Grafische Editoren für den Entwurf von VLSI-Systemen

Autor(en): **Weibel, B. / Morf, M.**

Objekttyp: **Article**

Zeitschrift: **Bulletin des Schweizerischen Elektrotechnischen Vereins, des Verbandes Schweizerischer Elektrizitätsunternehmen = Bulletin de l'Association Suisse des Electriciens, de l'Association des Entreprises électriques suisses**

Band (Jahr): **74 (1983)**

Heft 5

PDF erstellt am: **12.07.2024**

Persistenter Link: https://doi.org/10.5169/seals-904774

Grafische Editoren für den Entwurf von VLSI-Systemen

B. Weibel, M. Morf

Heutige CAD-Systeme sind komplex, schwierig zu unterhalten oder zu erweitern und daher sehr teuer. Zukünftige CAD-Software wird durch moderne Informatikwerkzeuge effizienter entwickelt und unterhalten werden können. Zukünftige CAD-Hardware wird hauptsächlich wegen der Entwicklung von VLSI-Chips billiger, schneller und mächtiger werden.
Es wird ein grafischer Editor beschrieben, und es werden Beispiele von Algorithmen und Datenstrukturen für diesen Editor gezeigt. Anwendungsbeispiele sind gedruckte Schaltungen sowie Logik-Diagramme, Stick-Diagramme und Layout-Diagramme, die effizientes Entwerfen von VLSI-Systemen erlauben.

Les systèmes actuels de conception assistée par ordinateur (CAO) sont complexes, malaisés à entretenir ou à développer, donc très coûteux. Désormais, le logiciel de CAO sera développé et entretenu plus efficacement par les moyens modernes de l'informatique et le matériel deviendra moins coûteux, plus rapide et plus puissant, surtout par le développement de puces à intégration à très grande échelle. On décrit un éditeur graphique et donne des exemples d'algorithmes et de structures de données pour celui-ci. Des applications sont les circuits imprimés, les diagrammes de logique, diagrammes de stick et diagrammes de layout, permettant une conception efficace de systèmes d'intégration à très grande échelle.

Dieser Aufsatz entspricht dem Vortrag des erstgenannten Autors anlässlich des «Fall Meeting 1982 on Computer Aided Design (CAD)» der IEEE Swiss Section, Chapter on Solid State Devices and Circuits, am 19. Oktober 1982 in Bern.

Adressen der Autoren
B. Weibel, Institut für Informatik, ETH-Zentrum, 8092 Zürich
Prof. Dr. *M. Morf*, Yale University, New Haven, Conn., USA

1. CAD-Systeme

Heute erhältliche CAD-Systeme[1]) sind oft komplex und sehr gross (typischerweise einige 10 000 Zeilen Fortran). Da solche Systeme über Jahre gewachsen sind und sich im Laufe der Zeit spezifischen CAD-Anwendungen anpassen mussten, für welche sie ursprünglich nicht vorgesehen waren, sind sie nicht hierarchisch strukturiert und zuwenig modular aufgebaut. Daraus ergibt sich der schwerwiegende Nachteil, dass diese Systeme nur mühsam zu unterhalten und zu erweitern sind.

Dank moderner Informatikwerkzeuge wird es in Zukunft möglich sein, Software effizienter zu entwickeln und zu unterhalten. Und gerade dank den Fortschritten auf dem Gebiet der hochintegrierten Schaltungen wird in Zukunft billigere, schnellere und mächtigere Hardware entwickelt werden können.

Mit fortschreitender Verbesserung der VLSI-Technologien[2]) wächst die Anzahl Transistoren, die auf einem Chip untergebracht werden können, sehr rasch. Da die Kosten für das Entwerfen von Schaltungen nahezu so schnell wachsen wie die Komplexität der Schaltungen zunimmt, ist es notwendig, Strukturen mit möglichst grosser Regelmässigkeit und möglichst oft dieselben Strukturen zu verwenden. Solche Betrachtungen haben schon früh zu Ansätzen geführt, bei denen Standardzellen in Reihen vordefiniert sind (z. B. Gate-Arrays). Die Aufgabe des Entwerfens besteht dann darin, die nötigen Gates geeignet zu plazieren und richtig zu verbinden.

