermeldungstext erh¨alt, wird das Ergebnis direkt als Fehler markiert.

Aufgrund der Komplexit¨at dieses Befehls sei hier auf die Online- Dokumentation verwiesen [4].

Die folgenden Beispiele f¨ur den drc- Befehl markieren direkt Entwurfsregelverst¨oße:

```
(drc VIA (width < gunViaMinWidth) "Via too small/narrow")
(drc VIA METAL (enc < gunViaMetalMinOverlap)
 "Via/ Metal-Overlap too low.")
```

Hilfslayer, die die Umrisse der beanstandeten Regionen erhalten, werden folgendermaßen generiert:

```
templayer=(drc METAL (sep <gunMetalSep))
```

`dubiousData`

Findet Polygonen, deren Definition formale Fehler enth¨alt. Diese k¨onnen zum Beispiel beim Einlesen von Layouts entstehen, die von externen Programmen generiert wurden.

Ein Beispiel f¨ur solche Objekte sind Polygone der Breite 0. Des Weiteren sucht dieser Befehl Polygone, die Hohlr¨aume enthalten, die die ¨außeren Grenzen des Polygons schneiden. [26, 6]

5.4.3 `divaDRC.rul`

Alle Design Rule Checks werden in der Datei `divaDRC.rul` definiert.

Die ganze Datei wird pro Extraktionsvorgang einmal eingelesen und interpretiert. Alle Befehle, die sich auf Layer beziehen, werden in eine Liste eingetragen. Operationen, deren Ergebnis nicht verwendet wird, werden aus dieser Liste entfernt, was im CIW mitprotokolliert wird. Die Generierung tempor¨arer Layer verursacht keinen merklich erh¨ohten Speicherverbrauch von Cadence, wenn auf diese Layer nur einmal, direkt nach deren Generierung, zugegriffen wird. Werden diese aber am Dateiende ein zweites Mal ben¨otigt, steigt der Speicherverbrauch massiv. Diva entfernt also Layer aus dem Speicher, sobald diese nicht mehr ben¨otigt werden.

Dieses Verfahren optimiert den Zeit- und Ressourcenaufwand. Der Nachteil ist, dass man keinen SKILL-Code (z.B. Aktualisierung eines Fortschrittsanzeigers) zwischen der wirklichen Ausf¨uhrung von zwei Layeroperationen ausf¨uhren lassen kann.

W¨ahrend DRC- oder Extraktionsoperationen sind weder das CIW noch der Virtuoso-Editor verwendbar. Einen DRC abzubrechen ist allerdings jederzeit ¨uber die Tastenkombination *STRG+C* m¨oglich.

Die Definition der DRCs [23] beginnt mit folgenden Zeilen:

```
(drcExtractRules
 (ivIf
  (switch "drc?") then
```

Die erste Zeile beginnt die DRC- oder Extraktionsregeln.

Der ivIf-Befehl ähnelt dem if-Befehl, und wird verwendet, um zu testen, welche Schalter man im Extraktions-Fenster (Bild 3.8) gesetzt hat.

Schalter, nach denen man testet, sind automatisch dort anwählbar; der „drc?“-Schalter ist bei Design Rule Checks automatisch gesetzt, und wird nur aus historischen Gründen abgefragt. Ursprünglich wurden Design Rule Checks und Extraktion, wie schon erwähnt, von derselben Datei aus gesteuert [3].

Auslesen der Parameter aus dem Techfile

Um den Rest der Datei übersichtlicher zu machen, wird das Auslesen der Parameter aus dem Techfile am Anfang der Datei `divaDRC.rul` erledigt. Die Ergebnisse der Anfragen werden in Variablen mit dem Präfix ‚`gun`‘ abgelegt.

Auf eine aufwändige Überprüfung, ob im Techfile wirklich alle benötigten Parameter definiert sind, wurde verzichtet: Anfragen auf nichtexistente Parameter liefern automatisch den Wert `nil` zurück. Dieser Wert führt, da keine Zahl, bei jedem Versuch, ihn zu verwenden, automatisch zu einer Fehlermeldung.

Das Techfile wird programmintern über eine Nummer repräsentiert. Um auf ein Techfile zugreifen zu können, ist es nötig, dessen Nummer zu ermitteln. Die dafür nötigen Befehle sind in [25] dokumentiert. Der Zugriff auf diese Nummer geschieht folgendermaßen:

```
(setq guntechfile techGetTechFile(ddGetObj("ptest")))
```

In diesem Fall wird das Techfile gesucht, das für die Bibliothek ‚`ptest`‘ zuständig ist. Falls der Name der Bibliothek geändert wird, ist es nötig, diesen in die Konfigurationsdateien für DRC und Extraktion einzutragen.

Benannte Parameter (Tabelle 5.1) werden folgendermaßen ausgelesen:

```
(setq gunMetalInsideFetSep (techGetParam guntechfile "gunMetalSep"))
```

Ein anderer Befehl liest Entwurfsregeln (Tabelle 5.2) aus dem Techfile aus:

```
(setq gunMetalSep (techGetSpacingRule guntechfile "minSpacing" "METAL"))
```

Vorbehandlung der Eingangslayer

Diva arbeitet ausschließlich mit Hilfslayern. Die nötigen Kopien von Layern aus dem Layout, die von Diva verwendet werden kann, erzeugt folgendes Codefragment:

```
(bkgnd  = (geomBkgnd ))
(bkgnd  = (geomOr bkgnd))
(METAL  = (geomOr "METAL"))
(GATE   = (geomOr "GATE"))
(VIA    = (geomOr "VIA"))
(HRES   = (geomOr "HRES"))
(SEMI   = (geomNot "HRES"))
(CAPDEF = (geomOr "CAPDEF"))
(FET    = (geomOr "FET"))
```

Von außen importierte Layouts enthalten oft, wahrscheinlich durch Rundungsfehler bei den Eckpunkten der Polygone bedingt, kleine Lücken, die empfindlich stören können. Ihre Größe und Position ist rechnerabhängig. Nach einigen Versuchen wurde ein Schalter „killstreamgaps“ eingeführt, der diese schließt. Er ist im DRC-Fenster (Bild 3.9) anwählbar.

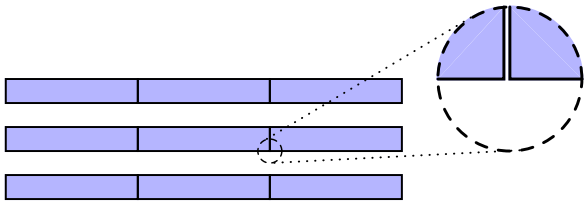


Abbildung 5.3: Leiterbahn mit stream-typischen Konvertierungsfehlern

Die Layer PAD, VIA und FET enthalten keine aus mehreren Polygonen zusammengesetzten Objekte. Konvertierungsfehler erzeugen hier keine Lücken, die vor dem DRC zu schließen sind. Die Lücken schließt folgende Routine:

```
(ivIf
 (switch "killstreamgaps") then
 (METAL = (geomSize (geomSize METAL .1) -.1))
 (GATE  = (geomSize (geomSize GATE  .1) -.1))
 (CAPDEF= (geomSize (geomSize CAPDEF .1) -.1))
 (HRES  = (geomSize (geomSize HRES   .1) -.1))
)
```

Solange das zweite Argument der `geomSize`- Befehle kleiner als das kleinste im Layout vorkommende Detail ist, entstehen hierdurch außer einem kleinem Zeitverlust keine Nachteile.

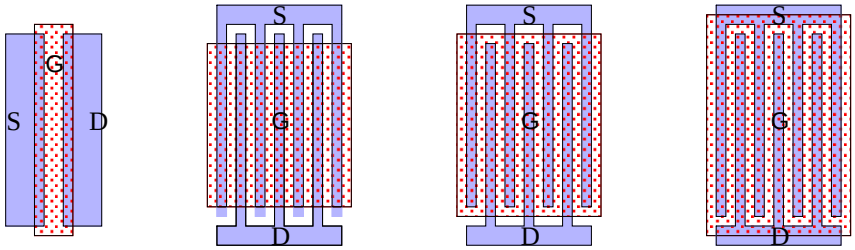
Neben der Verwendung dieses Schalters ist es empfohlen, beim Dateiimport den ‚Snap X/Y to grid‘-Schalter im Optionenmenu des Stream-to-Cadence-Konverters zu aktivieren, der sicher stellt, dass alle Eckpunkte im Layout auf Punkten des im Techfile definierten Gitters liegen.

Der Mechanismus des obigen Schalters ist nicht dokumentiert, vermeidet aber in allen getätigten Tests, dass Bauteilparameter durch Rundungsfehler verfälscht werden.

Interessant ist, dass in seltenen Fällen eng beieinanderliegende streng in x-oder y-Richtung verlaufende Kanten bei nichtverwendung obigen Schalters Schalter leicht nach links gekippt werden. Die Eckpunkte von achsen-

parallelen Kanten haben zwei identische x-bzw. y- Koordinaten und sollten daher eigentlich mit identischen Rundungsfehlern behaftet sein.

Automatische FET-Detektion



a) 1-Kanal-FET b) Gefalteter FET c) Alternative Faltung d) Schlangenf¨ormiger Kanal

Abbildung 5.4: FET-Typen

Die FET-Detektion ist aufgrund des Aufbaus der Transistoren (Bild 5.4) aufw¨andiger als in der CMOS-Technik. Dennoch war es m¨oglich, einen Algorithmus zu entwickeln, der im Layout eingezeichnete Feldeffekttransistoren zuverl¨assig detektiert.

Die Einschr¨ankungen der implementierten Autodetektion sind folgende:

- Bei FETs, die ein gemeinsames Gate teilen (Abbildung 5.5), muss die Leitung, die beide Transistoren trennt (im Bild mit ‚d' bezeichnet), mindestens die frei einstellbare Breite `gunSDMaxWidth` haben.

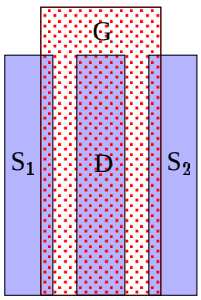


Abbildung 5.5: FETs mit gemeinsamem Gate

- Die Stellen, an denen der Kanal aus Einkanaltransistoren austritt, m¨ussen zu Punkten, an denen sich GATE- und METAL-Layer ¨uberschneiden, einen gewissen Mindestabstand einhalten. Dieser Minimalabstand ist entweder kleiner oder gleich der doppelten Kanall¨ange (siehe unten) oder der maximalen zu detektierenden Kanall¨ange, je nachdem, welcher von beiden Werten kleiner ist.
- Leitungen innerhalb von Transistoren d¨urfen diese Breite (`gunSDMaxWidth`) nicht ¨ubersteigen, da diese sonst auch getrennt werden. Das LVS wird von dieser Trennung nicht beeinflusst, da es parallelgeschaltete Transistoren von gleicher Gr¨oße automatisch vereint.
- Kreuzungen zwischen GATE- und mehreren METAL-Leitungen bilden parasit¨are Transistoren. Falls die Abst¨ande der METAL-Leitungen voneinander kleiner als die gr¨oßte erlaubte Kanall¨ange ist, und die GATE-Leitung sehr breit sind, ist der Transistor stark genug, um ihn als g¨ultig zu erachten.

- Der Einsatz sehr kleiner Transistoren mit nur einem Kanal ist unwahrscheinlich. Sollte dies trotzdem gew¨unscht sein, ist es erforderlich, den Kanal im FET-Layer zu markieren.

Praktisch sollten diese Einschr¨ankungen jedoch ohne Belang sein:

Der Einsatz der ersten Konstruktion ist unwahrscheinlich, da nur f¨ur die seltenen Transistoren mit nur einem Kanal geeignet, und nur im Full-Custom-Design denkbar.

Einkanal-FETs liefern nur geringe Str¨ome. Um dies zu kompensieren werden deren Kanall¨angen m¨oglichst gering gehalten. Es ist relativ unwahrscheinlich, dass die Abst¨ande zu anderen Objekten in dieser Gr¨oßenordnung sein werden.

Die Leitungen innerhalb von Transistoren mit mehreren Kan¨alen sind in der Regel viel d¨unner als Leitungen, die f¨ur die Verdrahtung verwendet werden. Dies verringert die durch das Halbleitermaterial fließenden Leckstr¨ome und den Platzbedarf. Dank der Parallelschaltung vieler Source- und Drain-Leitungen ist die Gesamtmetallfl¨ache trotzdem betr¨achtlich und der Leitungswiderstand gering.

Transistoren mit schlangenf¨ormigen Gates (Bild 5.4 d) haben wegen Kanteneffekten eine Reihe von Nachteilen gegen¨uber den ‚normalen‘ gefalteten Transistoren. Sie werden in der vorliegenden Technologie nicht eingesetzt. Da nichts gegen deren Funktionsf¨ahigkeit spricht, wurde vorgesehen, auch diese Art von Transistoren zu detektieren. Um m¨oglichst viele Fehlertypen abzudecken, werden sie jedoch vom DRC als Fehler markiert. Dadurch, dass die Kanalenden an den Seiten aus dem Transistor austreten, ist die Detektion dieses Transistortyps schwieriger und hat folgende zus¨atzliche Einschr¨ankungen:

- Transistoren mit schlangenf¨ormigen Kan¨alen haben nur einen Kanal aber eine h¨ohere Wahrscheinlichkeit f¨ur eine große Kanalweite. Die Abst¨ande zu unter Gatematerial gelegenen Metallobjekten k¨onnten also ein gr¨oßeres Problem f¨ur die Autodetektion werden.

- Falls der Abstand zwischen dem Ende der Finger und der gegen¨uberliegenden Sammelleitung ungleich der Kanalweite ist, wird der Kanal unter Umst¨anden nur teilweise erkannt.

  Eine Technologie mit Schwankungen der Kanall¨ange innerhalb eines Transistors ist dem Autor allerdings nicht bekannt.

Transistoren entstehen in der vorliegenden Technologie an Stellen, an denen der Freiraum zwischen zwei Metallbahnen von Gate-Material ¨uberbr¨uckt wird. Metall und Gatematerial m¨ussen sich ¨uberlappen, um eine zuverl¨assige Funktion des Transistors zu gew¨ahrleisten.

Um auch in F¨allen, in denen die obigen Bedingungen nicht gegeben sind, eine fehlerfreie Funktion des Programmes zu gew¨ahrleisten, wurde alternativ eine manuelle Markierung von Transistoren implementiert.

Beide Mechanismen k¨onnen innerhalb eines Layouts koexistieren. So ist es m¨oglich, einen sehr kleinen Transistor als nicht parasit¨ar zu markieren. Eine manuelle Markierung deaktiviert die Autodetektion f¨ur den jeweiligen Transistor.

Das Problem der FET-Detektion l¨asst sich in zwei Teilaufgaben zerlegen: Detektieren der aktiven Regionen, und das Zusammenfassen dieser zu ganzen Transistoren.

Aktive Regionen k¨onnen sich nur unter einer GATE-Fl¨ache befinden. Die erste Herangehensweise war, dort im METAL-Layer alle L¨ucken bis zur maximalen Kanall¨ange zu suchen, und diese als Kanal zu bezeichnen. Dies l¨asst sich folgendermaßen realisieren (Bild 5.6):

Zuerst werden alle METAL-Objekte, die unter GATE-Fl¨achen gelegen sind (Bild 5.6 b), um die halbe maximale zu detektierende Kanall¨ange expandiert (Bild 5.6 c; Halbe Kanall¨ange, da das Metall zu beiden Seiten des Kanals expandiert wird). Alle L¨ucken, die klein genug sind, um einen Kanal bilden zu k¨onnen, werden durch diesen Schritt gef¨ullt. Ein nachfolgendes Kontraktieren (Bild 5.6 d) f¨uhrt die Transistoren auf ihre urspr¨unglichen Umrisse zur¨uck. Die nun gef¨ullten Kan¨ale bleiben ¨ubrig, wenn vom Hilfslayer, der aus den Expansionen resultiert, die urspr¨unglichen METAL-Fl¨achen abgezogen werden (Bild 5.6 e):

```
(metalundergate= (geomAnd METAL GATE))

(active= (
 geomAndNot (
  geomSize (
   geomSize metalundergate 0.5*gunChannelMaxLength)
   -.5*gunChannelMaxLength )
  metalundergate)
)
```

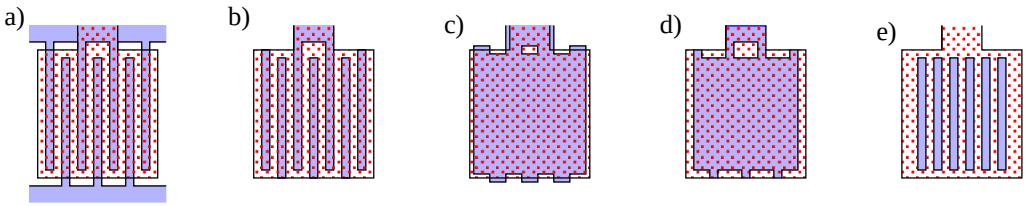


Abbildung 5.6: Ein-Schritt-Kanaldetektion: a) Grundzustand, b) METAL unter GATE, c) Expandieren, d) Kontraktieren, e) Die Kan¨ale

Die L¨ucken zwischen den einzelnen Fingern des Source- oder Drainanschlusses eines Transistors sind oft klein genug, um als Kan¨ale detektiert werden zu k¨onnen. Wenn diese Gebiete teilweise von GATE-Material ¨uberlappt werden, werden diese unter Umst¨anden ebenso als Kan¨ale detektiert. Bei dem Transistortyp aus (Bild 5.4 b) mit Kurzen Kan¨alen ist dies der Fall. In Bild 5.7 wird dies genauer dargestellt.

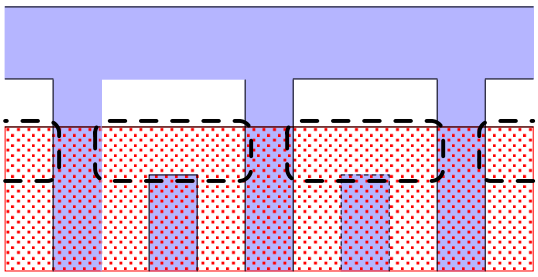


Abbildung 5.7: FET in Großansicht

Die ersten Versuche, diese Anh¨angsel an kurze Kan¨ale herauszufiltern, waren zum Scheitern verurteilt:

Das gefundene deformierte aktive Gebiet ist an einem St¨uck. Herausfiltern von einzelnen Objekten mit den Überlappungs- und Ber¨uhrungstestbefehlen scheidet also von vornherein aus.

‚Echte' Kan¨ale k¨onnen nur zwischen METAL-Objekten mit Netzknotennummern (Kapitel 3.4) existieren. Diva erlaubt die Überpr¨ufung, ob zwei Objekte dieselbe Netzknotennummer teilen. Diese Funktion l¨asst sich aber nicht mit der zur Kanalsuche ben¨otigten L¨uckendetektion kombinieren.

Alle Metallfinger enden auf derselben H¨ohe. Es ist naheliegend, diese Enden suchen zu lassen, einen Streifen hindurchzulegen, und alles Kanalteile außerhalb dieses Streifens wegzuschneiden. Befehle, die die Enden der aktiven Gebiete finden, existieren. Sie liefern allerdings nur die Kanten, keine geschlossenen Polygone zur¨uck, und diese lassen sich nicht f¨ur die gestellte Aufgabe verwenden.

Da alle weniger CPU-intensiven Methoden verworfen wurden, wird auf ein schrittweises Expandieren ausgewichen:

Zuerst werden alle Kan¨ale gesucht, die klein genug sind, dass die Gebiete zwischen den einzelnen Fingern eines Anschlusses sicher nicht mitdetektiert werden.

Ein weiterer nach obiger Methode erzeugter Hilfslayer enth¨alt alle Kan¨ale bis zur doppelten L¨ange.

Im zweitem Hilfslayer sind unter Umst¨anden an die schon im ersten Schritt detektierten Kan¨ale die Zwischenr¨aume zwischen den einzelnen Source/Drainfingern angeh¨angt. Kan¨ale, die keine im vorhergehenden Schritt detektierten aktiven Gebiete ¨uberlappen, werden aber sicher korrekt detektiert, da die Abst¨ande der einzelnen Finger eines Anschlusses zueinander gr¨oßer als die doppelte Kanall¨ange ist.