Mit zunehmender Komplexität der Schaltungen ist auch die Notwendigkeit erkannt worden, CAD-Programme zu entwickeln, die beim Schaltungsentwurf nützlich sind. So gibt es Programme, die versuchen, das eben erwähnte Plazieren und Verbinden von Gates automatisch vorzunehmen. In anderen Ansätzen wird versucht, für ganze Klassen von (einfacheren) VLSI-Architekturen den Vorgang des Entwerfens zu automatisieren. Die Tatsache, dass solches automatisches Entwerfen weniger gute Resultate liefert als eine Optimierung von Hand, kann sich unter Umständen nachteilig auswirken. Ein vielversprechender Mittelweg besteht darin, dem Benützer Zwischenresultate des automatischen Entwerfens zu zeigen und ihm Gelegenheit zu geben, kritische Bereiche von Hand zu optimieren und den Entwurf am Schluss zu vervollständigen.

[1]) CAD = Computer Aided Design
[2]) VLSI = Very Large-Scale Integration

2. Individuelle Arbeitsplatzrechner

Die Fortschritte auf dem Gebiet der Hardware zusammen mit modernen Methoden der Informatik auf dem Gebiet der Software haben in jüngster Zeit zur Entwicklung von leistungsfähigen und kostengünstigen Arbeitsplatzrechnern geführt. Eine höhere Programmiersprache und Hilfsmittel wie interaktive Debugger und Editoren sind in der Lage, den Software-Ingenieur beim Entwickeln von Programmen so zu unterstützen, dass er in der gleichen Zeit viel produktiver ist. Ein ständig für den persönlichen Gebrauch verfügbares Computersystem bildet eine ideale Arbeitsumgebung, wenn durch dessen Einfachheit und Klarheit und durch entsprechende Unterstützung mit Systemsoftware eine bequeme Benützung gewährleistet ist.

Für den Einsatz eines Arbeitsplatzrechners kommen vor allem Anwendungsgebiete in Frage, bei denen die direkte Verfügbarkeit und die Interaktivität des Computersystems besonders gut ausgenützt werden können. Auch beim Entwurf von VLSI-Systemen ist es zweckmässig, für diejenigen Teilprobleme, bei denen es auf eine intensive Interaktion zwischen Benützer

und Programm ankommt, Lösungen auf einem Arbeitsplatzrechner zu implementieren.

3. Entwurf von VLSI-Systemen

Es ist üblich, für die Beschreibung von VLSI-Systemen Darstellungen auf verschiedenen Stufen zu verwenden. Wenn alle geometrischen Details einer ausgelegten Schaltung sichtbar sein sollen, spricht man von Layout-Diagrammen. Wenn nur die Topologie der Verbindungen der Schaltung wesentlich ist, kann man sogenannte Stick-Diagramme verwenden. Als weitere Abstraktion stehen die üblichen Schaltungsdiagramme und Logik-Diagramme zur Verfügung. Beispiele für die verschiedenen Darstellungsformen sind in [1*, Kap. 3.2], einem Standardwerk über VLSI-Systeme, zu finden.

Ein Softwarepaket für das interaktive Entwerfen von VLSI-Systemen kann zum Beispiel folgende Funktionen anbieten: eine Sprache zur Beschreibung von Hardware (hardware description language), die automatische Übersetzung zwischen dieser Sprache und Diagrammen in verschiedenen Darstellungsarten, eine Familie von Editoren für das grafische Bearbeiten der Diagramme aller Stufen, Hilfsprogramme für Output auf Drukker oder Plotter, verschiedene Simulationsmöglichkeiten, das automatische Erzeugen von Testmustern und schliesslich das Berechnen der Masken aus den Diagrammen. Es ist sogar möglich, dass ein solches CAD-System Hilfsmittel zur Verfügung stellt, um Schaltungen in verschiedenen Technologien (z.B. als gedruckte Schaltung oder als integrierte Schaltung) zu implementieren.

Einige Teilaufgaben beim VLSI-Entwurf eignen sich für eine automatische Bearbeitung. Beispielsweise gibt es Programme, die recht kompakte Schaltungen erzielen, indem sie Stick-Diagramme automatisch in Layout-Diagramme übersetzen. Für andere Problemkreise ist es gut, wenn der Benützer die Möglichkeit hat, von Hand zu entwerfen.

4. Systemintegration

Ein CAD-System für das Entwerfen grosser VLSI-Systeme soll dem Benützer die Möglichkeit geben, seine Schaltung in unabhängige Blöcke zu zerlegen, um einerseits die Übersichtlichkeit des Entwurfs zu erhöhen und anderseits das Aufteilen in lokal zu behandelnde Teilprobleme zu ermöglichen. Eine solche hierarchische Strukturierung der Schaltungen ist aus demselben Grund wünschenswert, aus dem das Konzept der Modularisierung in höheren Programmiersprachen Eingang gefunden hat: Ein klarer Aufbau, dessen Struktur sichtbar bleibt, macht das Entwerfen weniger fehleranfällig. Zudem ist es einfacher möglich, dass beim Entwurf eines grossen Systems mehrere Personen verschiedene Teile bearbeiten, wenn Einheiten mit genau umschriebenen Schnittstellen definiert werden.

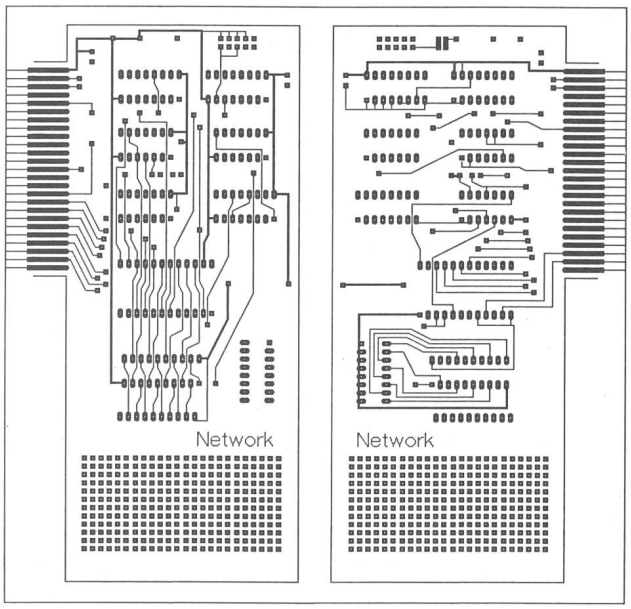

Fig. 1
Eine mit dem VLSI-Editor von Hand entworfene Karte einer asynchronen Schnittstelle als Beispiel für eine gedruckte Schaltung
Links Lötseite,
rechts Bestückungsseite

Es ist wichtig, dass die verschiedenen Komponenten eines CAD-Systems als zusammenhängendes Softwarepaket gestaltet werden. Die zur Verfügung stehenden Programme sollen eine Einheit bilden und untereinander verträglich sein. Darum muss jedes Programm sich an dasselbe Konzept der hierarchischen Strukturierung halten und darf die vom Benützer definierte Unterteilung einer Schaltung nicht wieder verwischen. Das CAD-System von INMOS [2] zeigt, dass heute dieser Grundsatz bei der Software-Entwicklung befolgt wird.

Ebenfalls dem übersichtlichen Aufbau des CAD-Systems dient ein von allen Programmen verwendetes Standardformat, wofür das Austauschformat STIF[3]) ein Beispiel ist [3]. Die Meinung ist nicht, dass einzelne Programme keine anderen Datenstrukturen verwenden dürfen. Die Programme können sogar auf verschiedenen Rechnern laufen. Aber die verschiedenen Datenstrukturen und Fileformate müssen einfach konvertierbar sein. Als gemeinsames Standardformat für den Schaltungsentwurf ist eine symbolische Darstellung auf möglichst hoher Stufe zu verwenden. Als Grundelemente sind etwa Transistoren, Kontakte und Verbindungen («Drähte») zu wählen, welche zwar als Rechtecke dargestellt werden, aber in der Datenstruktur nicht in Rechtecke aufgelöst werden dürfen. Dadurch wird das Überprüfen von Schaltungen (z.B. deren Simulation), die in verschiedenen Darstellungsformen vorliegen, wesentlich vereinfacht.

Schliesslich sollen sich Editoren für verschiedene Darstellungsformen möglichst gleich verhalten, d.h. Befehle, die allen Editoren gemeinsam sind, sollen für den Benützer auch gleich aussehen. Am besten werden die identischen Befehle als gemeinsamer Kern gestaltet. Die übrigen Befehle können je nach Bedarf dazugeladen werden. Mit diesem Mechanismus der Parametrisierung können dem Editor auch andere (z.B. von der Technologie abhängige) Regeln bekanntgegeben werden.