Das Verfahren funktioniert f¨ur kleine Layouts mit wenigen Transistoren. Der Zeitverbrauch wird vom Autor jedoch als nicht vertretbar angesehen. Gr¨oßere Layouts mit mehreren hundert Transistoren f¨uhren zu einer „Segment Violation“ mit Programmabbruch. Dokumentiert ist dieses Verhalten von Cadence nicht, Versuche lassen aber vermuten, dass dieser Fehler auf eine zu hohe Zahl von verwalteten Eckpunkten von Layern zur¨uckzuf¨uhren ist.

Außerdem k¨onnen Transistoren nach der Expansion mit ihren n¨achsten Nachbarn verschmelzen: Die L¨uckendetektion, die die aktiven Gebiete innerhalb von Transistoren detektieren soll, detektiert auch schmale L¨ucken zwischen Transistoren. (Bild 5.8).

Jeden Transistor individuell zu expandieren vermeidet dieses Problem, ist aber mangels geeigneten Befehlen nicht m¨oglich. Der Versuch, nach jedem Vergr¨oßern der Metallfl¨achen alle Bereiche außerhalb der Gates abzuschneiden, verhindert das Ineinanderfließen der Transistoren, das Kontraktieren f¨uhrt aber nicht mehr auf die urspr¨unglichen Umrisse des Transistors zur¨uck.

Das in der endg¨ultigen Implementierung verwendete Verfahren ist folgendes:

Anstatt jedesmal die gesamten Metallbereiche zu expandieren, die unter GATE-Objekten gelegen sind, ist es einfacher, die schon expandierten Layer vom letzten Schritt wiederzuverwenden (Bild 5.9). Da in jedem Schritt ein großer Teil der Hohlr¨aume, die dieser Hilfslayer enth¨alt, gef¨ullt werden, nimmt jeder Schritt nun weniger Ressourcen in Beschlag, wie der letzte. Die fehlerhaft als Kan¨ale detektierten Gebiete (Bild 5.8) ¨uberlappen diese nicht mehr, sondern ber¨uhren sie von außen. Der Testbefehl f¨ur Ber¨uhrungen war in im Rahmen dieser Arbeit gemachten Versuchen geringf¨ugig schneller als der Befehl f¨ur Überlappungen. Schutzverletzungen wurden mit dieser Methode nicht mehr beobachtet.

Die Implementierung im Detail:

```
(metalundergate= (geomAndNot METAL (geomNot GATE)))

(active1=(geomAndNot metalundergate FET))
```

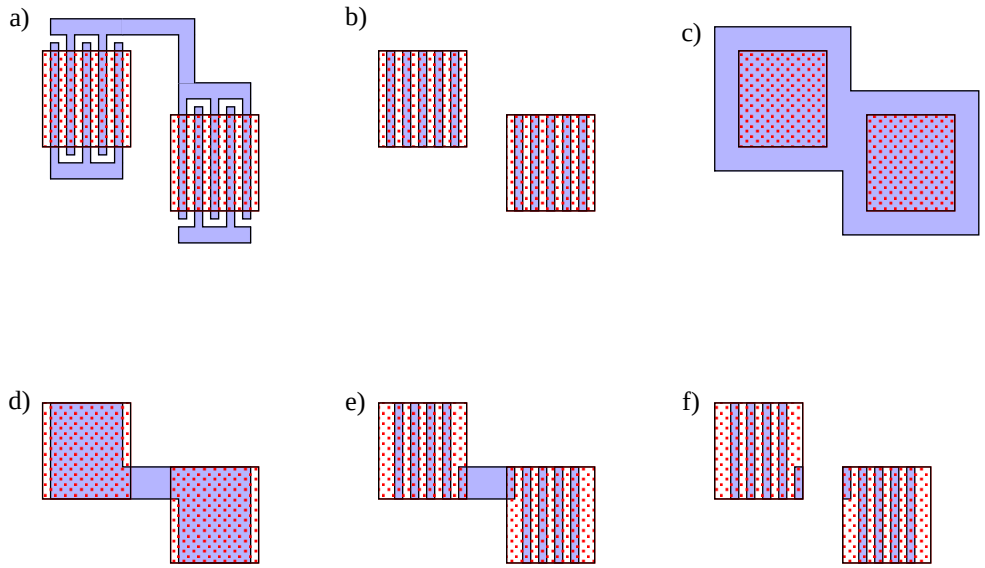


Abbildung 5.8: Verschmelzen von Transistoren bei der Kanaldetektion: a) Grundzustand, b) METAL unter GATE, c) Expandieren, d) Kontraktieren, e) detektierte Kan¨ale, f) detektierte Kan¨ale unter GATE

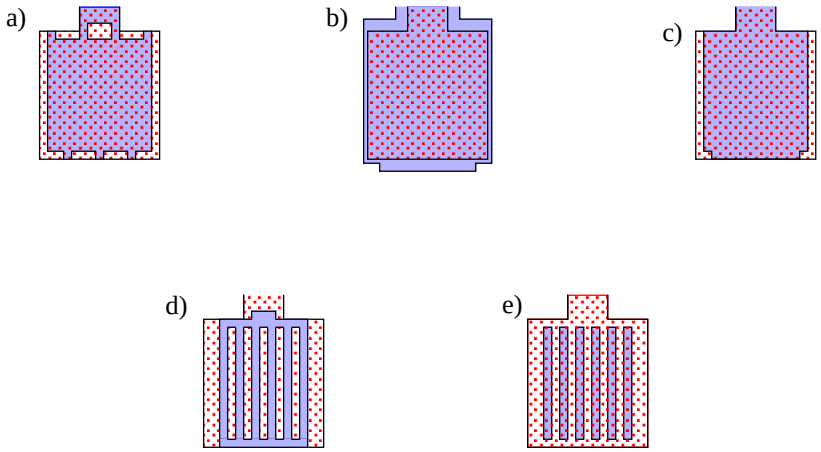


Abbildung 5.9: Kanaldetektion, weiterer Schritt: a) Hilfslayer aus Bild 5.6, b) Expandieren, c) Kontraktieren, d) Anh¨angsel, die die Kan¨ale ber¨uhren, e) detektierte Kan¨ale

Der Layer `active1` enth¨alt nun alle nicht vom Benutzer als FET markierten Metallteile, die unter GATE-Objekten liegen.

In einer Schleife werden alle Bereiche bis zur Gr¨oße $2k \cdot$ `gunDetectMags` sukzessive aufgef¨ullt ($k = \frac{i}{2^{\text{gunDetectMags}-1}}$). Zus¨atzlicher SKILL-Code ermittelt `gunDetectMags` automatisch aus der minimalen erlaubten und der maximalen zu detektierenden Kanalweite.

```
(for i 1 gunDetectMags
```

```
 (k=1.0/(2**(1+gunDetectMags-i)))
 (active2= ( geomSize
  (geomSize active1 k*gunDetectMaxWidth)
  -k*gunDetectMaxWidth ))
```

`active2` enthält jetzt das Metall mit gefüllten neu erkannten Kanälen. Diese sind also

```
 (active1=(geomAndNot active2 active1))
```

Es lohnt sich nicht zu testen, ob dies der erste Durchgang ist, und somit alle Elemente in active1 automatisch gültig sind: der folgende Test auf Gültigkeit der Kanäle verbraucht, wenn der Layer `active` noch leer ist, fast keinerlei Zeit.

```
  (active=(geomOr active
   (geomAndNot active1
    (geomButting active1 active)))
  )
```

Vor dem nächsten Schritt muss der Startlayer neu initialisiert werden:

```
   (active1=active2)
)
```

Diese Zuweisung benötigt nur wenig Zeit, da nur die Speicheradresse, auf die `active1` zeigt, verändert wird.

Kanäle, die durch das Verschmelzen von Transistoren bei der Kanaldetektion entstehen (Bild 5.8), berühren nur ein einzelnes METAL-Objekt. Sie werden mit folgenden Zeilen entfernt:

```
(active=(geomAnd active GATE))
(active=(geomButting active METAL keep>1))
```

Nun benötigt das DRC-Programm nur noch die Umrisse des gesamten Transistors:

```
(fet = (geomSize
 (geomSize active .5*gunSDMaxWidth)
 -.5*gunSDMaxWidth))
```

Kanäle, die kürzer sind als minimal erlaubt, werden immer als Fehler erkannt und markiert. Ihre Form kann allerdings fehlerhaft detektiert werden, wenn die Lücken zwischen den einzelnen Fingern des Source- bzw. Drainanschlusses kleiner als die minimale erlaubte Kanalweite ist.

Verdrahtungsleitungen sind immer breiter als `gunSDMaxWidth`. Leitungskreuzungen, die Transistoren ergeben (siehe Bild 5.10), werden daher als Transistoren mit nur einem Kanal detektiert.

Transistoren mit einem einzelnen Kanal enthalten keine METAL-Objekte innerhalb ihres Definitionsgebietes. Sie werden mit folgenden Zeilen detektiert:

```
(singlechannelfet=(geomAndNot fet
 (geomOverlap fet METAL)))
```

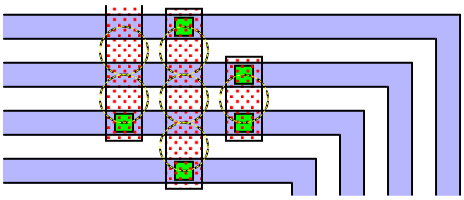


Abbildung 5.10: Parasit¨are FETs

Sie sind funktionsf¨ahige Bauteile, w¨urden aber, wenn als solche detektiert, einen Vergleich zwischen Layout und Schaltplan unm¨oglich machen. Unter allen Kriterien, die Cadence anbietet, um parasit¨are von gewollten Transistoren zu unterscheiden, hat sich der Umfang des Transistors am besten bew¨ahrt. Ein getrennter Test auf Kanall¨ange und -weite ist nicht m¨oglich. Da die Kanall¨ange ¨uber die minimale Leitungsbreite definiert ist, ist es allerdings m¨oglich, alle Transistoren mit einer hinreichend kleinen Kanall¨ange als gewollt zu detektieren. Diese Minimall¨ange (`gunParfetMinLength`) entspricht entweder der minimalen Breite einer GATE-Leitung oder dem minimalen Abstand von Metallleitungen zueinander, je nachdem, welcher Wert kleiner ist.

```
(parfet=(geomGetLength singlechannelfet
 length<=gunMaxCrossingPerimeter contiguous fig))
(parfet=(geomAndNot parfet (geomGetLength parfet
 length<=gunParfetMinWidth fig)))
(fet=(geomAndNot fet parfet))
```

Der `parfet`-Hilfslayer enth¨alt die FETs, die bei der Extraktion als parasit¨are Bauteile (Seite 26) erkannt werden.

Falls doch einmal ein sehr kleiner Transistor absichtlich eingezeichnet wird, ist es immer noch m¨oglich, dessen Umriss von Hand im Layer FET zu definieren:

```
 (FET = (geomOr FET fet))
)
```

Vor diesem Schritt werden alle automatisch detektierten Transistoren, die von Hand markierte Transistoren ¨uberlappen, aus den Layern ‚fet‘ und ‚parfet‘ gel¨oscht. Dies erm¨oglicht dem Benutzer eine vollst¨andige Kontrolle ¨uber die Umrisse von Hand markierter FETs.

Die Implementierung der eigentlichen DRCs

Die eigentlichen DRCs machen nur einen verschwindend geringen Teil der Definitionen des Programms aus. Dies liegt daran, dass obiger Code diese Aufgabe in eine standardisierte Form bringt. Die Fehlermeldungen, die die Entwurfsregeltests liefern k¨onnen, sind in englischer Sprache geschrieben und mit Fehlercodes versehen. Diese Fehlercodes bestehen aus zwei Zeichen.

Im Quellcode der vorliegenden Implementierung sind die einzelnen Tests nach Layern, nicht nach Typ geordnet.

5.4.4 Extraktion

Die Layoutextraktion ¨ahnelt in großen Teilen dem DRC, so dass beide Programme die selben Routinen zur Generierung der ben¨otigten Hilfslayer teilen.

Der Beginn der Datei `divaEXT.rul` entspricht deshalb auch dem von `divaDRC.rul` aus dem letzten Kapitel [23].

Vorbereitung der Extraktion

```
(drcExtractRules
 (ivIf
  (switch "extract?") then
```

Hauptunterschied (und -Einschr¨ankung) ist jedoch bei dieser Aufgabe das komplette Fehlen des `drc`-Befehls, der m¨achtige Messfunktionen zur Verf¨ugung stellt.

Die Autodetektionsfunktionen wurden von `divaDRC.rul` (Kapitel 5.4.3) ¨ubernommen.

Leitungen

Der erste Schritt der Leitungsextraktion ist das Definieren der Querverbindungen zwischen den einzelnen Layern [21]. Im Gegensatz zur DRC erlaubt Cadence bei der Extraktion, beliebig viele Vias gleichzeitig zu definieren (DRC: Nur ein Viatyp):

```
(geomConnect
 (via VIA METAL GATE)
 (via PAD METAL GATE)
)
```

Die Definition von Vias markiert die verbundenen Layer, GATE und METAL automatisch als leitf¨ahig.

Alle Objekte, die nicht vermessen werden m¨ussen, extrahiert folgender Befehl gleichzeitig:

```
(saveInterconnect
 (VIA "VIA")
 (PAD "PAD")
 (METAL "METAL")
 (GATE "GATE")
 (LETTER "LETTER")
)
```

Leitungen werden mit dem Verwendungszweck `net` abgespeichert, alle anderen Objekte als `drawing`.

Bauteile

Im ersten Schritt der Bauteilextraktion wird allen Objekten eines Layers gleichzeitig ein Bauteilmodell zugewiesen. Bauteile werden mit dem Befehl `extractdevice` definiert:

```
(extractDevice FET
 ;layer  name of terminal
 (GATE "G")
 (METAL "S" "D")
 "ofet auLvs ptest" physical
)
```

Die Zeilen obigen Beispiels setzen folgende Verkn¨upfungen:

- Den Erkennunglayer des Bauteils (`FET`)
- Die Layer, in denen sich die Anschl¨usse befinden, und deren Namen. Die Namen m¨ussen mit denen im Bauteilmodell identisch sein.
- Den Namen (`ofet`), die Ansicht (`auLvs`), und die Bibliothek (`ptest`) des Bauteilmodells.
- Das Kennwort ‚physical‘, das festlegt, dass alle elektrisch miteinander verbundenen Anschl¨usse in einem Layer als nur ein Anschluss erkannt werden sollen [12]. Dieser Parameter erm¨oglicht das Erkennen mehrerer Source- und Drainfinger als jeweils nur einen Anschluss des Bauteils.

Die Eigenschaften aller Bauteile eines Types k¨onnen mit einem umfangreichen Satz von Funktionen, die in [20] ausf¨uhrlich dokumentiert sind, gemessen werden. Verwendet werden folgende dieser Befehle:

```
(carea = (measureParameter area ( FET over active) ))
(ltmpa = (measureParameter length ( FET butting METAL) ))
(nr    = (measureParameter figCount (FET enclosing active)))
```

Ebenfalls im selben Dokument beschrieben sind große Mengen von arithmetischen Funktionen. In der vorliegenden Arbeit wurden davon nur Addition, Division und Multiplikation verwendet. Alle diese Funktionen werden in der von SKILL und C gewohnten Weise verwendet:

```
(l     = (calculateParameter (ltmpa)/2*nr))
(w     = (calculateParameter carea/l))

(cap=(calculateParameter
 Carea*(gunMetalOverGateCap)+
 Cdiameter*(gunMetalEdgeCap+gunGateEdgeCap)))
```

Nach deren Berechnung werden die Parameter in die CDF- Beschreibung [18] des Bauelements eingetragen:

```
(saveParameter w "W")
(saveParameter l "L")
(saveParameter nr "Nr")
```

Der Erste Parameter ist der Name der Variable aus dem `measureParameter`- oder `calculateParameter`-Befehl. Der zweite Parameter der Funktion entspricht dem Namen des CDF-Parameters. Andere Quellen für Parameter sind nicht erlaubt.

Das Bauteil wird mit dem `saveRecognition`-Befehl in den für Bauteile zuständigen Layer (`device`) eingetragen:

```
(saveRecognition FET "device")
```

Parasitäre Bauteile

Die Detektion parasitärer Bauteile unterscheidet sich von der beabsichtigter Elemente nur in wenigen Punkten:

Parasitäre Kondensatoren entstehen an allen Stellen im Layout, an denen sich METAL- und GATE-Layer überlappen, die sich außerhalb einer CAPDEF-Region befinden. Sie werden in ein eigenes Bauteilmodell, `pcap`, extrahiert, das eine Kopie des Modells für erwünschte Kondensatoren ist.

Für parasitäre Transistoren existieren zwei Detektionsroutinen. Die Autodetektion markiert Transistoren mit geringer Stromtragfähigkeit als parasitär (Kapitel 5.4.3). Bei ausgeschalteter Autodetektion gelten alle Transistoren unter der maximal erlaubten Kanallänge als parasitär, sofern diese nicht mit dem FET-Layer von Hand markiert sind. Das Modell für parasitäre Transistoren (`pofet`) erlaubt nur einen Kanal pro Transistor. Parallelgeschaltete Kanäle werden nicht zusammengefasst.

Diva bietet keine Messfunktionen an, die alle denkbaren Formen von aktiven Gebieten abdecken. Die verwendeten Messfunktionen ermitteln die Länge, auf der die aktive Region Metall berührt und die Fläche der aktiven Region. Dies funktioniert bei rechteckigen und schlangenförmigen Kanälen fehlerfrei.

5.4.5 LVS

Die LVS-Konfiguration trifft genau die Stärken von SKILL. Obwohl diese Aufgabe nicht weniger komplex ist als die vorherigen, lassen sich alle nötigen Funktionen in wenigen, leicht verständlichen und optimierbaren Zeilen zusammenfassen. Cadence führt alle Tests, die hier definiert sind, mit jeder in Frage kommenden Bauteilkombination genau einmal aus, da die Ausführungsgeschwindigkeit von SKILL gering ist. Die wirklich zeitaufwändigen Aufgaben werden vermutlich in C abgearbeitet.

Der Aufbau der LVS-Konfiguration [16] stellt sicher, dass alle Daten, die hierfür gebraucht werden, in einem technologieunabhängigen Format vorliegen.

Um das LVS-Programm möglichst einfach und flexibel zu halten, wurde auf jegliche automatische Optimierung des vom Entwickler geschriebenen SKILL-Codes verzichtet [16].

Alle LVS-Regeln befinden sich in der Datei `divaLVS.rul`, die sich —wie die weiter oben erläuterten Dateien— im Verzeichnis der Bibliothek `ptest` befinden müssen.

Vergleichsfunktionen

Die erste Art der zu schreibenden Funktionen vergleicht Bauteile in Schaltplan und Layout miteinander. Diese Funktionen können eine Fehlermeldung zurückliefern. Falls die Bauteile als identisch gelten können, ergeben

sie den Wert ‚nil‘.

Die Definition der Vergleichsfunktion f¨ur FETs beginnt mit folgenden Zeilen [9, 23]:

```
(procedure compareFET(lay sch)
 (prog ()
```

compareFET ist der Name der Prozedur, der frei w¨ahlbar ist, und lay und sch sind die Namen, unter dem die beiden zu vergleichenden Bauteile im Folgenden bekannt sein sollen.

Die Befehle procedure und proc wurden aus dem Quellcode des EMACS- Texteditors ¨ubernommen.

Parameter, die nicht definiert sind, enthalten den Wert ‚nil‘. Ein Vergleich mit einer Zahl oder Zeichenkette f¨uhrt zu einer Fehlermeldung, die das LVS beendet. Um dies zu vermeiden, empfiehlt es sich, zu ¨uberpr¨ufen, ob die zu vergleichenden Parameter wirklich existieren:

```
(if (lay->L&&sch->L&&lay->W&&sch->W)==nil then
 (return FET not fully defined.")
)
```

lay->W ist der W-Parameter des Bauteils lay, lay->L der L-Parameter desselben.

Die Implementierung des Vergleichs selbst ist trivial:

```
(if lay->W!=sch->W then
 sprintf( gunError, Channel width mismatch: %gu (layout)
  versus %gu (schematic)" float( lay->W) float(lay->W))
 (return gunError)
)    (if lay->L!=sch->L then
 sprintf( gunError, Channel length mismatch: %gu (layout)
  versus %gu (schematic)", float( lay->L), float(lay->L))
 (return gunError)
)
```

Die Fehlermeldung wird hier ¨uber die Variable gunError ausgegeben, da der sprintf-Befehl die M¨oglichkeit bietet, die Parameter des Bauteils auszugeben.