5. Ein VLSI-Editor

Im folgenden wird ein grafischer Editor beschrieben, welcher nach entsprechender Parametrisierung für das Entwerfen von VLSI-Systemen auf der Stufe von Logik-Diagrammen, Stick-Diagrammen und Layout-Diagrammen sowie für das Entwerfen von gedruckten Schaltungen (Fig. 1) geeignet ist.

[3]) STIF = Structured Interchange Format

Ein Editor ist ein Programm mit ausgeprägter Interaktivität. Deshalb ist ein Arbeitsplatzrechner speziell für diesen Anwendungsbereich geeignet. Im vorliegenden Fall wird als Hardware der Lilith-Rechner [4] verwendet. Dieser weist als wesentliche Merkmale neben einer zentralen Recheneinheit einen Speicher mit 128 kWort (zu 16 Bit) und eine auswechselbare Disk mit einer Kapazität von 10 MByte auf. Die Maschine ist mit einem Bildschirm ausgestattet, der für Text und Grafik gleichermassen geeignet ist. Zusätzlich zu einer Tastatur ist als Eingabegerät eine «Maus» mit drei Funktionsknöpfen angeschlossen. Das Positionieren auf dem Bildschirm geschieht durch Bewegen der Maus auf dem Tisch, was von der Software in Bewegungen auf dem Bildschirm übersetzt wird. Die meisten Befehle werden durch die drei Funktionsknöpfe ausgelöst, entweder direkt beim Betätigen derselben oder dann über eine flexible Menütechnik.

Die Software für den Lilith-Rechner wird in der Programmiersprache Modula-2 [5] geschrieben. Diese Sprache unterstützt durch das darin enthaltene Modulkonzept den modularen Aufbau und die Erweiterbarkeit von Programmen, und sie erleichtert das Zusammenarbeiten mehrerer Leute bei der Entwicklung grosser Softwarepakete.

6. Sicht des Benützers

Der Benützer kann (und soll) seine Diagramme in *Module* zerlegen; das sind Teilbereiche, welche wiederum aus Teilen zusammengesetzt sein können. Umgekehrt kann eine Anzahl (nicht überlappender) Module zu einem neuen Modul zusammengefasst werden. So ergibt sich eine hierarchische Unterteilung von der Form einer Baumstruktur. Es ist möglich, die hierarchische Unterteilung eines Diagramms im Verlaufe des Entwerfens zu ändern und neu zu definieren. Die Schnittstelle der Module besteht aus einem begrenzenden rechteckigen Rahmen, der entweder nicht sichtbar ist oder aber hervorgehoben werden kann, und aus den Anschlussverbindungen. Dank der hierarchischen Struktur der Diagramme ist es möglich, verschiedene Stufen der Abstraktion darzustellen, indem von einzelnen Modulen nur die Umrisse oder aber auch alle Details gezeigt werden (logical zooming). Von Modulen können Kopien hergestellt werden, die aus einem einzelnen Exemplar oder aus einer ganzen Reihe (Array) von mehreren Modulen bestehen.

In einer *Bibliothek* hat der Benützer eine Sammlung von Modulen zur Verfügung, die entweder im System vordefiniert sind oder die er selber entworfen hat (Fig. 2). Beispiele solcher Bausteine sind NAND-Gates, Flip-Flops, Schieberegister oder Speicherzellen. Beim Abrufen von Bibliotheksmodulen können allenfalls noch einige Parameterwerte angegeben werden, um eine geeignete Dimensionierung zu erreichen. Zum Beispiel kann ein Modul in verschiedenen geometrischen Ausdehnungen und Formen verfügbar sein.

Um die hierarchische Unterteilung eines Diagramms übersichtlich darzustellen, kann der Bildschirm in mehrere Bereiche (sog. *Fenster*) aufgeteilt werden, in welchen verschiedene Ausschnitte eines Diagramms unabhängig voneinander behandelt werden können. Die Fenster werden bei Bedarf neu erzeugt, verändert oder wieder gelöscht. Der in einem Fenster sichtbare Bildausschnitt kann über das Diagramm hinweg verschoben werden.