Prozeduren, die keinen return-Befehl ausf¨uhren, liefern automatisch nil zur¨uck und deklarieren so beide Bauteile als identisch.

Vereinfachungsfunktionen

Wenn Schaltbild und Layout vor dem Vergleich auf eine einfachere Form gebracht werden, spart dies Rechenzeit und erh¨oht die Wahrscheinlichkeit, dass ¨aquivalente Schaltungen als identisch erkannt werden. Die Transistoren, deren Source- und Drainstege breiter als gunsdmaxwidth (Tabelle 5.1) sind, wird von der automatischen Transistordetektion jeder Kanal als eigener Transistor aufgefasst. Eine Funktion, die parallelgeschaltete Transistoren zusammenf¨ugt, erm¨oglicht dies zu ignorieren.

Transistoren werden von den implementierten LVS-Regeln nur vereint, wenn sie identische Kanall¨angen haben. Die zugeh¨orige Vereinfachungsfunktion wird folgendermaßen definiert:

```
(procedure
 (mergeparallelFET fet1 fet2)
 (prog (fet)
```

Um den vereinten Transistor zur¨uckliefern zu k¨onnen, ist es n¨otig, eine tempor¨are Variable zu definieren. Sie muss als Liste mit dem Inhalt ‚`(nil)`‘ definiert werden [9, 16, 1]:

```
 (fet = ncons(nil))
```

Der Rest der Funktion arbeitet ¨ahnlich wie obige Vergleichsfunktion f¨ur Transistoren:

```
 (if (fet1->L) && (fet2->L) && (fet1->W) && (fet2->W) then
   (if fet1->L==fet2->L then
   (fet->L=fet1->L)
   (fet->W=fet1->W+fet2->W)
   (if (fet1->Nr) && (fet2->Nr) then
    (fet->Nr=fet1->Nr+fet2->Nr)
   )
   (if (fet1->Cgm) && (fet2->Cgm) then
    (fet->Cgm=fet1->Cgm+fet2->Cgm)
   )
   (return fet)
  )
 )
)
)
```

Die Funktion liefert den neuen Satz von Parametern. Wenn beide Bauteile nicht vereint werden konnten, ergibt die Funktion den Wert ‚`nil`‘.

Der letzte Teil der LVS-Regeln besteht in der Definition, welche Aufgaben in welcher Reihenfolge abgearbeitet werden sollen:

```
(permuteDevice parallel "cap" mergeparallelCap)
```

Dieser Befehl vereint alle parallelgeschalteten Kondensatoren ¨uber die Funktion `mergeparallelCap`. Der Name `"cap"` wurde bei der Extraktion festgelegt.

Auch eine Vereinigung in Serie geschalteter Kondensatoren ist vorgesehen:

```
(permuteDevice series   "cap" mergeseriesCap)
```

Pseudoparallele FETs (Bild 4.5, Seite 29) k¨onnen ebenso vereint werden:

```
(permuteDevice MOS "ofet")
```

Da dieser Schritt in der Praxis in der Regel keinerlei Vorteile bietet (Vgl. Kapitel 4.3), wird er jedoch in der aktuellen Implementierung weggelassen.

Die Vertauschbarkeit der Anschl¨usse von Kondensatoren gegeneinander wird folgendermaßen definiert [16]:

```
(permuteDevice permuteRule "cap" (p CMETAL CGATE))
```

Zu guter Letzt m¨ussen die einzelnen Bauteile in Layout und Schaltplan noch verglichen werden. Dies erledigen folgende Zeilen:

```
(compareDeviceProperty  "cap" compareCap)
(compareDeviceProperty  "ofet" compareFET)
```

Der Algorithmus, nach dem Diva vorgeht, um Layout und Schaltplan vu vergleichen, ist nicht dokumentiert.

Kapitel 6

Ausblick und Zusammenfassung

Ziel dieser Arbeit war es, das Layoutentwicklungssystem Cadence auf eine Technologie anzupassen, die Transistoren auf Polymerfilmbasis verwendet. Konkret waren ein Layout Versus Schaltplan-Test und eine Überpr¨ufung von formalen Entwurfsregeln zu implementieren. Neben der eigentlichen Arbeit und der Dokumentation derselben wurde versucht, zu beschreiben, wie Cadence intern funktioniert, und die Teile, die hier verwendet wurden, an beliebige neue Technologien anzupassen sind.

6.1 Zusammenfassung

Cadence ist ein modular aufgebautes Programmpaket, dass aus viele Einzelprogrammen besteht. Dies macht es unter Anderem m¨oglich, die Anpassung von Cadence an neue Bedingungen auf mehrere Gruppen aufzuteilen. Die nachfolgend beschriebenen Funktionen wurden im Zuge dieser Arbeit realisiert:

6.1.1 Bauelementextraktion

Die vorliegende Technologie unterst¨utzt laut Aufgabenstellung neben den Verdrahtungsleitungen ausschließlich Transistoren als Bauteile. Neben diesen wurden vertikale Kondensatoren als Bauteil implementiert.

Transistoren und Kondensatoren k¨onnen beim Zeichnen des Layouts als solche markiert werden. Eine automatische Transistordetektion erlaubt die Übernahme von Layouts, die ohne diese M¨oglichkeit zu nutzen entworfen worden. Die Detektionsroutinen wurden flexibel gehalten, um nahezu alle m¨oglichen Transistortypen abzudecken. Obwohl die Transistorerkennung generell ein heuristischer Prozess ist, wird die in der vorliegenden Technologie eingesetzte Transistorform fehlerfrei erkannt. Neben den Transistoren k¨onnen auch Anschlusspads und auf das Layout gedruckte Padnamen automatisch als Bauteile detektiert werden.

Die Extraktionsroutinen vermessen automatisch die Kapazit¨at von Kondensatoren. Bei Transistoren werden Kanall¨ange und -Weite, sowie die Gesamtkapazit¨at von Source zu den beiden anderen Anschl¨ussen extrahiert.

Extraktion parasitärer Bauelemente

Parasitäre Bauelemente entstehen bei der Verdrahtung der Bauteile. Sie sind ohne Wissen über die konkrete Realisierung der Schaltung nicht ermittelbar, und werden für genaue Schaltungssimulationen benötigt.

Implementiert sind parasitäre Transistoren und Kondensatoren. Beide Bauteiltypen werden automatisch erkannt und vermessen.

Die Extraktion von Leitungskapazitäten und Leitungswiderständen ist unter dem verwendeten System (Cadence 4.4.3) für die vorliegende Technologie nicht möglich.

6.1.2 Entwurfsregelüberprüfungen

Alle üblichen Entwurfsregelüberprüfungen wurden implementiert. Sie lassen sich in folgende Regeltypen aufteilen:

- Mindestabstände von Objekten zueinander
- Minimale Überlappungen von Objekten
- Minimale Linienbreiten
- Kleinste Einkerbungsweiten
- Minimale Flächen für Durchkontaktierungen und Anschlusspads
- Tests auf nicht eindeutig geformte Polygone
- Umfangreiche Überprüfung der Geometrie von Transistoren

6.1.3 Import von GDF II-Dateien

Der Import und Export von Layouts im GDF II-Format ist implementiert. Neben diesen Funktionen wurden Routinen implementiert, die alle im Zuge dieser Arbeit beobachteten Konvertierungsfehler beheben.

Die Konfigurationsdateien aller Programme wurden so flexibel wie möglich gehalten, um möglichst viele Polymerfilmtechnologien abzudecken. Alle Technologieparameter sind vom Benutzer menügeführt editierbar.

6.2 Ausblick

Studienarbeiten können naturgemäß nur eng begrenzte Aufgabengebiete bearbeiten. Viele Funktionen, die für den erfolgreichen Einsatz einer Technologie sinnvoll bis nötig sind, sind noch nicht implementiert. Dazu gehören folgende Aufgaben:

- Cadence verfügt über umfangreiche Werkzeuge, die das Zeichnen von Layouts stark vereinfachen können. Dazu gehört die automatische Generierung von Bauteilen wie Transistoren oder Kondensatoren mit einem gegebenen Satz von Parametern. Stark mit dieser Aufgabe verwandt ist die Generierung von automatischen Teststrukturen, die die Detektion von Verstößen gegen formale Entwurfsregeln verifizieren.

- Die Implementation eines Autorouters f¨ur die vorliegende Technologie ist aufw¨andiger als f¨ur die etablierten Technologien: Die Verdrahtung ist auf nur zwei Ebenen beschr¨ankt. Außerdem l¨asst sich die Generierung von parasit¨aren Bauteilen durch die Verdrahtung nicht vermeiden. Der Autorouter hat daf¨ur zu sorgen, dass die Auswirkungen dieser parasit¨aren Bauteile minimiert werden.
- Die physikalischen Parameter und Simulationsmodelle f¨ur Technologien, die mit Polymerwerkstoffen arbeiten, sind Gegenstand zahlreicher Untersuchungen. Noch fehlt aber eine Schnittstelle, die den Schaltplan oder die Ergebnisse der Extraktion parasit¨arer Bauteile an einen Schaltungssimulator weitergibt.

Halbleitertechnologien auf Polymerfilmbasis werden zumindest in absehbarer Zeit nicht die hervorragenden Bauteilg¨uten und komplexen Bauelemente eines CMOS-Prozesses bieten k¨onnen. Ihre hervorragende Eignung f¨ur Kleinserien und die niedrigen Investitionskosten d¨urften dieser Technologie dennoch viele interessante Einsatzgebiete erschließen.

Anhang A

SKILL

Die Programmiersprache SKILL dient in Cadence als allgemeines Konfigurierungswerkzeug.

Wegen der niedrigen Geschwindigkeit (SKILL l¨asst sich aufgrund der Komplexit¨at dieser Aufgabe noch nicht in Maschinencode umsetzen) und dem Speicherverbrauch von SKILL beschr¨ankt sich der Einsatz von SKILL haupts¨achlich auf selten durchlaufenen Programmteile.

Um Zeit beim Interpretieren zu sparen erlaubt Cadence allerdings, SKILL zu compilieren. Compiler f¨ur reine Interpretersprachen nummerieren in der Regel die wichtigsten Befehle durch, und legen Zahlenwerte in Bin¨arform ab [8]. Dieses Verfahren wird auch als„Tokenisieren“ bezeichnet. Das Tokenisieren spart viel Rechenzeit bei der Interpretation des Programmcodes.

Die Syntax von SKILL unterscheidet sich von der etablierten Programmiersprache C am augenf¨alligsten darin, dass Befehle nicht von Klammern gefolgt werden, sondern selbst mit eingeklammert sind. Auch Bl¨ocke, zu denen mehrere Befehle zusammengefasst sind, und die zum Beispiel einen Zweig einer Verzweigung bilden, werden in runde Klammern eingefasst.

Der letzte augenf¨allige Unterschied ist die Trennung der Argumente von Funktionen durch Leerzeichen voneinander. In den ¨ublichen Programmiersprachen C, Basic und Pascal geschieht dies durch Kommata.

Der Hauptvorteil von SKILL ist die Listenorientiertheit: Programme sind Listen, Variablen potentiell auch. Funktionen k¨onnen so Zahlen als Argumente erhalten, Zeichenketten, oder ganze Programme, die dynamisch ermittelte Werte liefern. Einfach und ¨ublich ist auch,,Hooks‘ einzuf¨uhren. Dies sind Variablen, die Programme enthalten, die vor und/oder nach wichtigen Aufgaben ausgef¨uhrt werden.

Jeder Entwickler kann an die Hooks, die seine Arbeit betreffen, eigene Funktionen anh¨angen. So gibt es zum Beispiel einen Hook, der ausgef¨uhrt wird, wenn ein Objekt im Editor seine Form ¨andern soll, oder neue Parameter zugewiesen bekommt. Falls es n¨otig ist, ein Bauteil z.B. an eine Gr¨oßen¨anderung [15] anzupassen, tr¨agt man die Anpassungsroutine in den Hook ein.

Einzelne Listenelemente k¨onnen selbst wieder Listen sein, so dass es oft m¨oglich ist, eine komplexe Funktion ¨uber eine kurze, ¨ubersichtliche Liste zu realisieren, die einem Programm ¨ubergeben wird.

Variablen m¨ussen vor ihrer ersten Verwendung nicht formal deklariert werden. C und Pascal verlangen vom Benutzer vor der ersten Verwendung einer Variable die Angabe des Typs derselben. Tippfehler in Variablennamen f¨uhren so mit hoher Treffsicherheit zu Fehlermeldungen beim Compilieren des Programms. SKILL erzeugt Variablen dynamisch bei deren Zuweisung, was Schreibarbeit spart. Variablentypen werden bei der

ersten Zuweisung zu einer Variablen ermittelt. Zu beachten ist, dass der erste Wert, der einer Fließkommavariable zugewiesen wird, ein Komma zu enthalten hat (z.B. `12.0`). Die Variable wird ansonsten ausschließlich f¨ur ganze Zahlen verwendbar sein. Der Versuch, noch nicht definierte Variablen auszulesen, f¨uhrt zu einer Fehlermeldung.

Aufw¨andiges Speichermanagement von Benutzerseite, wie in Pascal und C n¨otig, entf¨allt. Der Speicherverbrauch einer Variable wird automatisch dynamisch ermittelt. Nicht mehr ben¨otigte Listen werden im Zuge eines ‚Garbage Collects' bei Bedarf automatisch aus dem Speicher entfernt [8].

Globale Variablen sollten auch in SKILL nur selten verwendet werden. Alle in den von dieser Arbeit verwendeten Dateien definierten Variablen sind automatisch lokal f¨ur die jeweilige Datei.

Cadence-interne Variablen beginnen mit einem Unterstrich. Dem Entwickler wird empfohlen, seine eigenen Variablen mit einem 3-Buchstaben-Pr¨afix (in dieser Arbeit,`gun`', die Anfangsbuschstaben des Vornamens des Programmierers) zu beginnen [1, Kapitel 2]. Dies vermeidet Kollisionen mit Variablen von anderen Projekten. Der Pr¨afix ist frei w¨ahlbar. Im vorliegendem Fall wurde der Anfang des Vornamens des Programmierers verwendet.

SKILL versteht die konventionelle Schreibweise f¨ur Formeln:

```
i=5*(o+7)
```

Intern wird eine umgekehrte polnische Notation verwendet:

```
(setq i (times 5 (plus o 7)))
```

Die umgekehrte polnische Notation hat f¨ur die Programmierung ausschließlich historische Bedeutung. Formeln in Fehlermeldungen werden jedoch in dieser Schreibweise ausgegeben([1, Kapitel 3]).

Die Wahrheitswerte „Wahr“ und „Falsch“ werden in SKILL ¨uber die Variablenwerte‚**`t`**' und ‚**`nil`**' beschrieben.

A.1 Die wichtigsten SKILL-Befehle

Neben Befehlen, die von allen Applikationen unterst¨utzt werden, existieren große Mengen applikationsspezifischer Befehle. Des Weiteren erlauben einige Applikationen die Definition von eigenen SKILL-Funktionen oder den Zugriff auf Variablenkonstrukte, die an die Programmiersprache C angelehnt sind. Um diesen Anhang nicht zu sehr aufzubl¨ahen, und unn¨otige Querverweise zu vermeiden, wurden diese zusammen mit der jeweiligen Applikation dokumentiert, soweit f¨ur Verst¨andnis oder Implementierung dieser Arbeit n¨otig.

A.1.1 Text-Behandlungs-Routinen

`strcat`	f¨ugt Zeichenketten zusammen.
`print`(Objekt)	druckt ein Objekt (Zahl, Zeichenkette, . . .) aus.

`println`(Objekt) arbeitet wie `print`, führt jedoch nach der Ausgabe einen Zeilenvorschub aus.

`printf`(”format“ argumente) arbeitet analog zu `print`, unterstützt allerdings Formatargumente analog der printf-Funktion von C (Tabelle A.1).

Tabelle A.1: Die Formatargumente der `printf`-Funktion

%d	ganze Zahl
%f	Fließkommazahl
%s	Zeichenkette (string).
%c	Einzelnes Zeichen
%n	Zahl
%L	Liste
%P	Liste von einzelnen Punkten
%B	Liste von Rechtecken
%%	Ein Prozentzeichen

Zwischen Prozentzeichen und Buchstaben kann man außerdem ein Minuszeichen für linksbündige Ausgabe und den Parameter ‚`Vorkommastellen.Nachkommastellen`‘ einfügen:
‚`%-2.1f`‘bedeutet also: „Gebe eine Fließkommazahl mit einer Nachkommastelle linksbündig aus, und lasse Platz für zwei Vorkommastellen.“

A.1.2 Listen

SKILL besitzt eine große Menge an Befehlen, die zur Behandlung von Listen dienen. Die beim Schreiben dieser Arbeit verwendeten Befehle sind folgende:

`(cons` a liste) hängt einer Liste ein neues erstes Element a an. Elemente an das Ende einer Liste anzuhängen ist mit diesem Befehl nicht möglich.
Ein Element an einer anderen Position an Listen anzufügen ist mit diesem Befehl nicht möglich.

(`ncons` a) `ncons` liefert eine Liste, die ausschließlich das Element `a` enthält.

`(append` liste1 liste2) erzeugt eine Liste, die aus allen Elementen beider Argumente besteht.

`(car` liste) liefert das erste Element einer Liste.

`(cdr` liste) liefert alle Elemente bis auf das Erste. Die Befehle `car` und `cdr` sind nach Assemblerbefehlen des ersten Rechners, auf dem Lisp, der Vorgänger von SKILL, realisiert wurde benannt. Sie dienen dazu, Listen elementweise abzuarbeiten.
Aufeinanderfolgende `car` und `cdr`-Befehle können in einer Kurzschreibweise geschrieben werden. Die Kurzbefehle beginnen alle mit einem ‚`c`‘. Danach folgen ein ‚`a`‘ für jeden `car`- und ein‚`d`‘ für jeden `cdr`-Befehl. Die Kurzbefehle enden alle mit einem ‚`r`‘:
`caadr` entspricht `(car(car(cdr(a))))`.

`(nth` n liste)	Gibt n-te Element einer Liste zur¨uck. Die Nummerierung der Listenelemente beginnt hierbei mit dem Element Nummer 0.
`(length` liste)	Z¨ahlt die Elemente einer Liste.

A.1.3 Dateibehandlung

Nahezu alle Programmiersprachen greifen auf Dateien ¨uber Dateidescriptoren zu. Diese sind Nummern, oder Speicherbereiche, die f¨ur den Rechner einfacher zu handhaben sind, als die Dateinamen selbst. F¨ur jeden Dateideskriptor verwaltet der Rechner einen eigenen Zeiger auf das n¨achste zu lesende oder zu schreibende Zeichen. Die selbe Datei kann ¨uber mehrere Dateidescriptoren gleichzeitig ausgelesen werden.

`(outfile` name)	Erstellt eine neue Datei, und liefert einen Dateideskriptor auf diese Datei. Falls die Datei vor dem Öffnen existiert, wird der Inhalt gel¨oscht.
`(infile` name)	Liefert einen Dateideskriptor, der das Lesen aus einer Datei erlaubt.
`(fprint` Deskriptor Objekt) `(fprintln` ...) `(fprintf` ...)	Die `fprint`-Funktionen funktionieren analog den `print`-Funktionen (S. 65), nur dass sie in die zum Deskriptor geh¨orende Datei schreiben, anstatt auf den Bildschirm.
`gets`(Deskriptor)	Liest eine Zeile aus einer Datei ein, und liefert deren Inhalt zur¨uck. Ein `nil` als Resultat bedeutet Dateiende.

A.1.4 Koordinaten

Die Koordinaten von Objekten werden in Listen zusammengefasst.