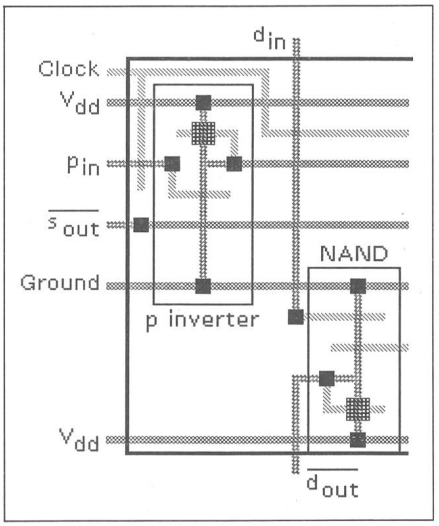

Fig. 3 Vergrösserter Ausschnitt aus Figur 2b

Um das Entwerfen möglichst angenehm zu gestalten, hat der Benützer die Möglichkeit, den Ausschnitt eines jeden Fensters einzeln mit einem Vergrösserungsfaktor zu strecken (Fig. 3). Dadurch erkennt er Details besser und kann mit der Maus genauer positionieren. Ebenfalls dem genauen Positionieren dient die Einrichtung eines (veränderbaren) Rasters auf dem Bildschirm, welches die Auflösung beim Eingeben von Koordinaten bestimmt.

Die verschiedenen *Schichten* (Metall, Polysilicon, Diffusion usw.), die in den verschiedenen VLSI-Technologien nötig sind, werden auf dem Bildschirm durch verschiedene Muster

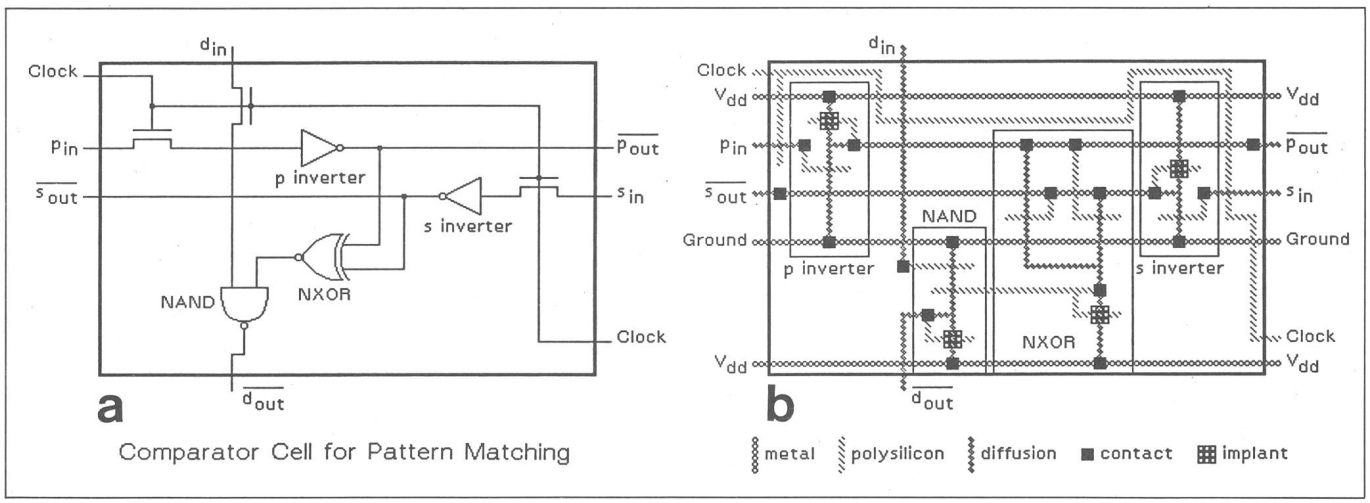

Fig. 2 Beispiel eines Moduls, mit dem VLSI-Editor, dargestellt als Logik-Diagramm (a) und als Stick-Diagramm (b) [6]

oder Farben (falls ein Farbbildschirm vorhanden ist) dargestellt.

Der Editor kennt als einfache *Arten von Objekten* Linien und Polygone (zusammengesetzte Linien), die immer einer bestimmten Schicht angehören. Die Linien können horizontal, vertikal oder in einem Winkel von 45 Grad gezeichnet werden und sind als Rechtekke dargestellt. Kompliziertere Objekte sind verschiedene Typen von Transistoren und Kontakten, die sich über mehr als eine Schicht erstrecken.