Obwohl die Schreibweise von Koordinaten von denen von Listen durch die Doppelpunkte differiert, lassen sie sich auf die ¨ubliche Weise auslesen:

Koordinaten eines Punktes

Die Koordinaten des Punktes stehen in der Variable

`P=(`x-Koordinate`:`y-Koordinate`)`

Die Befehle zur Ermittlung der Zahlenwerte sind folgende:

x	y
`(car` P)	`car(cdr` P)

Koordinaten eines Rechtecks

Ein Rechteck besteht aus zwei Punkten:

R=((x1:y1) (x2:y2))

Polygone entstehen auf die selbe Weise [1]. Die Zahlenwerte der Koordinaten eines Rechtecks werden folgenderweise ermittelt:

Linke untere Ecke			Rechte obere Ecke		
x	y	Beide	x	y	Beide
(caar R)	**(cadar** R)	**(car** R)	**(caadr** R)	**(cadar** R)	**(cadr** R)

In obiger von SKILL unterstützten verkürzten Syntax entspricht jeder Buchstabe zwischen dem‚c‘ und dem ‚r‘ einem `cdr`-oder `car`-Befehl: `(cadar` R)≡`(car(cdr(car` R)))

A.1.5 Verzweigungen und Schleifen

Die Verzweigungsbefehle sind denen nahezu aller anderen verbreiteten Programmiersprachen sehr ähnlich.

Die Funktionen der Verzweigungsbefehle sind folgende:

`if` führt die ersten Befehle genau dann aus, wenn die Bedingung erfüllt ist. Die Befehle hinter dem `else` werden ausgeführt, wenn die Bedingung nicht erfüllt war. Der `else`-Zweig ist optional. Die Schreibweise für den if- Befehl ist folgende:

```
(if Bedingung then
   Befehle  else
   Befehle )
```

`case` Testet eine einzelne Variable auf Gleichheit mit einer ganzen Reihe von Werten. Dieser Befehl wird oft verwendet, um für jede Variante eines Bauteils einen anderen Programmteil auszuführen:

```
(case Variable
   (Fall 1
     Befehle   )
   (Fall 2
     Befehle
   )
 )
```

Das `case`-Statement hat allerdings zwei Besonderheiten: Man kann statt Ereignissen auch Listen von Ereignissen angeben, von denen nur eins erfüllt sein muss, um die Bedingung zu erfüllen. Ein‚t‘ als letzte Bedingung gilt dann als erfüllt, wenn sonst kein Ereignis zutrifft:

```
(case Layer
 (FET    (print "Feldeffekttransistor"))
 (CAPDEF (print "Kondensator"))
 ((VIA PAD)
        (print "Durchkontaktierung")
 )
 (t      (print "Unbekanntes Bauteil"))
)
```

Zus¨atzlich zu diesen existieren einige Schleifenbefehle:

`while` F¨uhrt eine Liste von Befehlen so lange aus, bis die Bedingung nicht mehr erf¨ullt ist. Ist die Bedingung von vornherein nicht erf¨ullt, wird dieser Befehl ¨ubersprungen.

```
(while Bedingung
  Befehle
)
```

`unless` Das Gegenteil von `while`. F¨uhre die Befehle aus, bis die Bedingung erf¨ullt ist:

```
(unless Bedingung
  Befehle
)
```

`for` Z¨ahlt die Variable von Start bis Ende in Einerschritten durch, und f¨uhrt jedesmal die gegebenen Befehle aus:

```
(for Variable Start Ende
  Befehle


)
```

Dies erm¨oglicht, den selben Programmteil z.B. 100 Mal auszuf¨uhren.

`foreach` F¨uhrt die Befehle f¨ur jedes Element der Liste genau Einmal aus. Die Variable enth¨alt dabei jedes Mal ein anderes Element der Liste, beginnend mit dem ersten Listenelement:

```
(foreach Variable Liste
  Befehle
)
```

Obige Schreibweisen der Befehle entsprechen der empfohlen Syntax in [1]. Viele andere —darunter C-¨ahnliche Notationen sind in den meisten Programmen ebenso m¨oglich.

Ein einfaches Beispiel f¨ur die Konfiguration von Cadence-Programmen in SKILL ist die Datei `display.drf` (Kapitel 5.2)

A.2 Verwendete Schalter

In den Fenstern, von denen aus DRC und Extraktion gestartet werden (Bilder 3.9 und 3.8) k¨onnen Schalter gesetzt werden. Die im Zuge dieser Arbeit implementierten Schalter sind in Tabelle A.2 dokumentiert.

Tabelle A.2: Verwendete Schalter

drc?	Cadence setzt den Schalter ‚drc?‘ automatisch, wenn ein Design Rule Check aufgerufen wird.
extract?	Dieser Schalter wird bei Extraktionen automatisch von Cadence gesetzt.
letterautodetection	Dieser Schalter legt fest, dass nicht elektrisch verbundene METAL-Objekte als Metallbuchstaben behandelt werden sollen.
padautodetection	legt fest, dass Durchkontaktierungen, deren Umfang einen bestimmten Betrag ¨ubersteigt, als Anschlusspads angesehen werden sollen.
killstreamgaps	f¨ullt kleine durch Konvertierungsfehler entstandene L¨ucken in elektrisch leitf¨ahigen Materialschichten.
noFETautodetect	deaktiviert die automatische FET-Erkennung.
check_dubiousData	besagt, dass die Objekte in den Eingangslayern von Diva auf formale Fehler ¨uberpr¨uft werden sollen.
allowserpentshapedfet	Transistoren sind in der Regel rechteckig. Falls Transistoren, die anders geformt sind, erw¨unscht sind, k¨onnen diese durch diesen Schalter als erlaubt deklariert werden.
extractpcaps	tr¨agt alle detektierten parasit¨are Kondensatoren in das extrahierte Layout ein.
extractpfets	arbeitet analog extractpcaps, detektiert jedoch parasit¨are Transistoren.

Anhang B

Implementierte DRCs

Die nachfolgenden Tabellen listen die deutschen Übersetzungen aller Fehlermeldungen, die die implementierten DRCs generieren k¨onnen, auf. Allen Fehlermeldungen wurde ein zweistelliger Fehlercode vorangestellt. Die Auflistung erfolgt im Gegensatz zu Kapitel 4.1, soweit m¨oglich, in Physikalischer Reihenfolge der Layer.

Die graphisch darstellbaren Entwurfsregeln wurden in die Bilder B.1–B.5 eingezeichnet.

B.1 METAL (Bild B.1)

NC: METAL darf kein HRES-Gebiet ¨uberlappen
NF: METAL darf kein LETTER-Gebiet ¨uberlappen
D8: Mindestabstand METAL–HRES
D9: Mindestabstand METAL–GATE außerhalb von FETs
DA: Mindestabstand METAL–GATE innerhalb von FETs
DB: Mindestabstand METAL–METAL außerhalb von FETs
DC: Mindestabstand METAL–METAL innerhalb von FETs
DG: Mindestabstand METAL–LETTER
A4: Vias m¨ussen von METAL ganzfl¨achig ¨uberlappt werden
A5: Pads m¨ussen von METAL ganzfl¨achig ¨uberlappt werden
O5: Minimale Überlappung METAL ¨uber FET
O6: Minimale Überlappung METAL ¨uber VIA
O7: Minimale Überlappung METAL ¨uber PAD
M7: Fehlerhaft definierter Polygonenzug
ME: Nicht achsenparallele Kanten
MJ: Minimale Linienbreite außerhalb von FETs
MK: Minimale Linienbreite innerhalb von FETs
MQ: Minimale Einkerbungsweite
MZ: Nichtrechteckiges aktives Gebiet
MY: Minimale Kanall¨ange

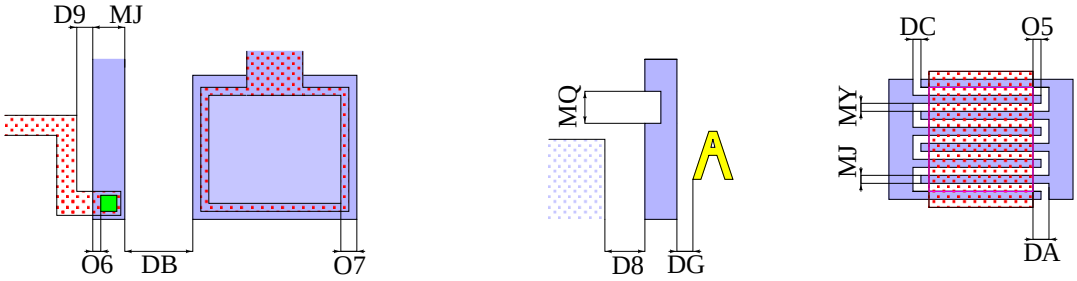


Abbildung B.1: DRCs f¨ur METAL

B.2 PAD/VIA (Bild B.2)

N5: VIA darf kein CAPDEF-Gebiet ¨uberlappen
N6: VIA darf kein FET-Gebiet ¨uberlappen
NA: VIA darf kein PAD-Gebiet ¨uberlappen
N8: PAD darf kein CAPDEF-Gebiet ¨uberlappen
N9: PAD darf kein FET-Gebiet ¨uberlappen
D3: Mindestabstand VIA-VIA
D4: Mindestabstand VIA-PAD
D5: Mindestabstand PAD-PAD
DJ: Mindestabstand VIA-FET
DL: Mindestabstand PAD-FET
DI: Mindestabstand VIA-CAPDEF
DK: Mindestabstand PAD-CAPDEF
A2: VIAs m¨ussen von GATE ganzfl¨achig ¨uberlappt werden
A3: PADs m¨ussen von GATE ganzfl¨achig ¨uberlappt werden
O3: Minimale Überlappung GATE ¨uber VIA
O4: Minimale Überlappung GATE ¨uber PAD

B.2.1 PAD

M5: Fehlerhaft definierter Polygonenzug
MC: Nicht achsenparallele Kanten
MH: Minimale Linienbreite
MO: Minimale Einkerbungsweite

B.2.2 VIA

M4: Fehlerhaft definierter Polygonenzug
MB: Nicht achsenparallele Kanten
MG: Minimale Linienbreite
MN: Minimale Einkerbungsweite

B.3 GATE (Bild B.3)

A1: FET- Gebiet außerhalb eines GATE- Gebiets
NB: GATE darf kein HRES-Gebiet ¨uberlappen
NE: GATE darf kein LETTER-Gebiet ¨uberlappen