Einige *Parameterwerte* für das Zeichnen von Objekten sind global wählbar und bleiben solange gültig, bis sie neu definiert werden. Beispielsweise wird die Schichtzugehörigkeit von Linien und Polygonen auf diese Art gewählt. Ihre Breite kann für jede Schicht getrennt gesetzt werden.

Das Bestimmen auf dem Bildschirm sichtbarer Positionen und das *Auswählen* bereits gezeichneter einzelner Objekte oder ganzer Bereiche von Objekten geschieht, indem man mit der Maus die entsprechenden Stellen bezeichnet. Die so ausgewählten Objekte dienen dann verschiedenen Operationen als implizite Argumente.

Es stehen dem Benützer verschiedene *Grundoperationen* für das Erzeugen, Plazieren und Abändern von Objekten zur Verfügung, welche durch einfaches Betätigen der Mausknöpfe direkt ausgelöst werden können. Unter diesen Grundoperationen befinden sich das Zeichnen von Linien, Polygonen, Transistoren oder Kontakten sowie das Verschieben des gegenwärtig sichtbaren Bildausschnittes. Ebenso einfach werden zuvor ausgewählte Objekte verschoben oder kopiert. Einzelne Objekte (insbesondere Linien oder Polygone) können gestreckt (d.h. verlängert oder verkürzt) werden, indem vorher ein Ende des entsprechenden Objektes ausgewählt wird. Zu den Grundoperationen ist schliesslich noch das Eingeben von Text (für Beschriftungen und Kommentare) zu zählen.

Weniger häufig auszuführende Operationen sind dem Benützer über verschiedene *Menüs* zugänglich. So können ausgewählte Objekte wieder gelöscht werden, oder sie können (horizontal oder vertikal) gespiegelt oder (in Einheiten von 90 Grad) gedreht werden. Die einzelnen Schichten können selektiv auf dem Bildschirm gelöscht oder wieder sichtbar gemacht werden. Es kann verlangt werden, dass Ausschnitte des erstellten Diagramms aufbereitet werden zum anschliessenden Drucken oder Plotten.

7. Wahl der Datenstruktur

Im folgenden wird der Aufbau der *Datenbank* für den beschriebenen VLSI-Editor gezeigt. Da auf dem Lilith-Rechner zuwenig Speicherplatz zur Verfügung steht, um die ganze Beschreibung grosser Diagramme im Speicher zu halten, ist eine Struktur für die Daten auf der Disk gesucht worden, welche auch für den interaktiven Betrieb geeignet ist. Im wesentlichen geht es darum, grosse Mengen von Objekten (nämlich Linien, Polygone, Transistoren oder Kontakte) so zu strukturieren, dass genügend schnell auf einzelne dieser Objekte zugegriffen werden kann.

Die gewählte Datenstruktur beruht auf dem Prinzip der sog. *Grid-Files* [7]. Der Wertebereich verschiedener Attribute, welche gleichberechtigt sind und symmetrisch behandelt werden, bildet einen mehrdimensionalen Suchraum. Die Grundidee besteht in der Gitter-Teilung (daher die Bezeichnung Grid-Files) dieses Suchraumes. Der Zugriff zu den Objekten erfolgt über das Gitter-Directory, in welchem die Veränderungen der Gitter-Teilung eingetragen werden.

Die Gitter-Teilung liefert dann gute Resultate, wenn die Objekte durch eine *kleine Anzahl von Attributen* charakterisiert sind, deren *Wertebereich gross* ist. Da die darzustellenden Objekte alle auf Rechtecke mit zusätzlichen Attributen zurückgeführt werden können, kommen für die vorliegende Anwendung die horizontale und vertikale Koordinate sowie Breite und Höhe von Rechtecken als Charakterisierung in Frage. Natürlich besitzen die Objekte noch weitere Attribute, die aber nicht zum Aufbau des Suchraumes beitragen.

Diese Datenstruktur ist für *häufige Änderungen* geeignet, weil sich die Gestalt der Filestruktur beim Einfügen und Löschen von Objekten jeweils der neuen Situation anpasst. Sie bietet sowohl *schnellen Zugriff* zu einzelnen Objekten als auch effizientes Behandeln von Abfragen für ganze Bereiche. Es ist also einfach, alle Objekte zu bestimmen, die in einem bestimmten Ausschnitt eines Diagramms liegen. Zu den weiteren Eigenschaften gehört die gute Speicherausnützung (70%), die unabhängig von der Verteilung der Daten ist.

8. Direktes Prüfen der Entwurfsregeln

Für jeden Fabrikationsprozess einer bestimmten Technologie gibt es *Entwurfsregeln*, die dem Entwerfen von Diagrammen geometrische Einschränkungen auferlegen, um eine problemlose Fabrikation von funktionierenden Transistoren und Verbindungen zu garantieren. Zu den Entwurfsregeln gehört eine Anzahl von Regeln über die Abmessungen (etwa die minimale Breite) jedes Objektes und eine Anzahl von Regeln über die minimalen Abstände zwischen den Objekten [1, Kap. 2.6].

Das *Prüfen der Entwurfsregeln* (design rule checking) wird bis heute üblicherweise mit einem separaten Programm nachträglich erledigt. Ein solches Programm ist mit einem grossen Aufwand verbunden, weil zu jedem Objekt alle Objekte der unmittelbaren Umgebung von neuem berechnet werden müssen. Darum ist es viel geschickter, das Prüfen der Regeln direkt beim Plazieren von Objekten vorzunehmen, weil dann die unmittelbare Umgebung ohnehin bekannt und auf dem Bildschirm sichtbar ist. Zu diesem Zweck werden die verschiedenen Regeln dem Editor bei dessen Parametrisierung bekanntgegeben.

Für das *Einhalten der Entwurfsregeln* stehen dem Benützer des VLSI-Editors Hilfen zur Verfügung. Wenn ein Objekt neu erzeugt oder verschoben wird, prüft der Editor, ob alle Regeln eingehalten worden sind. Ist dies der Fall, so wird das Objekt auf dem Bildschirm gezeichnet. Ist jedoch eine der Regeln verletzt worden, so wird das Objekt provisorisch gekennzeichnet, indem es lediglich gestrichelt gezeichnet wird. Falls der Benützer nicht sicher ist, kann er nun abfragen, welche der verschiedenen Regeln verletzt worden ist und welches von den benachbarten Objekten die Regelverletzung verursacht hat. Zudem kann er sich zeigen lassen, wie nahe an das betreffende Objekt irgendein anderes Objekt (etwa eine Linie auf einer bestimmten Schicht) gesetzt werden darf.

9. Schlussbetrachtung

Der beschriebene grafische Editor ist vergleichbar mit neueren interaktiven Editoren für das Entwerfen von VLSI-Chips, z.B. mit dem Programm ICARUS von Xerox, das in [1, Kap.

4.4] beschrieben ist, oder mit dem Hierarchie-Editor HED und dem Entwurfsprogramm FatFreddy von INMOS, beschrieben in [2]. Er ist Teil eines integrierten CAD-Systems für den VLSI-Entwurf, welches am Institut für Informatik der ETH Zürich entwickelt wird. Mit diesem VLSI-Editor soll insbesondere gezeigt werden, dass ein Arbeitsplatzrechner, der über einen begrenzten Speicherausbau, dafür aber über eine eigene Disk verfügt, für die Verarbeitung grosser Datenmengen genügt.

Literatur

[1] *C. Mead* and *L. Conway:* Introduction to VLSI systems. Reading/Massachusetts, Addison-Wesley, 1980.

[2] Working material for the advanced course on VLSI architecture. Bristol, University of Bristol, 1982.

[3] *C. Séquin:* Standard interchange formats for integrated circuit design. Berkeley, University of California, 1980.

[4] *N. Wirth:* The personal computer Lilith. In: Microcomputer system design. Lecture notes in computer science No. 126. Berlin/Heidelberg/New York, Springer, 1981.

[5] *N. Wirth:* Programming in Modula-2. Berlin/Heidelberg/New York, Springer, 1982.

[6] *M. J. Foster* and *H. T. Kung:* Design of special-purpose VLSI chips. Example and opinions. (AD-A-081 451). Pittsburgh, Carnegie-Mellon University, Department of Computer Science, 1979.

[7] *J. Nievergelt, H. Hinterberger* and *K. C. Sevcik:* The gridfile: An adaptable symmetric multi-key file structure. Berichte des Institutes für Informatik No. 46. Zürich, ETH, 1981.