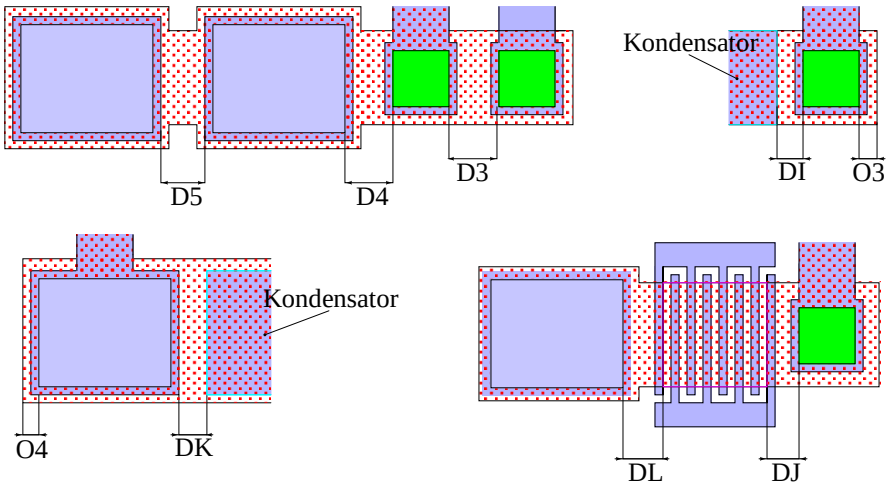


Abbildung B.2: DRCs f¨ur VIA

D6: Mindestabstand GATE–HRES
D7: Mindestabstand GATE–GATE
O2: Minimale Überlappung GATE ¨uber FET
M6: Fehlerhaft definierter Polygonenzug
MD: Nicht achsenparallele Kanten
MI: Minimale Linienbreite
MP: Minimale Einkerbungsweite

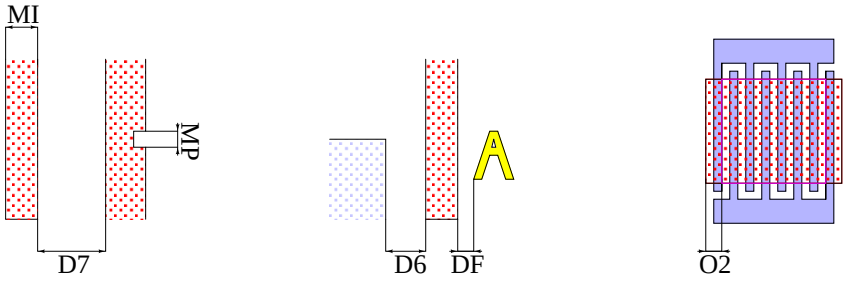


Abbildung B.3: DRCs f¨ur GATE

B.4 HRES (Bild B.4)

ND: HRES darf kein LETTER-Gebiet ¨uberlappen
DD: Mindestabstand HRES–LETTER
M1: Fehlerhaft definierter Polygonenzug
MF: Minimale Linienbreite
ML: Minimale Einkerbungsweite

B.5 FET (Bild B.5)

N3: FET darf kein CAPDEF-Gebiet ¨uberlappen

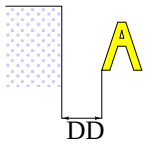


Abbildung B.4: DRCs f¨ur HRES

DH: Mindestabstand FET–CAPDEF
D2: Mindestabstand FET–FET
M3: Fehlerhaft definierter Polygonenzug
MA: Nicht achsenparallele Kanten
MR: FET nicht rechteckig.

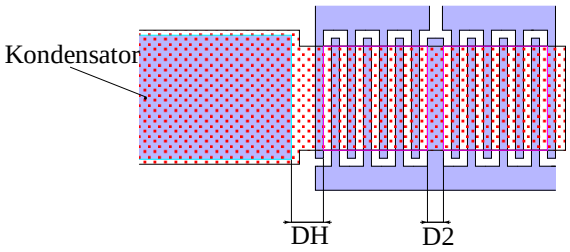


Abbildung B.5: DRCs f¨ur FET

B.6 LETTER

M8: Fehlerhaft definierter Polygonenzug

B.7 CAPDEF

M2: Fehlerhaft definierter Polygonenzug
M9: Nicht achsenparallele Kanten
C1: CAPDEF enth¨alt keine Kondensatoren
C2: Kondensator kreuzt den Rand eines CAPDEF-Gebietes

Literaturverzeichnis

[1] Cadence Design Systems, Inc. *SKILL Language User Guide*, product version 06.00 edition, June 2000. www.sli.strath.ac.uk/sklanguser.pdf, 17. August 2001.

[2] Cadence Design Systems, Inc. *Virtuoso Layout Editor User Guide*, product version 4.4.6 edition, June 2000. www.sli.strath.ac.uk/vlehelp.pdf, 17. August 2001.

[3] Cadence Design Systems, Inc. *Diva design rules, Overview*, product version 06.00 edition, June 2000. icverif/divaref/chap1.obk im Dokumentationsverzeichnis von Cadence.

[4] Cadence Design Systems, Inc. *Diva design rules, Design Rule Checking (iDRC)*, product version 06.00 edition, June 2000. icverif/divaref/chap7.obk im Dokumentationsverzeichnis von Cadence.

[5] Cadence Design Systems, Inc. *Diva design rules, Extracting Parasitic Resistance (iPRE)*, product version 06.00 edition, June 2000. icverif/divaref/chap11.obk im Dokumentationsverzeichnis von Cadence.

[6] Cadence Design Systems, Inc. *Diva design rules, Processing Layers*, product version 06.00 edition, June 2000. icverif/divaref/chap5.obk im Dokumentationsverzeichnis von Cadence.

[7] Cadence Design Systems, Inc. *Technology File and Display Resource File User Guide*, product version 4.4.3 edition, March 1999. dfIIcore/techfileuser/techfileuserTOC.obk im Dokumentationsverzeichnis von Cadence.

[8] The GNU software Foundation. Die Online-Dokumentation des EMACS-Texteditors. ftp://prep.ai.mit.edu/gnu/pub, 2. April 2002, 2001.

[9] The GNU software Foundation. Die Quellcodes des EMACS-Texteditors. http://www.prep.ai.mit.edu/gnu/pub, 2. April 2002, 2001.

[10] Cadence Design Systems, Inc. *Virtuoso Relative Object Design User Guide*, product version 4.4.6 edition, June 2000. http://www.ee.ucr.edu/~czhang/roduser.pdf, 12. November 2001.

[11] Cadence Design Systems, Inc. *Sample Parameterized Cells Installation and Reference*, product version 4.4.6 edition, June 2000. vlsilab.ee.uec.ac.jp/gakusei/home/cad/cadence.new/doc/pclib/appA.html, 12. November 2001.

[12] Physical Verification. http://www.utdallas.edu/~rvanz/cadence/tools/extraction/Extraction.pdf, 12. November 2001.

[13] Ambrish Kant Varma. Computer aided Tools for seamless high density Interconnects. Masters thesis, Graduate Faculty of North Carolina State University, 2001. www.lib.ncsu.edu/etd/public/etd-4618171031012991/, 12. November 2001.

[14] Cadence Design Systems, Inc. *Diva Reference, Checking Electrical Rules (iERC)*, product version 4.4.3 edition, November 1998.

[15] Cadence Design Systems, Inc. *Component Description Format User Guide, Defining Parameters*, product version 4.4.3 edition, November 1998.

[16] Cadence Design Systems, Inc. *Diva Reference, Comparing Layout to Schematic (iLVS)*, product version 4.4.3 edition, November 1998.

[17] Cadence Design Systems, Inc. *Component Description Format User Guide, CDF Commands*, product version 4.4.3 edition, November 1998.

[18] Cadence Design Systems, Inc. *Component Description Format User Guide, What is CDF*, product version 4.4.3 edition, November 1998.

[19] Cadence Design Systems, Inc. *Diva Reference, Extracting Parasitics (iLPE)*, product version 4.4.3 edition, November 1998.

[20] Cadence Design Systems, Inc. *Diva Reference, Extracting Parasitics (iLPE)*, product version 4.4.3 edition, November 1998.

[21] Cadence Design Systems, Inc. *Diva Reference, Extracting Connectivity*, product version 4.4.3 edition, November 1998.

[22] Cadence Design Systems, Inc. *Diva Reference, Executing Diva Using SKILL and UNIX commands*, product version 4.4.3 edition, November 1998.

[23] Cadence Design Systems, Inc. *Diva Reference, Executing Diva Using SKILL and UNIX commands*, product version 4.4.3 edition, November 1998.

[24] Cadence Design Systems, Inc. *Vampire User Guide, Differences betwenn Diva and Vampire*, product version 4.4.3 edition, November 1998.

[25] Cadence Design Systems, Inc. *Technology File and Display Resource File User Guide*, product version 4.4.3 edition, March 1999. dfIIcore/sktechfile/sktechfileTOC.obk im Dokumentationsverzeichnis von Cadence.

[26] W. Glauert. Scriptum zur Vorlesung Design of Integrated Circuits I, 2000/2001. Gehalten an der Universit¨at Erlangen- N¨urnberg.

[27] W. Glauert. Scriptum zur Vorlesung Design of Integrated Circuits II, 2001. Gehalten an der Universit¨at Erlangen- N¨urnberg.

[28] Eike Becker, Hans-Hermann Johannes, Torsten Benstem, Thomas Dobbertin, Dirk Heithecker, Dirk Metzdorf, Helge Neuner, and Wolfgang Kowalsky. A New Structuring Technique for Polymer Integrated Circuits. In *Polytronic 2001*. First international IEEE Conference on Polymers and Adhesives in Microelektronics and Photonics, 2001.

[29] M. Schr¨odner, S. Sensfuss, H.-K. Roth, R.-I. Strohn, W. Clemens, A. Bernds, and W. Fix. Plastic Electronics Based on Semiconducting Polymers. In *Polytronic 2001*. First international IEEE Conference on Polymers and Adhesives in Microelektronics and Photonics, 2001.

[30] Alexander Knobloch, Adolf Bernds, and Wolfgang Clemens. Printed Polymer Transistors. In *Polytronic 2001*. First international IEEE Conference on Polymers and Adhesives in Microelektronics and Photonics, 2001.

[31] Cadence Design Systems, Inc. *Technology File and Display Resource File User Guide, Creating a Technology File: Control, Layer, and Device Definitions*, product version 4.4.6 edition, June 2000. http://vlsilab.ee.uec.ac.jp/gakusei/home/cad/cadence/doc/techfileuser/chap3.html, 12. November 2001.

[32] International Technology Roadmap for Semiconductors. http://public.itrs.net/files/1999_SIA_Road_map/Home.htm, NOTE=12. November 2001, 1999.

Tabellenverzeichnis

Abbildungsverzeichnis

Index